# A Network Measurement Tool for Handheld Devices

SiewYeen Agnes Tan

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Dr. Scott F. Midkiff, Chair
Dr. Ira Jacobs
Dr. Yao Liang

May 16, 2003
Blacksburg, Virginia

Keywords: Pocket PC, network measurement tool, Windows CE, Wireless LANs

**A Network Measurement Tool for Handheld Devices**

SiewYeen Agnes Tan

# Abstract

This thesis describes a performance measurement tool that allows a user to measure network performance using a handheld device. The measurement tool consists of a client program that runs on a Microsoft Pocket PC device and a server program that runs on a regular Microsoft Windows computer. Both programs are Windows applications implemented in C/C++ using the Microsoft Embedded Visual Tool and Microsoft Visual Studio. The use of a Pocket PC device provides mobility to users, which can save time and energy when performing experiments. The thesis describes the design of the performance measurement application, implementation issues, and tests conducted using the tool.

# Acknowledgements

First of all, I want to thank my parents for their supports throughout my school years at Virginia Tech. Then, I want to thank Dr. Midkiff for working as my advisor and giving me wise counsel during the past two years of my graduate school. I also want to thank Dr. Jacobs and Dr. Liang for being my committee members and for their insightful inputs. Also, thanks to everyone who have helped me in my research. To all my friends, thanks for the good times and support that you have given me over the years. All the best to all of you!

I also wish to acknowledge the Office of Naval Research for financial support during Fall 2002 through the Navy Collaborative Integrated Information Technology Initiative (NAVCIITI).

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1. Introduction

Advances in technology have made handheld devices extremely efficient and flexible tools that are widely used for many applications. The primary advantage of using a handheld device is that it can help users save time by enabling mobility. In addition to their obvious convenience, handheld devices can also support reasonably complex database applications and web applications that can connect over a wired or wireless network.

Network measurement tools are needed to maintain the health and efficiency of a network. Most network measurement tools run on traditional computers using the Microsoft Windows operating system, Linux, or some version of Unix. As handheld computers using Microsoft Windows CE becomes increasingly popular, it is an obvious choice to use them for network measurement tools. The main purpose for developing the network measurement tool described in this thesis is to provide mobility. The tool consists of a client program and a server program that are used to measure network throughput. Both programs are written in C/C++ using Microsoft Embedded Visual Tools and Microsoft Visual Studio. The client runs on a Windows CE device and the server runs on a regular Windows computer. Almost all Windows CE devices come with a PC Card or other network adapter that allows users to connect to an Ethernet or a wireless local area network (LAN). Using the measurement tool, users can check network reachability and measure network throughput. Besides providing mobility, the tool is relatively easy to use. In the client configuration dialog box, users can specify all the necessary information and do not have to memorize command options as with most console applications.

Experiments were done to verify the operation of the tool and the measurement results are reported. The experiments consist of measuring the throughput of the on-campus wireless network at Virginia Tech within three campus buildings and an off-campus network. The performance of three transport protocol variations, UDP, TCP, and TCP with the TCP_NODELAY option enabled, is observed with and without a virtual private network (VPN) connection. The results are analyzed to conclude that the tool functions as expected.

Chapter 2 describes some existing measurement tools and discusses the general problem of measuring network performance. It also describes development tools for Pocket PC devices and procedures for developing a Pocket PC application.  Chapter 3 defines the objective and approach for the development of the network tool. Chapter 4 describes the general design of the client and the server programs. Chapter 5 defines the metrics and the experiments. Chapter 6 presents and analyzes the experimental results. Chapter 7 summarizes the overall work and discusses potential future work. There are three appendixes. Appendix A provides instructions on how to run the tool, Appendix B describes the programming details for the client and the server, and Appendix C shows all measurement results in tabular form.

# Chapter 2. Background

## 2.1 Introduction

Network performance is the ability of the network to deliver data from specific applications and is commonly expressed as delay, throughput, packet loss, utilization, etc. Tools are available for developers, network administrators, and researchers to measure performance over many different networking arrangements in an IP network.

There are a number of factors that can affect network performance. For example, the TCP buffer size, incorrectly configured network equipment such as routers and switches, system limitations, and congested network conditions at routers are common causes of degradation in performance. A variety of measurement tools are available for testing, monitoring and measuring performance. These tools can be used to gather network data used for analyzing network performance in terms of latency, packet loss, throughput, etc. However, most of these tools have some limitations, such as targeting only a specific problem or platform, not locating and distinguishing bottleneck links, not measuring high-speed link bandwidth accurately, and not providing a network analysis mechanism. The next section introduces and describes different network tools that are available for measuring backbone and end-to-end network performance.

Section 2.3 introduces the Windows CE operating system and development tools for developing Pocket PC applications. Windows CE provides a subset of the Win32 application program interface (API) and is designed for embedding in mobile and other space-constrained devices. As the handheld market grows, many tools for personal digital assistants (PDAs) are emerging due to their ease of use. Examples of PDA-based network tools include PocketLanCE [1] that allows Pocket PC users to access, browse, and manage LAN resources from their PDA devices, Sniffer Wireless PDA [2] that pinpoints security breaches and checks all access points to ensure that Wired Equivalent Privacy (WEP) is enabled, and POCKETSAFE [1] for management and secure storage of passwords and confidential information. Pocket PC is an operating environment for handheld computers evolved from the Microsoft Windows CE operating system. Development tools such as Microsoft Embedded Visual Tools 3.0, Visual CE, and PocketC are used to build

Pocket PC applications. Section 2.4 gives a tutorial on developing, building, and installing a simple Windows CE application using Microsoft Embedded Visual C++ 3.0.

## 2.2 Measuring Network Performance

Network performance can be measured within a wide area backbone network or for an end-to-end system. Measuring end-to-end performance does not require any special information about the network when performing network measurement. To measure backbone performance, Simple Network Management Protocol (SNMP) [3] polling and Multirouter Traffic Grapher (MRTG) [4] can be utilized. Ping [5], Traceroute [6], Treno [7], TTCP [8], and Nettimer [9] are commonly used tools for measuring end-to-end network performance. Some tools are intended for use by network engineers, while some are more appropriate for use by researchers. These tools range from evaluating parameters of TCP to investigating deficiencies in routers and the operating system's TCP implementation. TTCP and Nettimer are examples of application-level software tools.

Tools based on the Internet Control Message Protocol (ICMP) [10], such as Ping and Traceroute, have two main problems, loss asymmetry and ICMP filtering. Since packets flowing in each direction are different and may take different routes, the packet loss rate on the forward path to a target maybe different from the packet loss rate on the reverse path from a target. The ability to measure loss asymmetry is an important factor to locate and troubleshoot a network bottleneck. ICMP-based tools do not provide enough information to determine if a packet was lost or if the response was lost. Furthermore, the malicious use of ICMP services is a major reason that many networks filter ICMP packets and limit the rate of ICMP responses. ICMP spoofing by firewalls and load balancers have also precluded users from performing real end-to-end network measurement [11].

### 2.2.1 SNMP and MRTG

SNMP is an application-layer protocol designed to facilitate the exchange of management information between network devices [3]. Network administrators use SNMP to collect data, such as packets sent or received per second and network error rates, monitor and manage network performance, find and solve network problems, and plan for network growth. SNMP works by sending messages, called protocol data units (PDUs), to different parts of a network. Its compliant devices, called agents, store data about themselves in Management Information Bases (MIBs) and return the data to the SNMP requesters. SNMP gathers statistics from routers and switches, including the number and size of IP packets, total bytes, router CPU utilization, and discarded packets.

MRTG is an advanced tool written by Tobias Oetiker and Dave Rand [4] used to monitor the traffic load on network links and to display network traffic over time. Basically, MRTG uses SNMP to fetch counters from devices, such as routers, to calculate and store the rate and to display graphs from the stored data. Most routers have a counter in them and the counters will increase whenever there is traffic passing through them. Thus, MRTG will read and store the counter every five minutes to obtain the amount of traffic that was sent through the router. The stored data is used to generate graphs that represent the traffic load on the monitored network. These graphs are then embedded into Hyper Text Marking Language (HTML) pages that can be viewed from any web browser. Since MRTG logs all data that it has pulled from the routers, it also creates visual representations of the traffic seen during the last seven days, the last five weeks, and the last twelve months.

The SNMP implementation is coded entirely in Perl and the main program is written in C to speed up the logging process and the generation of GIF images. The graphics are generated with the help of the GD Graphic library from Thomas Boutell. MRTG with SNMP polling is a simple but powerful tool to monitor backbone and router performance. However, users have to be aware when configuring MRTG that SNMP might send out so many request packets that it can overload the network and degrade network performance.

### 2.2.2 Ping

Ping is the most common utility used to validate the accessibility of an IP address and to troubleshoot Internet connectivity. The name “Ping” is sometimes claimed to be an acronym that stands for Packet InterNet Gopher, but the author of Ping based the name on a sonar analogy [5]. Ping sends an ICMP echo packet to the specified address and waits for a reply. Upon receiving the ICMP echo reply packet, the Ping program displays the round trip time, the sequence number of each packet, and checksums. A round-trip time (RTT) is calculated as the difference between the time the echo request is sent and the time a matching response is received [12].

Routers are not considered good measurement targets because they are optimized for the relatively simple task of forwarding packets. The Ping application that measures network connectivity at the router level cannot give reasons why packets are damaged, delayed, or duplicated. ICMP packets are often filtered and blocked at many routers for security reasons and there is no guarantee that the ICMP packet will reach the destination because some datagrams may still be undelivered without any report of their loss. As a consequence, users may not know whether the forward path link or the reverse path link is down or degraded and no further information is provided about packet loss. Therefore, higher level protocols that use IP must implement their own reliability procedures if reliable communication is required. Ping is a simple program that sends a fixed amount of data (e.g., 32 bytes) to the destination. As users can only configure the target information, Ping is not a sufficient tool for measuring network performance.  To analyze network connectivity in more depth, more control functions, such as the total bytes to send, are added.

### 2.2.3 Traceroute

The original Traceroute[1] algorithm was developed by Van Jacobson and Steve Deering [6]. It is a commonly used diagnostic tool that sends ICMP packets with varying time-to-live (TTL) values to specify how many hops the packets are allowed before they expire and an error message is returned to the sender. The TTL field of the packets contains an integer value that is decremented as the packet passes through a router to its destination. Traceroute traces packets from a source host to the target host by showing a hop-by-hop listing of each router along the path to the target host, the estimated latency of each router, and an asterisk ("*") or other indication if there was no response. When a packet cannot reach its destination because the TTL value is too low or a zero TTL value is detected, the packet is considered to have expired. The IP specifications state that

[1] Traceroute source code and documentation are available at http://ncne.nlanr.net/software/tools/connection.html.

when a packet is expired, it is returned to the source with an ICMP message of "time exceeded in transit" and the address at which the timeout occurred.

Path measurement using Traceroute requires many packet probes per path that cause a major drawback. As the number of active measurement activities increases in a network, a significant amount of network resources are consumed and a heavy load on the network may result. Thus, the number of packet probes added to the network should be as few as possible to avoid disturbance of the normal network operations. Also, interpreting Traceroute's output accurately is not an easy task and requires experience.

### 2.2.4 Treno

Treno[2] is an Internet throughput measurement tool based on a user-level implementation of a TCP-like protocol based on TCP Reno [7]. It accurately measures the bulk transfer capacity of network links by implementing its own TCP algorithm with selective acknowledgements (SACK). Users can measure the throughput of a given link independently of the TCP implementation of end hosts and it serves as a useful platform for prototyping TCP changes. Treno probes the network with either ICMP ECHO message, as in Ping, or low TTL UDP packets, as in Traceroute, that solicits ICMP errors. The probe packets and the replies only rely on standard portions of the IP and ICMP protocols. Each packet carries sequence numbers that are reflected in the replies, so that Treno can always determine which probe packet caused each response. Treno uses the sequence numbers to emulate TCP, performing an idealized version of the *de facto* standard TCP congestion control algorithms, such as slow start and congestion avoidance methods. Treno also allows targeting individual routers along the path to discover problematic links.

The Treno tool has some limitations of which users need to be aware, especially when using it as a server in performance experiments. Treno does not support multiple client connections because it is a single-threaded program. Since the slow start algorithm is implemented in Treno, running tests that are too short in time may not obtain valid results. Thus, each hop of the test has to run

[2] A version of Treno can be found at http://ncne.nlanr.net/software/tools/tcp.html.

for at least 10 seconds to reach equilibrium and to get accurate bandwidth measurements. Furthermore, some routers in the network may not respond to Treno probes as fast as they forward other packets. This might affect the accuracy of bandwidth measurements.

### 2.2.5 TTCP

Test TCP (TTCP[3]) is a command-line sockets-based benchmarking tool for measuring TCP and UDP performance between two systems [8]. The US Army Ballistics Research Lab first created TTCP for the Berkeley Software Distribution (BSD) operating system in 1984.Users can employ TTCP by running a copy of TTCP in receive mode and a second copy in transmit mode. The results for the data transfer from the transmitter to the receiver represent the approximate performance of the path between the source and destination hosts. By selecting the source and destination at various points within the network, one can analyze critical portions of the network path. Since TTCP is an open source program, many versions of TTCP have been developed over the years, such as PCATTCP [8], NTTCP [13], NUTTCP [13], and Iperf [14]. Among the different versions of TTCP, Iperf is a well-known application for testing network bandwidth developed by the National Laboratory for Applied Network Research (NLANR). It measures the maximum TCP bandwidth and the UDP performance between two nodes in a network without monitoring intermediate nodes and routers. Iperf can also measure packet loss and jitter and is a useful tool for non-network engineers.

TTCP allows users to control the transport protocol used, length of buffer read or written to the network, port number and more. Upon completion of the test, TTCP reports the amount of data transferred, the transfer time, and the approximate throughput. The throughput is calculated using the time and the amount of data transferred between the sender and the receiver. To check whether the network is functioning as expected, users can compare the actual throughput with the theoretical bandwidth. Variations in throughput performance might be caused by overloaded network equipment, such as routers, other significant traffic in the network, or transmission errors that caused packets to be corrupted or dropped. In addition, users can also perform TTCP

[3] A version of the TTCP can be found at http://www.ccci.com/tools/ttcp/index.html.

tests on each individual section of the path to resolve which links or devices may have affected the network throughput.

TTCP has an advantage over some measurement tools such as using FTP to report performance. In a high performance network, it is difficult for any single computer to transfer data to or from a disk at rates that are sufficient for real network testing. TTCP achieves high performance by filling a memory buffer with data, then repeatedly transmitting the data. It acts as a traffic transmitter and receiver that can operate at true network speeds [8]. It has also become a standard UNIX networking tool. In addition to performing its intended function of testing TCP performance from user-memory to user-memory, TTCP is an excellent tool for bootstrapping hosts onto the network by providing a UNIX "pipe" between two machines across the network. While TTCP is one of the most commonly used tools for network performance testing, it has some significant drawbacks and limitations. Most of the TTCP implementations are inefficient, include circuit setup and teardown time in the calculation of throughput, and only allow one kind of network traffic. Furthermore, it is difficult to extend the tool to permit creation of a large-scale experiment that can be reproduced. Furthermore, most of the measurement tools, including TTCP, do not provide a graphical user interface (GUI) to users. With a GUI implementation, the tool would be easier to use, especially for infrequent or new users.

### 2.2.6 Nettimer

Lai and Baker have implemented Nettimer that estimates link-level bandwidth by passively collecting network traffic and actively probing the network [9]. Nettimer measures capacity on a hop-by-hop basis by performing a packet-pair "tailgating" method. Packet tailgating captures link-specific characteristics by causing queuing of packets at some particular links. This method defines a very large packet followed by a train of smaller identical packets, which are still large enough to queue at the bottleneck. For each link, the tool sends a large leading packet with a time-to-live (TTL) value set to expire at that link, followed by very small packets that will queue continuously behind the large packet until the large packet expires [9]. Thus, the following packet train is set to measure bottlenecks from that particular hop and to the end of the link. The collected data is output using the "ns" database format. The packet-pair tailgating method consumes less network bandwidth than ICMP-based tools since it does not rely on consistent router behavior for handling ICMP packets and does not rely on timely delivery of acknowledgements. This method can theoretically detect multi-channel links and can be run on multicast trees.

Potential Bandwidth Filtering (PBF) is employed to filter small and slowly sent packets, such as TCP acknowledgements, that can mislead the current packet-pair implementation. The PBF method correlates the potential bandwidth and the measured bandwidth of a sample in deciding how to filter packets. The Nettimer tool also employs the Receiver Only Packet Pair (ROPP) algorithm that measures bandwidth without deployment of special software at two nodes.

However, the Nettimer tool also has some disadvantages. Although the PBF method that can handle all packet sizes and rates, this passive technique, by its nature, is limited to a local statistical average for packet sizes that is proven to vary widely on the Internet. Implementation using the ROPP algorithm also sacrifices the accuracy of bandwidth measurement a little since ROPP only takes timing measurements at the receiver site. To generate accurate bandwidth estimations, Nettimer requires a relatively large number of measurement packets to be sent. Theoretically, it can be used to estimate queuing delays, but there has been no experimental work exploring its accuracy. The number of measurement packets needed for estimation may prove prohibitive given the short duration over which delay distributions are generally stable [15]. To achieve more reliable bandwidth estimations, low network utilization seems to be needed. Therefore, Nettimer is not yet a suitable tool for measuring performance of a network that usually has high utilization and more experimental work needs to be done.

## 2.3 Development Tools for Pocket PC Applications

Our interest is to develop a networking tool for handheld devices, specifically for devices using the Microsoft Windows CE operating system. Windows CE is a 32-bit lightweight, multitasking, multithreading operating system with an optional graphical user interface designed for small devices. Although Microsoft does not explain the “CE” abbreviation, it is reported to have originally stood for “Consumer Electronics” [16]. Windows CE supports multiple platforms such as Palm-size PC and Pocket PC. Microsoft introduced Palm-size PC in January 1998 and Pocket

PC in April 2000. Pocket PC is an enhanced platform evolving from the Windows CE 3.0 operating system, which was optimized for better real-time support, rich multimedia functionality, enhanced Internet capabilities, and advanced application services. Microsoft has also released the Windows CE 3.0 Add-On Pack [17] that provides improved multimedia support through a media player control, XML support, and new networking and communication support with the Point-to-Point Tunneling Protocol (PPTP), Internet Connection Sharing (ICS), and the Remote Desktop Protocol (RDP). These features allow a Windows CE-based device user to remotely access a corporate network in a secure manner, to share its Internet connection among one or more devices running on the same network, and to enable Windows CE-based terminals to work seamlessly with a Windows 2000 terminal server.

Pocket PC devices can be built from different processors that come from other manufacturers, such as the MIPS processor found in Casio devices, the SH3 processor used in HP Jornada devices, and the Strong ARM processor used in Compaq iPAQ devices. A processor-neutral compiler engine uses the Common Executable Format (CEF) [16] technology that enables developers to build portable solutions that work seamlessly across the various Pocket PC devices regardless of the processor used. When compiling a CEF application, the compiler and linker build the executable files that contain intermediate language instructions instead of machine-specific code instructions. When the user installs a CEF application on a Windows CE device, a translator generates the corresponding native machine code for that device. CEF does not build applications for running on different devices or platforms. For example, a CEF executable file built for the Pocket PC will run on any Pocket PC device, but not on a Handheld PC. CEF support is available on the Pocket PC platform and any Windows CE platform created with Microsoft Windows CE Platform Builder 3.0 [18]. Currently, Windows CE supports the ARM, MIPS, PPC, SHx, and x86 processors. The latest version of the Pocket PC platform is Pocket PC 2002, which provides an enhanced user interface, Pocket Office (Inbox, Internet Explorer, Word and Excel), a handwriting recognition program, an e-book reader, Microsoft Media Player, and longer battery life. These tools provide a developing environment for embedded developers to create Pocket PC applications.

Visual C++ and Visual Basic are the most common languages used to build Pocket PC applications, although other languages, such as Java, Perl, Lisp, and Forth, also support Pocket PC. Software packages are available either commercially or non-commercially for Pocket PC developers. Microsoft Embedded Visual Tools 3.0 [19] is an example of freeware, while Visual CE [20], NS Basic/CE [21], and PocketC [22] are commercial tools for Pocket PC development.

### 2.3.1 Microsoft Embedded Visual Tools 3.0

Microsoft Corporation offers a freeware tool set for developers, Microsoft Embedded Visual Tools version 3.0. It is a set of tools for embedded developers to use to build software applications for the new generation, 32-bit devices based on the Microsoft Windows CE operating system. Embedded Visual Tools 3.0 includes full versions of Microsoft Embedded Visual C++ 3.0, Embedded Visual Basic, and Software Development Kits (SDK) for the Pocket PC, Palm-size PC, and Handheld PC [19]. Embedded Visual Basic is tailored for high productivity as it delivers many of the benefits offered by Visual Basic on the desktop. Embedded Visual C++ produces native machine code, such as MIPS or x86 code, applications for handheld devices and is optimized for faster and easier development. In addition, it also supports the CEF tool that generates code and applications that can be installed and run on any of the supported Windows CE processors for a particular device.

Developers can build Windows CE applications by using a standalone integrated development environment (IDE), as well as construct enterprise solutions with data-access capabilities through ActiveX Data Objects for Windows CE. An emulator for handheld devices is also implemented in the tool for debugging purposes. Programmers can setup a connection to the target device either through Microsoft ActiveSync, PPP Transport, or TCP/IP Transport for Windows CE to automatically download a copy of the executable file to the device. According to a survey of the Pocket PC Developer Network group, more than 95% of Pocket PC developers use this package [23].

### 2.3.2 Visual CE

Visual CE from Syware Institute Database Technology Corporation [20] is one of the most widely used commercial tools for building database applications for Windows CE or Pocket PC devices. It was designed for people who are comfortable with computers, but are not necessarily programmers. Visual CE provides the application development features typically found only in Visual Basic, Visual C++, and similar platforms used by professional programmers.

Visual CE Version 7 is the latest version featuring a flexible array of development tools, including full support for Global Positioning Satellite (GPS) devices, complete picture control, multi-device synchronization, and support for a wide range of numeric, string, and date functions. Macro programming with the drag and drop controls allows users to create electronic forms quickly and easily. The Professional and Enterprise Editions of Visual CE include the mEnable SDK that allows mobile users to easily access and modify Open Database Connectivity (ODBC) server data in real time. Visual CE with mEnable also lets developers and system integrators quickly create data-driven handheld applications with wireless, interactive access to "live" corporate data. With the GPS support feature, Visual CE can read and store positioning information for a variety of field applications that have traditionally relied on the slow, paper-based communication system.

### 2.3.3 NS Basic/CE

NS Basic/CE is a commercial Windows CE development tools developed by NS Basic Corporation [21]. It is a complete BASIC toolkit that allows developers to build applications on a Windows CE device or on a desktop computer. NS Basic/CE uses Microsoft's standard VBScript engine as its core, combined with extensions to create a complete development and runtime environment. It fully supports Pocket PC 2002 handheld devices including interfaces that allow the full feature set of Pocket PC devices to be controlled from a BASIC program. NS Basic/CE also includes features such as database support, serial and infrared communications, signature capture, Internet access through Winsock, and support for ActiveX controls.

### 2.3.4 PocketC

PocketC from OrbWorks [21] is a simplified version of the C programming language, but has a different feature set. Instead of generating machine code like Visual C++, PocketC generates intermediate code that is interpreted by a freely distributable runtime virtual machine. The PocketC compiler produces a “pseudo-executable” file that runs on all Windows CE platforms. Most importantly, even a novice programmer is able to create simple applications without knowing anything about Windows programming [22].

The PocketC toolkit contains two main components, PCeditor and PCshell. PCeditor is the development tool used to write and compile codes, while PCshell is the run-time module used to execute applications created by PocketC. In addition, Windows CE devices can have many different processors and hardware environments. Instead of producing a separate executable for each platform and each processor, PocketC allows programmers to distribute one binary for all Windows CE devices.

### 2.3.5 Other Tools

There are other tools for developing Pocket PC applications such as ANS Forth for Win32 [24], Perl 5.6 for Windows CE [25] and Waba [26]. Both ANS Forth and Perl 5.6 for Windows CE are currently not very stable tools for use by developers. Waba is a free open-source programming platform for small devices produced by Wabasoft. It defines a language, a virtual machine, a class file format and a set of foundation classes. Developers can use Java development tools to develop Waba programs. However, Waba is not a derivative of Java and has no connection to Sun Microsystems. Waba allows developers to quickly build programs on almost any platform with development tools that are inexpensive, familiar and easy to use. It is quite a reliable tool as Waba safeguards memory access to prevent failures, such as memory corruption, and, thus, avoids users needing to reset the device.

## 2.4 Developing Windows CE Applications with Microsoft Embedded Visual C++ 3.0

The environment for developing Microsoft Windows CE applications is different from that for developing standard Microsoft Windows applications because Windows CE programs are written on PCs running Windows 2000/XP/NT and debugged and executed mainly on separate Windows CE-based target devices. Microsoft Embedded Tool 3.0 is chosen to develop the Pocket PC application described in this thesis since it provides a more stable and more reliable environment than other tools. This software consists of the tools described in Section 2.3.1. In a full installation, the whole Embedded Visual Tool package will be installed. In a customized installation, programmers can decide what language and device SDK(s) to install. A typical recommendation is to choose the language that is the most familiar to the programmer. For the work described in this thesis, Embedded Visual C++ 3.0 and the Pocket PC/Pocket PC 2002 SDKs were installed.

The next section provides a tutorial for developing a Windows CE application and downloading the application to the target Pocket PC device using Embedded Visual C++ 3.0.

### 2.4.1 Building Windows CE Applications

After installing Embedded Visual C++ 3.0, programmers can start developing applications as usual by defining a project type and workspace. Since Windows CE does not support DOS, there is no console application available. Before compiling a program, programmers have to select a correct platform and central processing unit (CPU) to build a suitable executable file for the intended target device. For example, iPAQ programmers have to choose either Pocket PC or Pocket PC 2002 as the platform and WIN32 WCE ARM as the CPU for the target device. Figure 1 and Figure 2 show the platform and project configuration settings, respectively, under the "Build" menu. For debugging purposes, one can set the CPU to WIN32 WCE x86em that represents the Pocket PC emulator. Figure 3 and Figure 4 display the Pocket PC 2002 emulator and Pocket PC emulator, respectively.

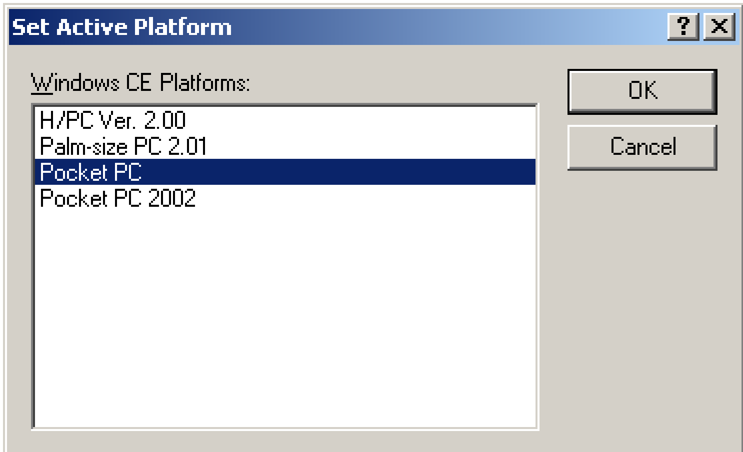



**Figure 1.** Selecting an appropriate platform for the target device under the “Build” menu.

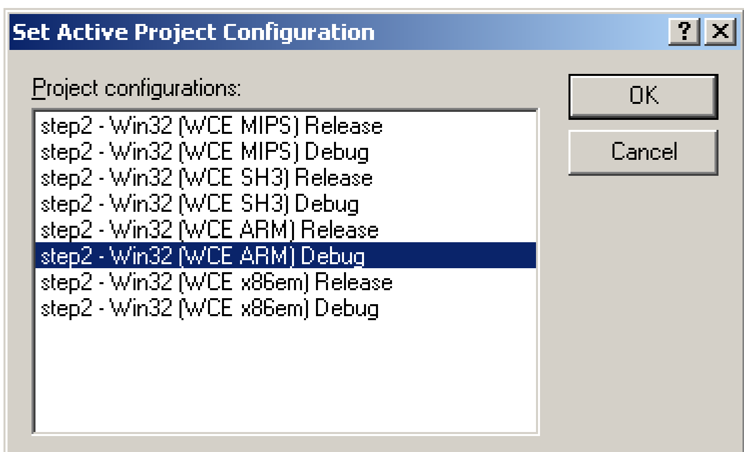



**Figure 2.** Selecting an appropriate CPU and project configuration under the “Build” menu.


**Figure 3.** Pocket PC 2002 emulator.

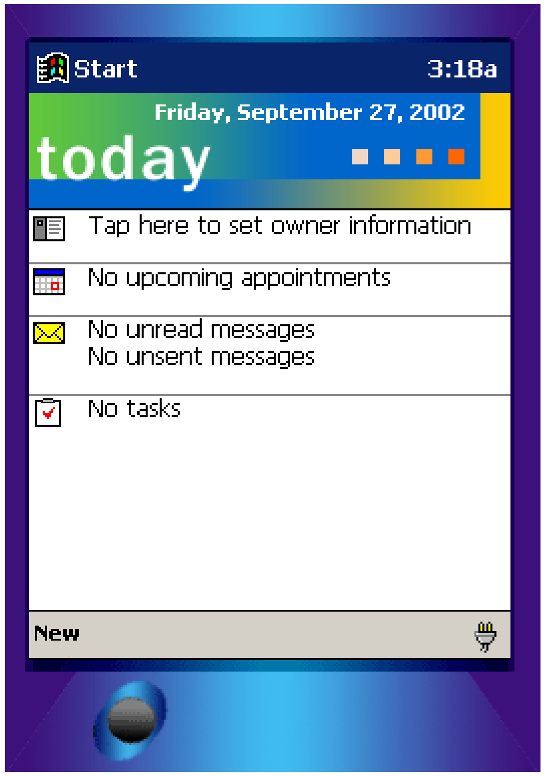



**Figure 4.** Pocket PC emulator.

### 2.4.2 Downloading Applications to the Target Device

After building the application, programmers can download the program automatically or manually to the target device. To download automatically, the user selects the "Windows CE Platform Manager" option under the "Tools" menu as shown in Figure 5 to configure the device properties. There are three types of protocols used to establish the connection from the development computer to the target device, Microsoft ActiveSync, PPP Transport for Windows CE, and TCP/IP Transport for Windows CE. Any of these methods can be used to connect to the device with the correct configuration. To ensure that a connection is established, users can test the connectivity by clicking the "Test" button. The device properties window is shown in Figure 6 and the testing window is displayed in Figure 7. If a connection is made, the "Connection is established" message is displayed. Thus, for every successful compilation, a copy of the executable file is downloaded automatically to the target device. To download manually, the user places the executable file in the Explorer window for the target device. Then, the Microsoft ActiveSync program will transfer the file to the device when synchronizing the two machines.

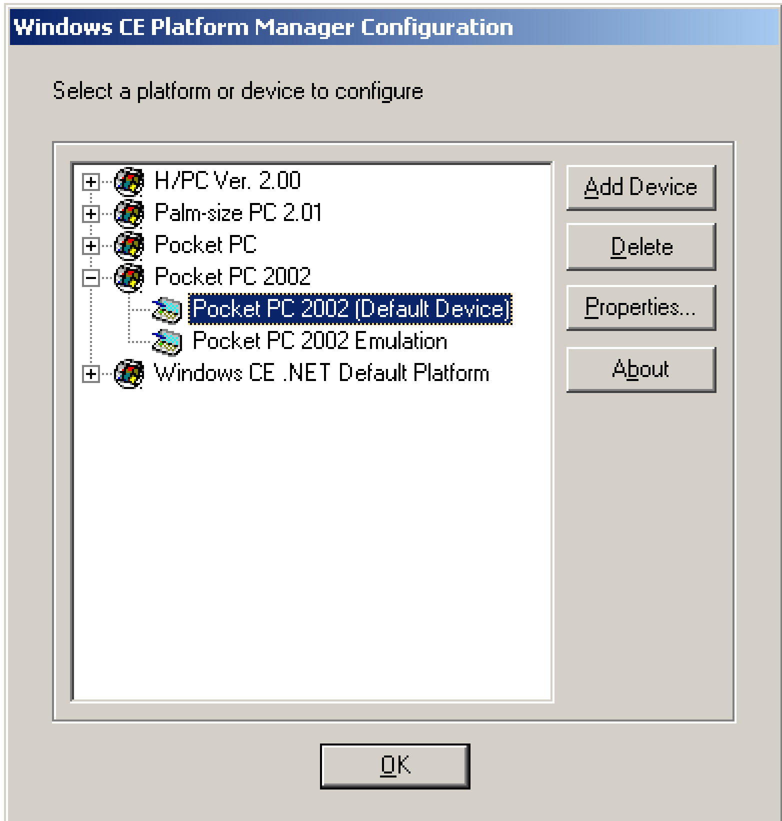



**Figure 5.** Configuring the device's properties under the "Tools" menu.

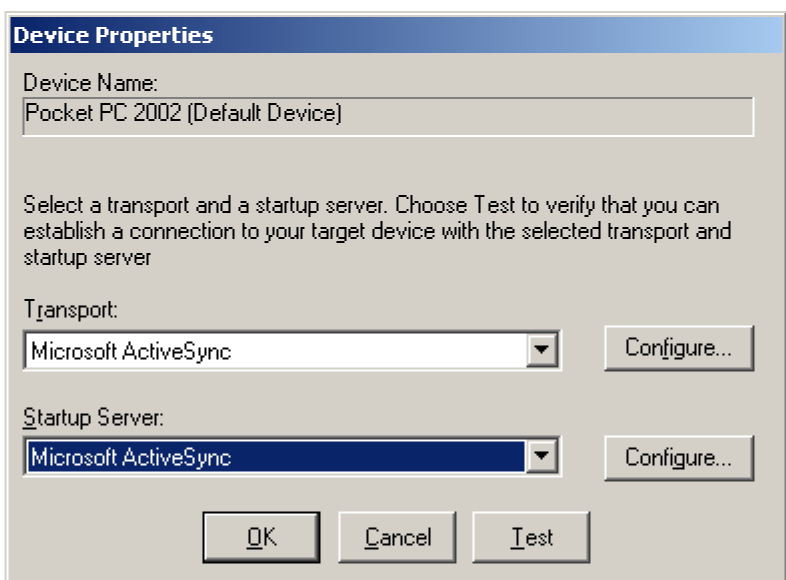



**Figure 6.** Defining transport and setup connection for the device.


**Figure 7.** Testing the connection to the device.

## 2.5 Summary

This chapter described various network measurement tools, provided background information on Windows CE, introduced development tools for Windows CE applications, and presented a tutorial for developing Pocket PC applications especially for the Compaq iPAQ Strong ARM processor. The following chapter gives the objective of the network measurement tool developed in this research and describes the approach used to develop the tool.

# Chapter 3. Objective and Approach

This chapter defines the objective of developing the network measurement tool and describes the approach taken for the design of the tool.

Advances in technologies have made handheld devices extremely efficient and flexible tools that are widely used in medical, pharmaceutical, business, construction, manufacturing, and marketing industries. Handheld devices are more portable and better suited for many applications than desktop and notebook computers. Most of the network measurement tools run on regular Windows operating systems, such as Windows 2000 and Windows XP, on desktop and notebook computers. Instead of transporting these machines from place to place when taking measurements or collecting data, using a lightweight device makes the task much easier and more convenient.

The tool developed in this research consists of two components, the server and the client. The server runs on a regular Windows operating system, while the client runs on the handheld device, a Pocket PC device. This tool is different from other tools since it provides mobility for users when performing experiments. Users can start the server program and leave it running on the desktop computer, while taking measurements using the handheld device at any place that has a wired or wireless network connection. It is convenient to users, especially network administrators, when measuring and monitoring the network performance. Microsoft Embedded Visual Tools version 3.0 is used to develop the client program for the handheld device in C/C++, while the server program is developed using Microsoft Visual C++. The tool is designed to measure the network throughput by varying the total number of bytes to send, the block size, and the transport protocol. The server can run on any host, but the client has to have the IP address or the host name of the computer on which the server is running to establish a connection. The network tool is also a user-friendly program since it is a windows application that implements a GUI.

Experiments are performed to test and verify the operation of the tool. The experimental setup consists of two end hosts, the iPAQ device that acts as the client and the desktop that acts as the server. The performance for three transport protocols, TCP, TCP with the TCP_NODELAY option, and UDP, are measured with and without a VPN connection. The performance is measured for three locations on the Virginia Tech campus IEEE 802.11b wireless local area network and one location on an off-campus Ethernet network. Measurement data, such as the elapsed time and throughput, are collected and analyzed to ensure that the tool functions as expected.

The primary advantage of using the handheld device is to help users save time by providing mobility, especially when using the device as a data collection tool. The following chapter describes the design of the network tool and the operation of the server and the client programs.

# Chapter 4. Design

This chapter describes the design of the network measurement tool developed in this research to measure TCP/UDP performance. The chapter also explains how the client and server operate and presents the detailed design of the client and server programs.

## 4.1 General Design

The purpose of this network application is to measure real-time end-to-end network performance between a PDA and a regular desktop computer. The tool consists of two programs, a server and a client, that are written in the C++ language. The client program executes on the PDA (a Compaq iPAQ), while the server program runs on a desktop or notebook computer with the Microsoft Windows operating system. Since the Compaq iPAQ uses the Pocket PC operating system, the client program is developed with the Microsoft's Embedded Visual C++ Tools along with the Windows CE (Pocket PC) SDK. Procedures for developing Windows CE applications are described in Section 2.4.

This tool is different from most network measurement tools because it is a Windows application instead of a command-line application. Its GUI is more user-friendly and, more importantly, Pocket PC does not support console application. When the tool is started, the client and server display connection information and data transaction status on the screen to show that the test is running as expected. This provides a visual aid for users monitoring a test's progress. In the client program, users can define a transport protocol, a connecting site (server), the total number of bytes to send, block size, and port number. These parameters allow users to easily create different kinds of network performance tests. Three transport protocol variations are supported: UDP, TCP, and TCP with the TCP_NODELAY option. The TCP_NODELAY option turns off the Nagel algorithm [27]. The Nagel algorithm may combine some small packets into a larger packet to reduce overhead and use the network bandwidth more efficiently. For the server program, only a port number needs to be defined by the user. It waits and accepts any of the protocol connections from the client. Users can start the server program by selecting the "Start" option. Instructions on how to operate the tool are provided in manuals for the client and server in Appendix A.

The next section describes how a client and a server establish a connection and communicate with each other. Sections 4.3 and 4.4 present client and server design, respectively. Refer to Appendix B for further details on the design of the client and server programs.

## 4.2 Communication Between Client and Server

The connection between the client and the server is established, as usual, according to the transport protocol defined by the client. After establishing the connection successfully, the client and server can start to communicate with each other. However, before the actual data transmission, the client has to notify the server of the total number of bytes it is going to transfer and the server has to reply with an "ack" for confirmation, as shown in Figure 8. The "notification" message specifies the total number of bytes to send. The "notification" message is sent through a control socket using TCP for reliability. The client and server also have to use the same port number for address binding. The default port number for the server and client socket is 5555. If 5555 is used as the port number, 5556 is the port number used to send the control information. The control port number is always one greater than the port number used for data transmission.

Upon completion of data transfer, the client asks the server about its performance, as measured at the server, with a "query" message. Figure 9 shows the sequence of communications between the client and server after the notification message. The "query" message and response are also transmitted through the control socket using TCP to ensure reliable transmission. Throughout this thesis, "TCP socket" represents a socket using TCP, "UDP socket" represents a socket using UDP as the transport protocol, and the control information corresponds to either the "notification" or "query" message. Both programs display their own sending and receiving information, such as the connecting host's name, transport protocol used, and data sent/received, on the screen. In addition, the client also displays the performance reported by the server. All transactions are automatically saved to a file. The client stores the server's and its own performance information, but the server only saves its own performance information. Users can provide their own file name or use the default file name, "data.txt," for storage of performance results.

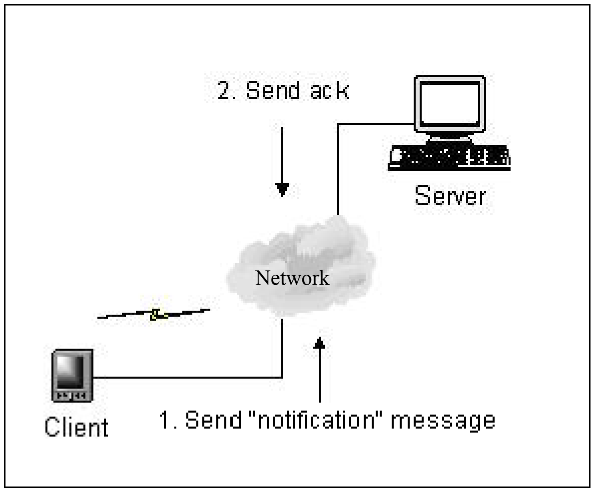



**Figure 8.** Establishing initial communication between client and server.

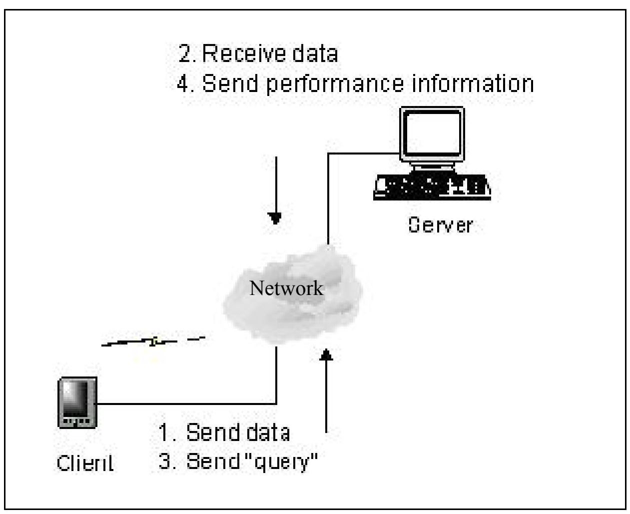



**Figure 9.** Transmission between server and client.

## 4.3 Client Design

### 4.3.1 Features

Users can configure the transmission parameters, including host name or host address, total number of bytes to send, block size, port number, and transport protocol, in the configuration dialog box. Setting these parameters allows users to perform different types of network experiments. Users can choose among UDP, TCP, and the TCP_NODELAY option as the transport protocol. The protocols all use the same port number for the server address. For convenience, the client program saves a copy of the most recent host name or IP address in the "init.txt" file. Thus, users do not have to input this parameter again for the next program execution if the same connecting server host is used. To further analyze the network performance, data is saved automatically to four separate files. One of the files stores all the transaction information about the client and the server and the other three files save the performance information according to the protocol used when performing experiments. The file "tcp.txt" is associated with the performance of the TCP protocol, "udp.txt" is for UDP protocol data, and "tcpnd.txt" is for data associated with the TCP protocol with the TCP_NODELAY option. Data including total number of bytes sent, block size, throughput, number of application-level writes and reads, and elapsed time are stored in columnar style, which can be used to generate tables and graphs with Excel or similar programs, as shown in Appendix C.

### 4.3.2 Data Transactions

The client sends a request to the server to establish a connection. If the connection is successfully established, the client sends a "notification" message with the total number of bytes to send to the server and waits for an acknowledgement from the server through a TCP connection. Upon receiving the acknowledgement, the client starts sending data to the server. The client defines two sockets for connections, a control socket using TCP for transferring control information and a socket using the transport protocol specified by the user for data transfer. To estimate the client throughput, the times when the first bit and last bit of data are sent are recorded. The throughput (in Kbytes/s) is calculated by dividing the total data transferred (in Kbytes) by the elapsed time (in seconds), as shown in Equation (1).

$$Throughput = \frac{Total\ data\ sent\,/\,received\ (Kbytes)}{Elapsed\ time\ (s)} \quad (1)$$

For the server throughput, the elapsed time is the time from when the last bit and first bit of data are received. When the client has finished sending the data, it waits for approximately two seconds and sends a “query” message through the TCP control socket to the server asking for performance information. With a high-speed connection, the server should have received all the packets and can send performance information to the client in a short amount of time. Thus, the waiting time of two seconds is reasonable for the server to finish receiving data and send performance information to the client.

### 4.3.3 Interface

After a connection has been established, the client displays the connecting site, protocol used, port number, total bytes sent, and block size to the screen, as shown in Figure 3. When data is being sent to the server, a dot symbol is displayed on the screen. If UDP is used as the transport protocol, each dot represents one packet. If either TCP or the TCP_NODELAY option is used as the transport protocol, each dot indicates one write to the output buffer. Both the client and the server throughputs are calculated using Equation 1 and are displayed on the screen.

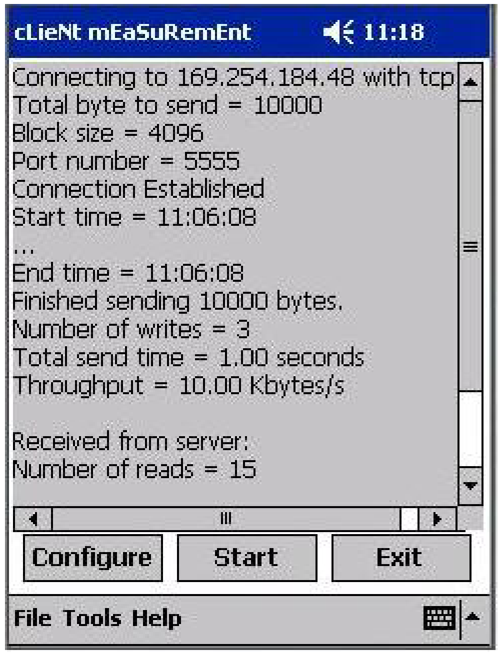



**Figure 10.** Client display screen.

## 4.4 Server Design

### 4.4.1 Features

The main function of the server is to wait for connections from the client and process client requests. It is a multiprotocol server that uses asynchronous input/output to handle communication over either TCP or UDP. The server program automatically creates three asynchronous sockets, TCP and UDP sockets for transmitting data and a TCP control socket for sending "notification" message acknowledgements and "query" responses. A TCP socket is used to accept connections using either TCP or TCP with the TCP_NODELAY option. Since the accepting sockets are defined as asynchronous sockets, there is no time limit on waiting for client connections. The port number defined has to be the same as specified in the client's configuration. All receiving information seen on the display screen is saved to a file named "data.txt" or a user-defined file name. The user can define a file name by selecting the "File" menu and the "Save Log File…" option. The "About" option under the "Help" menu shows the server host name and IP address in a dialog box.

### 4.4.2 Data Transactions

After setting up the sockets, the server waits for client requests. It first receives a "notification" message from the client and then responds with an "ack" through the TCP control socket. Within the program, since an asynchronous socket is used, a read notification message is sent to the corresponding socket when the data is ready to be read from the buffer. The server also keeps track of how much data it has received and the elapsed time. The elapsed time is the time from when the first bit of data is received until the last bit of data is received. When the server receives the "query" message, it stops reading data from the buffer and sends its performance information, including the throughput, the total number of bytes it has received, the number of reads, and the elapsed time, to the client. The difference between the total number of bytes actually received and the number of bytes to be sent specified in the "notification" message is used to estimate data loss.

### 4.4.3 Interface

At startup, the port number used in the experiment is shown in the server window after the port number is configured and the "start" button is clicked. A "Listening for incoming connections" line indicates that the server is waiting for a client request. Once the server has accepted a connection from the client, it shows the transport protocol being used and the client's IP address. When the server starts receiving data, dots are displayed to the screen indicating that it is receiving packets. When the transmission is finished, the total number of bytes received and the throughput are also shown in the display window. The message "Finished sending performance information to the client" indicates that it has replied to the client "query" request. Refer to Appendix B for design details.

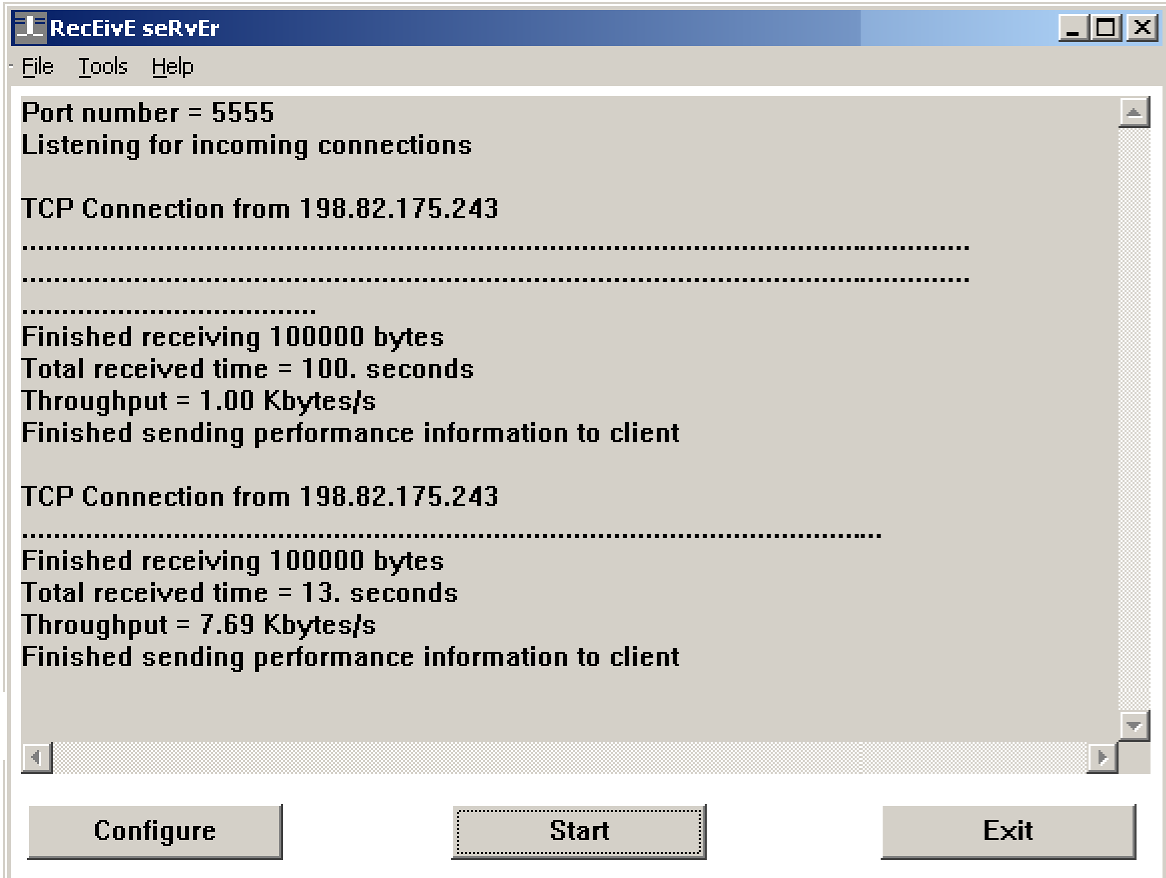



**Figure 11.** Server interface.

## 4.5 Summary

This chapter presented the overall design and operation of the network measurement tool. In summary, a client program and a server program are used to measure the network performance. The use of a handheld device for completing the measurement tasks is much easier and more convenient than the traditional use of a desktop computer or even a laptop. The next chapter defines the metrics and factors that may affect the network performance and describes the experiments conducted using the tool described in Chapter 4.

# Chapter 5. Description of Experiments

This chapter describes the measurement environment and defines the metrics observed in the experiments.

The experimental setup consists of a desktop running Windows 2000 and a Compaq iPAQ model 3670 device running Pocket PC 2002. The desktop acts as the server, while the iPAQ acts as the client. The server runs on a machine in Torgersen Hall and connects to the network over the campus Ethernet using a 10-Mbps connection. The iPAQ uses a Compaq WL110 Wireless LAN PC Card along with the expansion pack as a PC Card adapter to establish a network connection. The client obtains an IP address assigned by a Dynamic Host Configuration Protocol (DHCP) server. Figure 12 shows the experimental testbed configuration for the measurement. The client sends data to the server and measures the network performance from three campus buildings, Torgersen Hall, Randolph Hall, and Pamplin Hall, over the campus wireless network.

A second scenario is to have the client reside on an off-campus network, while the server remains attached to the on-campus network, as in the first scenario. The client's IP address is assigned by the off-campus Internet Service Provider (ISP). The network performance is measured over a wired Ethernet connection with a Socket LP-E Ethernet PC Card for this second scenario.

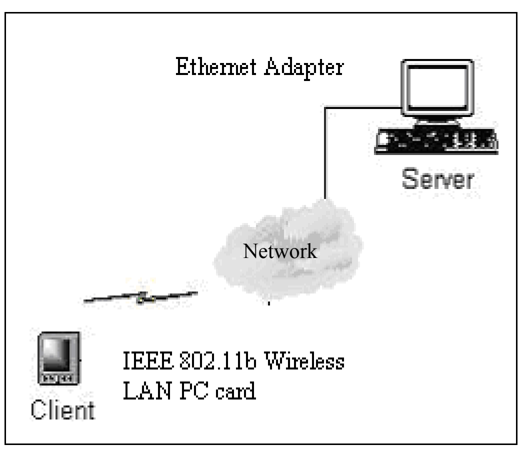



**Figure 12**. Experimental testbed configuration for on-campus measurements.

Besides the regular wired and wireless network connection, the network performance using a VPN connection is also measured for both scenarios. The client sets up the VPN connection by connecting to a VPN server in Torgersen Hall before starting the measurement. In both scenarios, both programs are configured to use port number 5555. A summary of the experiments is shown in Table 1. The throughput is measured with three different transport protocol variations, TCP, TCP with the NO_DELAY option, and UDP, that can be defined in the client configuration dialog box. 1 Kbytes to 100 Kbytes of data are sent using block sizes of 50 to 100 Kbytes. Note that the server always runs in Torgersen Hall on the on-campus network. Also, note that Kbytes equals 1000 bytes, not 1024 bytes.

**Table 1.** Summary of Experimental Parameters

| Client location | Security | Transport protocol | Number of bytes to send | Block size |
|---|---|---|---|---|
| **Scenario 1:**<br>• Torgersen Hall<br>• Randolph Hall<br>• Pamplin Hall<br><br>**Scenario 2:**<br>• Off-campus | • No VPN<br><br>• VPN | • TCP<br><br>• TCP_NODELAY<br><br>• UDP | 1000 to 100 K bytes<br><br>(See Table 2) | 50 to 100K bytes<br><br>(See Table 2) |

The total number of bytes to send and the block size used in the experiments are defined in Table 2. These values can be configured in the client configuration dialog box. A total of 17 tests are defined for each experiment. Since the maximum payload length for an IP packet is 65,535 bytes, a 50-Kbyte block size instead of a 100-Kbytes block size is used for the UDP measurements. Instructions on how to operate the tool are provided in Appendix A. The Ping Plotter[4] program is used to determine the number of hops from the client to the server. The client and the server are four hops away from each other for the on-campus scenario and nine hops away for the off-campus scenario.

[4] Ping Plotter can be found at http://www.pingplotter.com. It is a network troubleshooting and diagnostic tool that combines trace route, ping, and whois.

**Table 2.** Test Parameters Defined in the Client

| Total number of bytes to send (bytes) | Block size (bytes) |
|---|---|
| 1 k | 50 |
| 1 k | 100 |
| 1 k | 500 |
| 1 k | 1 k |
| 10 k | 50 |
| 10 k | 100 |
| 10 k | 500 |
| 10 k | 1 k |
| 10 k | 5 k |
| 10 k | 10 k |
| 100 k | 50 |
| 100 k | 100 |
| 100 k | 500 |
| 100 k | 1 k |
| 100 k | 5 k |
| 100 k | 10 k |
| 100 k | 100 k / 50 k |

The tool reports the performance of the path between the source (client) and the destination (server) by showing the amount of data transferred, the transfer time, and the approximate throughput measured at the client and the server, as defined in Equation 1. The next chapter presents the results and analysis of the data collected.

# Chapter 6. Results

This chapter reports and analyzes the results observed from the experiments defined in the previous chapter. Detailed results are tabulated in Appendix C. For Scenario 1 (with the client connecting to the on-campus network), the throughput values are similar for all locations. Therefore, only the performance measured in Torgersen Hall is discussed in the next section as the on-campus scenario. The results of Scenario 2 (with the client connecting to an off-campus network) are analyzed in Section 6.2. The performance when sending 100,000 bytes of data is discussed for both scenarios. Section 6.3 compares the throughput observed for Scenario 1 and Scenario 2.

## 6.1 Scenario 1: On-Campus Measurements

The server performance measured in Torgersen Hall is displayed in Figure 13 to Figure 18 and Table 3 to Table 8 according to each protocol and connection type. Figure 13, Figure 14, and Figure 15 show the throughput for a regular connection, while Figure 16, Figure 17, and Figure 18 show the throughput measured for a VPN connection. As seen in the figures, throughput values are similar for all protocols and reach a maximum throughput of 100 Kbytes/s when the block size is 5,000 bytes or more. Recall that the throughput is the total number of bytes received divided by the total time to receive the bytes. Thus, for 100 Kbytes of data and a receive time of 1 second, the throughput is 100 Kbytes/s. In Figure 17, the throughput for the TCP with TCP_NODELAY option using a VPN connection is different from the others because the server receives all 100 Kbytes of data in 2 seconds instead of 1 second, as shown in Table 5, and produces a throughput of 50 Kbytes/s. (The longer receive time might be caused by delays at the VPN server or a burst of traffic when performing the experiment.)

At the transport layer, TCP directs the data to a send buffer and sends the data at its own convenience. The Maximum Segment Size (MSS) is the maximum amount of application-level data that can be placed in a single TCP segment. The MSS is equal to the Maximum Transfer Unit (MTU) minus the size of the IP header and the TCP header. TCP tries to determine the MTU size when segmenting data to avoid IP fragmentation. Data segments are passed down to the network layer, where IP checks the length of the TCP segment and may fragment the data, if needed. With Ethernet having an MTU of 1,500 bytes, all packets sent should be 1,500 bytes or less in length. At the destination, the fragmented packets need to be reassembled by IP before they are delivered to the transport layer. Both UDP and TCP expect to receive complete and unfragmented datagrams or segments, respectively, from the network layer.

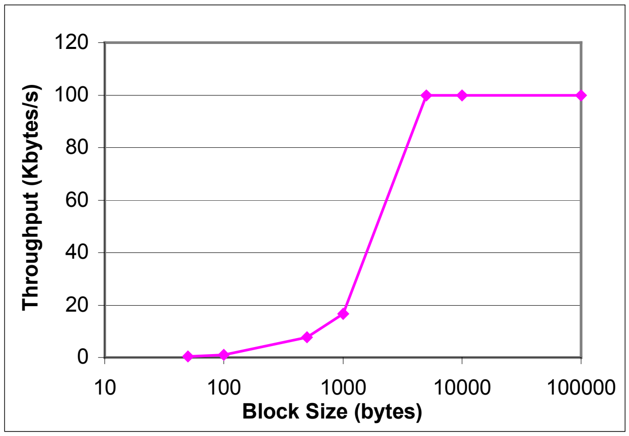



**Figure 13.** TCP throughput measured in Torgersen Hall.

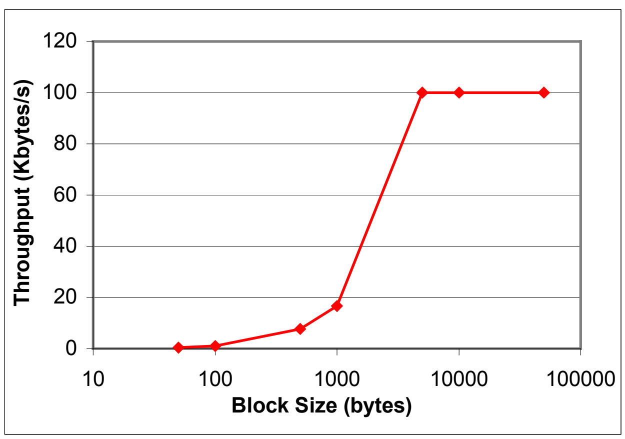



**Figure 14.** TCP throughput with TCP_NODELAY option measured in Torgersen Hall.

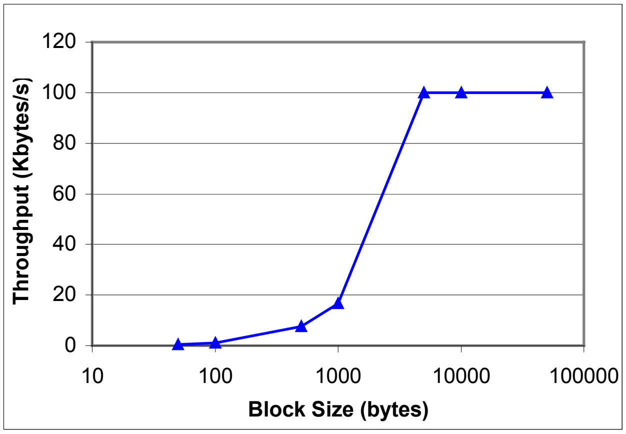



**Figure 15.** UDP throughput measured in Torgersen Hall.

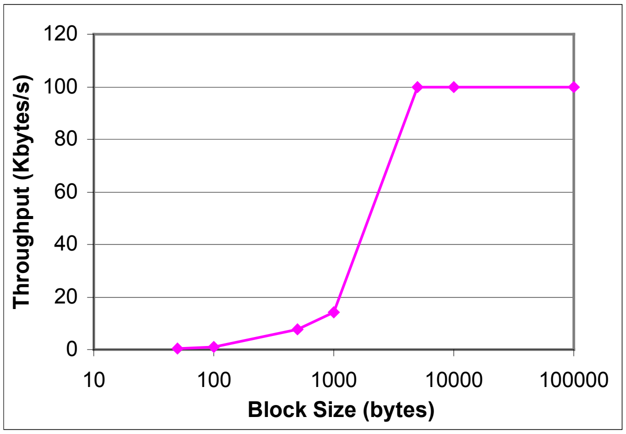



**Figure 16.** TCP throughput with VPN measured in Torgersen Hall.

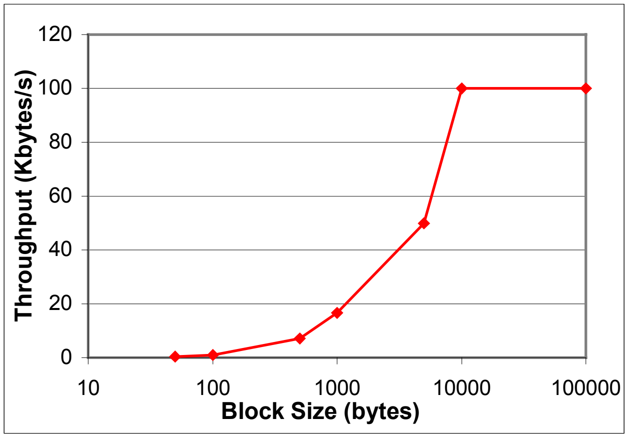



**Figure 17.** TCP throughput with TCP_NODELAY and VPN measured in Torgersen Hall.

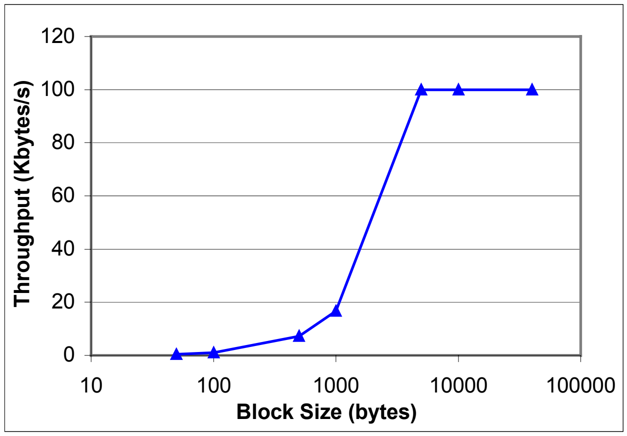



**Figure 18.** UDP throughput with VPN measured in Torgersen Hall.

Table 3, Table 4, and Table 5 show the results observed using the regular connection, while Table 6, Table 7, and Table 8 show the results measured using the VPN connection. The number of writes shown in the tables refers to application-level writes, which is the total number of bytes sent divided by the block size. With 100 Kbytes of data and block size of 50 bytes, the number of writes is 2,000. The number of reads is different than the number of writes when using TCP because when the TCP buffer is filled with data, the receiving program is notified that there is data in the TCP buffer. As shown in Table 3, the client has to write 2,000 times to the buffer when sending 100 Kbytes of data with 50 bytes per write, but the server only reads 971 times. Some of the data packets might have arrived at the server (receiver) at the same time and were put into the TCP receive buffer to be read. Also, a TCP application can read data at its own convenience. So, the number of reads is smaller than the number of writes in this situation. However, for the 100-Kbyte block size, the client only writes once to the buffer and the server reads 72 times to retrieve all 100 Kbytes of data. The block of data is segmented by TCP for sending over the network. A large block of data written by the application must be sent as multiple segments. When sending 100 Kbytes of data, the client only writes once at the application level, but this packet is segmented by TCP and, potentially, may be fragmented by IP. Thus, the server tends to read more often at the application level especially for large block sizes.

The TCP_NODELAY option implies that the TCP protocol sends data immediately without waiting for more data. So, TCP at the client sends more frequently and the server reads more frequently. The number of reads in Table 3 and Table 4 shows that slightly fewer reads occur with the regular TCP connection than when the TCP_NODELAY option is used.

Unlike TCP, UDP is an unreliable protocol since lost packets will not be retransmitted. Since UDP is a datagram service that does not segment data before it is send to the IP layer, the maximum data length allowed is 64 Kbytes due to IP's maximum payload. Data segments larger than the MTU are fragmented into MTU-size packets by IP. When the fragmented packets reach the destination, IP at the receiver reassembles the fragmented packets to their original form and sends them to the transport layer. Thus, the number of writes is the same as the number of reads if no data is lost. If data is lost, the number of lost bytes is always a multiple of the block size.

**Table 3.** TCP Performance Measured in Torgersen Hall

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 214 | 0.47 | 2000 | 214 | 0.47 | 971 | 100000 |
| 100000 | 100 | 100 | 1 | 1000 | 100 | 1 | 456 | 100000 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.69 | 128 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 77 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 69 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 72 | 100000 |

**Table 4.** TCP Performance with TCP_NODELAY Option Measured in Torgersen Hall

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 214 | 0.47 | 2000 | 214 | 0.47 | 978 | 100000 |
| 100000 | 100 | 97 | 1.03 | 1000 | 97 | 1.03 | 457 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 13 | 7.69 | 129 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 78 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 71 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 72 | 100000 |

**Table 5.** UDP Performance Measured in Torgersen Hall

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 216 | 0.46 | 2000 | 216 | 0.46 | 1998 | 99900 |
| 100000 | 100 | 100 | 1 | 1000 | 100 | 1 | 1000 | 100000 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.69 | 200 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 20 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 10 | 100000 |
| 100000 | 50000 | 1 | 100 | 2 | 1 | 100 | 2 | 100000 |

**Table 6.** TCP Performance with VPN Measured in Torgersen Hall

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 220 | 0.45 | 2000 | 221 | 0.45 | 428 | 100000 |
| 100000 | 100 | 101 | 0.99 | 1000 | 104 | 0.96 | 192 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 14 | 7.14 | 145 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 7 | 14.29 | 99 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 68 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 73 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 70 | 100000 |

**Table 7.** TCP Performance with TCP_NODELAY Option and VPN Measured in Torgersen Hall

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 220 | 0.45 | 2000 | 220 | 0.45 | 398 | 100000 |
| 100000 | 100 | 102 | 0.98 | 1000 | 103 | 0.97 | 226 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 14 | 7.14 | 144 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 2 | 50 | 74 | 100000 |
| 100000 | 10000 | 2 | 50 | 10 | 1 | 100 | 68 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 72 | 100000 |

**Table 8.** UDP Performance with VPN Measured in Torgersen Hall

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 222 | 0.45 | 2000 | 222 | 0.45 | 2000 | 100000 |
| 100000 | 100 | 102 | 0.98 | 1000 | 102 | 0.98 | 1000 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 14 | 7.14 | 200 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 20 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 10 | 100000 |
| 100000 | 40000 | 1 | 100 | 3 | 1 | 100 | 3 | 100000 |

The network throughput measured using the VPN connection is close to the throughput measured without the VPN connection. The only difference is the number of reads when using TCP as the transport protocol. TCP attempts to send TCP segments that fit within the link layer's frame. The Point-to-Point Tunneling Protocol (PPTP) is used in conjunction with the Generic Route Encapsulation (GRE) to create a VPN between the client and the server. Data is encapsulated, encrypted, and compressed in a GRE packet and tunneled through the VPN server before reaching the destination. At the VPN server (the end-point of the tunnel), the GRE header and the PPTP header are striped off and the encapsulated packet (decrypted data) is sent to the server in an IP packet. Figure 19 shows that the server receives data from 128.173.52.26 (VPN server), but the actual sender is the client. Since the VPN server sends data directly to the server, bursts of data arrive at the server causing a smaller number of reads.

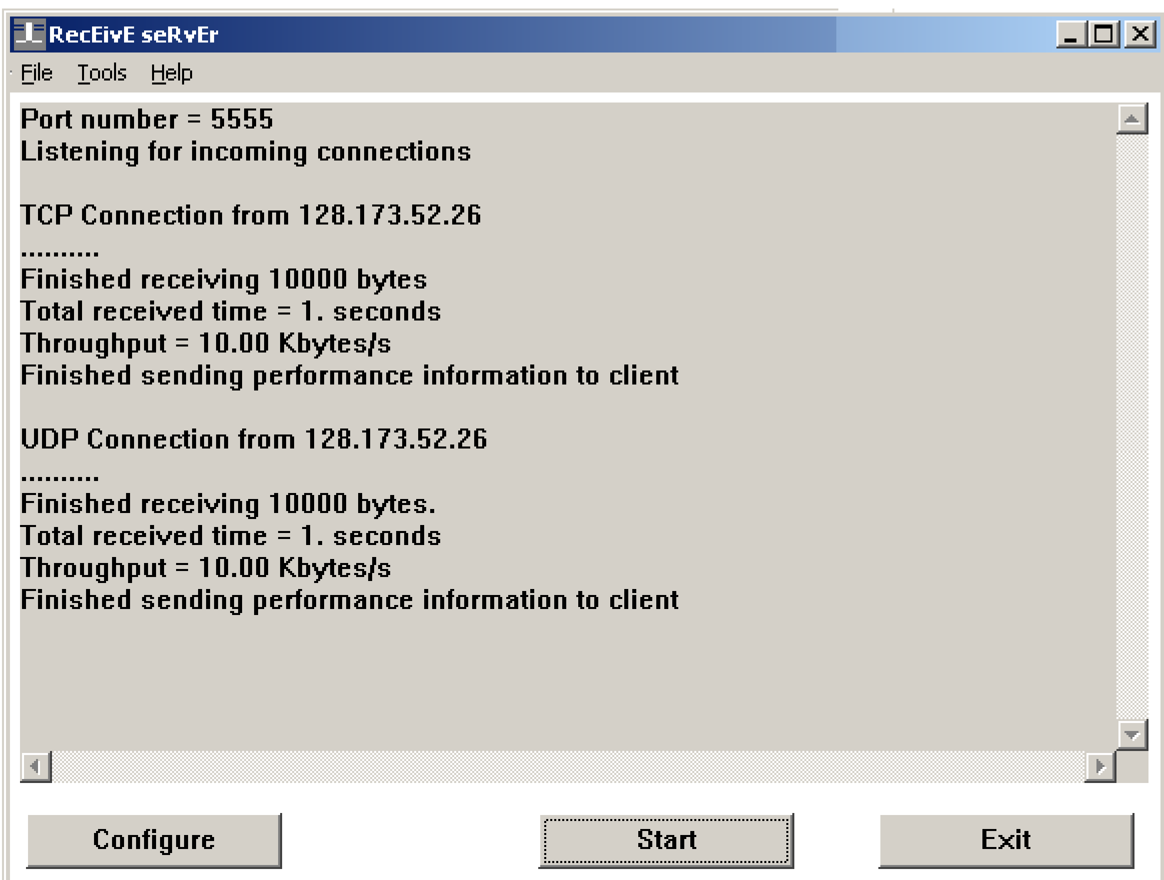



**Figure 19.** Data is sent by the client, but the server only sees the VPN server as the sender.

The MTU size is a known problem with GRE when the "do not fragment" (DF) bit is set and the tunneling system fragments the packet [28]. Some applications cannot process the packets once they have been fragmented and, thus, discard those packets. When measuring UDP performance using a VPN with 50 Kbytes per block, packets appear to be sent successfully, but the server does not receive any of the data. This might be caused by faulty fragmentation or compression in the iPAQ when handling UDP packets carrying more than 45 Kbytes of data.

For this experiment, the client showed that it sent all data, but it did not receive a response from the server. Packets captured with Microsoft's Network Monitor program on the VPN server verified that the packets did arrive at the VPN server. However, there was a packet that was fragmented into two packets before reaching the VPN server. The iPAQ might have generated a 1500-byte packet and, then, PPTP tried to add the 24-bit GRE header to the packet, making this packet longer than the link layer MTU size. Thus, this packet is fragmented by IP to fit within the MTU limit. These two packets consist of a 1500-byte PPTP packet with the "more fragments" bit set followed by a 32-byte IP packet without GRE encapsulation. The VPN server might not be able to process these two packets and, thus, cannot reassemble the 50 Kbytes of data. Even through the previous data are received correctly, all packets that carried the 50 Kbytes of UDP data are dropped by the VPN server. For every 50 Kbytes of data sent by the client, there was one faulty packet. Therefore, all the received packets were discarded at the VPN server and the server never received any data from the client. It is assumed that IP at the iPAQ did not fragment packets correctly when handling data segments longer than 45 Kbytes. This behavior does not affect TCP because TCP itself segments the application data to avoid further fragmentation at the network layer and it resends lost and dropped packets. However, UDP sends the application data directly to the network layer for fragmentation.

## 6.2 Scenario 2: Off-Campus Measurements

The results observed for the off-campus scenario are shown in Figure 20 and Figure 21. In Figure 20, the observed values for throughput for TCP and for TCP with the TCP_NODELAY option are similar and both converge to 100 Kbytes/s for block sizes larger than 10 Kbytes.

However, UDP has lower throughput and throughput starts to decline for block size larger than 10 Kbytes. TCP has better throughput since it is a reliable protocol that retransmit packets when data is lost. Due to relatively heavy traffic load in the off-campus network, UDP data might be lost or experience excessive delay. Table 11 shows that the total number of bytes received by the server is less than the total number of bytes sent by the client for each experiment. As the block size is increasing, the total number of bytes received by the server is decreasing. As soon as the server receives the query packet, the server stops receiving data and sends its performance results to the client. Data not received by that time are considered lost or delayed. Thus, the lost data causes lower throughput.

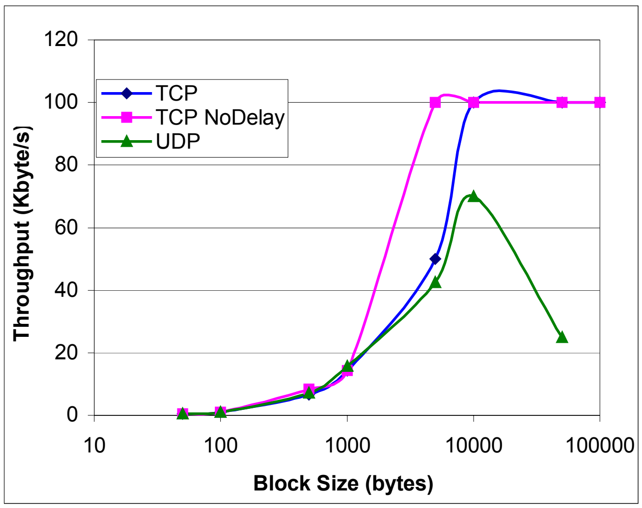



**Figure 20.** Throughput for the off-campus scenario.

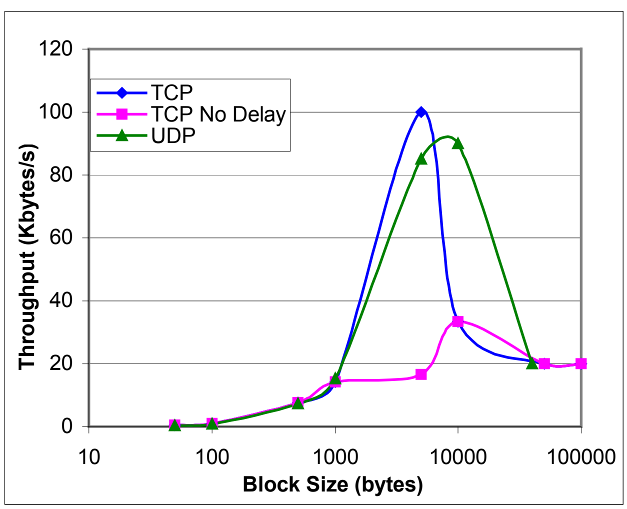



**Figure 21.** Throughput with VPN connection for the off-campus scenario.

In Figure 21, the throughput measured with the VPN connection starts declining at a block size of 10 Kbytes and reaches 20 Kbytes/s for all three protocols at a block size of 50 Kbytes. The sudden reduction in throughput might be caused by segmentation by TCP or data fragmentation by IP since larger blocks must be segmented and/or fragmented. When handling block sizes larger than 50 Kbytes and using TCP as the transport protocol, either the client might have transmitted fragmented IP packets that cannot be reassembled by the VPN server or the client did not receive the acknowledgments in time and, thus causing TCP to resend the packets. The client sent 100 Kbytes of data in 8 seconds over the VPN connection and in 1 second over the connection without VPN. For a block size of 100 Kbytes, the send time and the receive time shown in Table 12 and Table 13 are higher than those in Table 9 and Table 10. The client send time affects the server receive time. Therefore, the throughput is lower.

The throughput for the TCP with TCP_NODELAY option is lower than those of the TCP because Nagel algorithm was disabled and more packets need to be sent and acknowledged. With heavy traffic, the server takes longer time to receive all data and caused lower throughput. Furthermore, Internet traffic that varies widely is also a reason why the TCP with TCP_NODELAY option enabled has lower throughput value than with regular TCP. For UDP, the throughput is similar with VPN and without VPN, and the throughput begins to drop for block sizes larger than 10 Kbytes. Table 11 and Table 14 show that UDP data might be lost or delayed when transferring through the off-campus network. The reason that throughput for a block size of 50 Kbytes cannot be measured for UDP was reported in the previous section.

**Table 9.** TCP Performance for Off-Campus Scenario

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 216 | 0.46 | 2000 | 216 | 0.46 | 859 | 100000 |
| 100000 | 100 | 100 | 1 | 1000 | 100 | 1 | 412 | 100000 |
| 100000 | 500 | 15 | 6.67 | 200 | 15 | 6.67 | 110 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 7 | 14.29 | 73 | 100000 |
| 100000 | 5000 | 2 | 50 | 20 | 2 | 50 | 56 | 100000 |
| 100000 | 10000 | 2 | 50 | 10 | 1 | 100 | 59 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 6 | 100000 |

**Table 10.** TCP Performance with TCP_NODELAY Option for Off-Campus Scenario

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 216 | 0.46 | 2000 | 216 | 0.46 | 844 | 100000 |
| 100000 | 100 | 99 | 1.01 | 1000 | 99 | 1.01 | 397 | 100000 |
| 100000 | 500 | 12 | 8.33 | 200 | 12 | 8.33 | 109 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 7 | 14.29 | 82 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 63 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 61 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 65 | 100000 |

**Table 11.** UDP Performance for Off-Campus Scenario

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 219 | 0.46 | 2000 | 222 | 0.43 | 1902 | 95100 |
| 100000 | 100 | 100 | 1 | 1000 | 101 | 0.94 | 951 | 95100 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.15 | 186 | 93000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 15.67 | 94 | 94000 |
| 100000 | 5000 | 2 | 50 | 20 | 2 | 42.5 | 17 | 85000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 70 | 7 | 70000 |
| 100000 | 50000 | 1 | 100 | 2 | 2 | 25 | 1 | 50000 |

**Table 12.** TCP Performance with VPN for Off-Campus Scenario

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 218 | 0.46 | 2000 | 219 | 0.46 | 372 | 100000 |
| 100000 | 100 | 99 | 1.01 | 1000 | 99 | 1.01 | 197 | 100000 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.69 | 126 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 7 | 14.29 | 88 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 73 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 3 | 33.33 | 57 | 100000 |
| 100000 | 100000 | 8 | 12.5 | 1 | 5 | 20 | 66 | 100000 |

**Table 13.** TCP Performance with TCP_NODELAY Option and VPN for Off-Campus Scenario

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 217 | 0.46 | 2000 | 217 | 0.46 | 374 | 100000 |
| 100000 | 100 | 100 | 1 | 1000 | 100 | 1 | 224 | 100000 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.69 | 116 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 7 | 14.29 | 91 | 100000 |
| 100000 | 5000 | 6 | 16.67 | 20 | 6 | 16.67 | 66 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 3 | 33.33 | 61 | 100000 |
| 100000 | 100000 | 4 | 25 | 1 | 5 | 20 | 64 | 100000 |

**Table 14.** UDP Performance Measured over VPN for Off-Campus Scenario

| Total bytes sent (bytes) | Block size | Client send time | Client throughput (Kbytes/s) | Number of writes | Server receive time | Server throughput (Kbytes/s) | Number of reads | Total received (Bytes) |
|---|---|---|---|---|---|---|---|---|
| 100000 | 50 | 222 | 0.45 | 2000 | 223 | 0.41 | 1822 | 91100 |
| 100000 | 100 | 102 | 0.98 | 1000 | 102 | 0.9 | 917 | 91700 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.31 | 190 | 95000 |
| 100000 | 1000 | 5 | 20 | 100 | 6 | 15.17 | 91 | 91000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 85 | 17 | 85000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 90 | 9 | 90000 |
| 100000 | 40000 | 1 | 100 | 3 | 1 | 20 | 5 | 20000 |

## 6.3 Comparison of On-Campus (Scenario 1) and Off-Campus (Scenario 2) Results

Throughput when sending 100 Kbytes of data for Scenario 1 and Scenario 2 is compared and analyzed in this section. Throughput measured in Torgersen Hall is used for Scenario 1. The TCP throughput measurements over the on-campus and off-campus network have similar values, as shown in Figure 22 and Figure 23. However, UDP performance over the off-campus network has lower throughput and throughput starts to decline at a block size of 10 Kbytes. Data fragmentation and heavy traffic are two important reasons for low throughput. Figure 25, Figure 26, and Figure 27 show that the throughput measured using a VPN connection on the off-campus network is lower than that measured on the on-campus network. At a blocks size of 10 Kbytes, the off-campus throughput for all three protocols starts to decline, while the on-campus throughput reaches the maximum throughput of 100 Kbytes/s. When using TCP and PPTP, every packet sent needs to be acknowledged by the VPN server. With the heavier traffic in the off-campus network, the acknowledgment packets take longer to reach the client. Furthermore, lost packets are retransmitted and all packets have to be acknowledged before TCP can declare that it has successfully sent all data. As a consequence, the receive time is higher and the throughput is lower. As for UDP, no acknowledgment is needed and, thus, it has higher throughput when sending large block of application data (10 Kbytes or more). However, UDP data is lost on the off-campus network, which reduces the throughput when compared to the on-campus network.

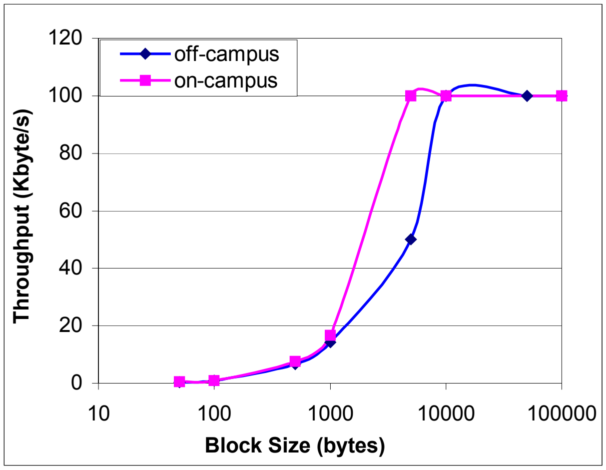



**Figure 22.** TCP throughput (Scenario 1 vs. Scenario 2).

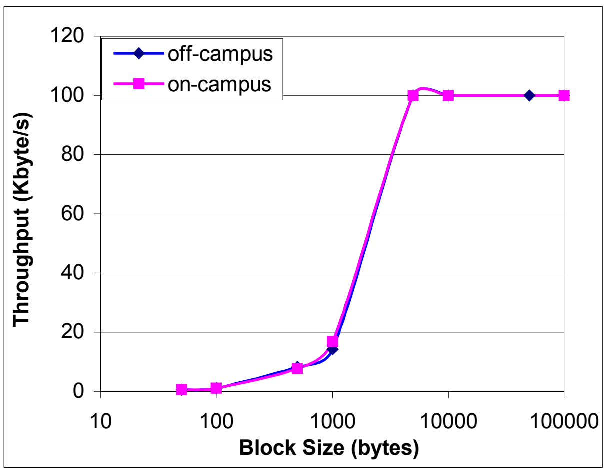



**Figure 23.** TCP throughput with TCP_NODELAY option (Scenario 1 vs. Scenario 2).

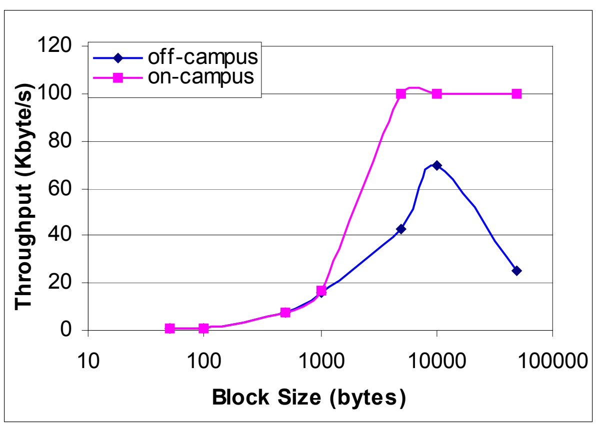



**Figure 24.** UDP throughput (Scenario 1 vs. Scenario 2).

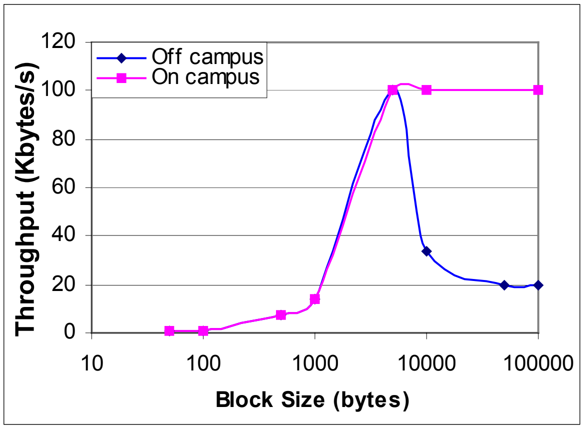



**Figure 25.** TCP throughput with VPN (Scenario 1 vs. Scenario 2).

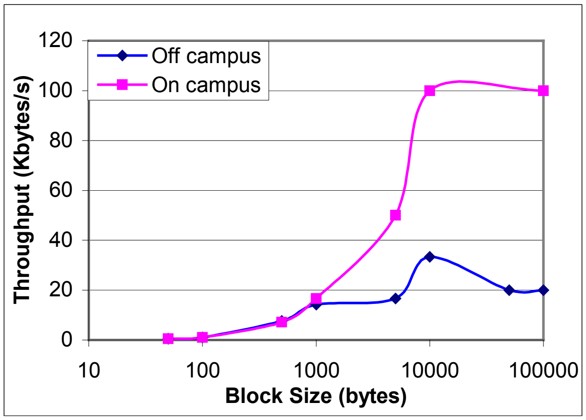



**Figure 26.** TCP throughput with TCP_NODELAY option and VPN (Scenario 1 vs. Scenario 2).

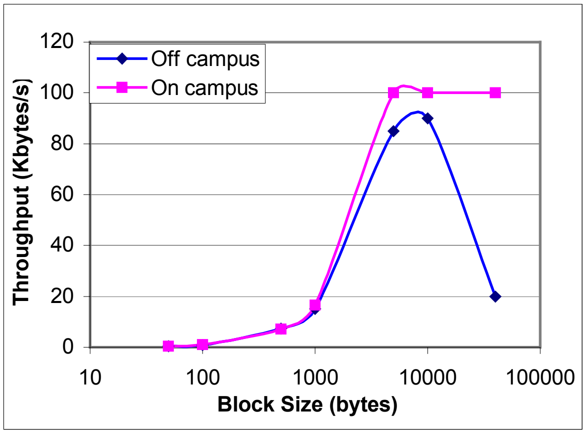



**Figure 27.** UDP throughput with VPN (Scenario 1 vs. Scenario 2).

## 6.4 Summary

This chapter reported and analyzed the network performance measured over the on-campus network and an off-campus network. The throughput results for the on-campus scenario with and without a VPN connection for all three protocols do not vary significantly. In general, the throughput results for TCP and TCP with the TCP_NODELAY option are similar in all experiments, except for the off-campus scenario with a VPN connection. UDP has higher throughput when observed over the on-campus network than over the off-campus network. Low throughput in the off-campus network is likely caused by heavy traffic in the off-campus network.

The network tool functioned as expected. The results generated by the tool seem reasonable, although exact results have not been verified against those produced by a comparable tool. However, there appears to be the possibility of incorrect fragmentation by Pocket PC 2002 on the iPAQ for large block sizes when a VPN is used.

# Chapter 7. Summary and Conclusions

A tool has been designed, implemented, and tested to measure network throughput by sending test data from a client to a server. The client program is a Microsoft Pocket PC application written in C++ using the Microsoft's Embedded Visual Tools and runs on a Compaq iPAQ. The server program is a Microsoft Windows application implemented using C++ in Microsoft Visual Studio and runs on a standard desktop or notebook computer. Users can define a host name, a port number, the transport protocol, the total number of bytes to send, and the block size in the client configuration dialog box. Users can choose among TCP, TCP with the TCP_NODELAY option, and UDP as the transport protocol. Only the port number parameter needs to be defined at the server. Both the client and the server must have the same port number to communicate with each other.

Experiments were performed to verify the operation of the tool. Throughput was measured over the on-campus network and an off-campus network with and without a VPN connection. Results observed were reported and analyzed. The on-campus network exhibited higher throughput than the off-campus network because the path between the client and the server has less traffic and greater capacity. In general, the throughput measured over the connections with VPN and without VPN remains the same as long as the traffic load is light. However, heavy traffic degrades the performance of a VPN connection more than for a regular connection. After analyzing the results, it was concluded that the network measurement tool operates correctly and appears to give reasonable results. Future work could include a more systematic verification of the results by comparing them to those generated by Iperf or other similar tool.

The graphical user interface makes this tool user-friendly and the use of a Pocket PC platform provides mobility. Users can save time and energy when measuring network performance. As technology advances, handheld devices will become more powerful with faster processors and more memory. Future work might involve adding more features to improve the measurement tool. Parameters such as the MTU size can be added to the client configuration to observe its effect on network performance. Also, the data files that are generated by the client can be used to automatically generate graphs and tables to display on the screen.

## References

[1] Cresotech, Inc., http://www.cresotech.com (current Jan. 2003).

[2] Network Associates, Inc., http://www.sniffer.com/products/sniffer-wireless-pda/default.asp (current Jan. 2003).

[3] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," RFC 1157, Internet Engineering Task Force (IETF), May 1990.

[4] T. Oetiker and D. Rand, "Multirouter Traffic Grapher (MRTG)," http://people.ee.ethz.ch/~oetiker/webtools/mrtg (current Jan. 2003).

[5] M. Muuss, "The Story of PING Program," http://ftp.arl.army.mil/~mike/ping.html (current Jan. 2003).

[6] T. Mendolia, "ICMP Datagrams In Traceroute Algorithms," *The Journal of Computing in Small Colleges*, vol. 16, no. 3, March 2001, pp. 215-217.

[7] M. Mathis and J. Mahdavi, "Diagnosing Internet Congestion with a Transport Layer Performance Tool," *Proceedings of INET '96*, Montreal, Quebec, June 1996, http://www.isoc.org/inet96/proceedings/d3/d3_2.htm (current Jan. 2003).

[8] J. Matthews and J. Hollenbeck, "Using TTCP or PCATTCP," November 13, 2001, http://www.clarkson.edu/projects/itl/HOWTOS/PCATTCP-jnm-20011113.htm (current Jan. 2003).

[9] K. Lai and M. Baker, "Nettimer: A Tool for Measuring Bottleneck Link Bandwidth," *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, April 4, 2001, pp. 1-4.

[10] J. Postel, "Internet Control Message Protocol," RFC 792, *IETF*, September 1981, http://www.ietf.org/rfc/rfc0792.txt?number=792 (current Sept. 2002).

[11] S. Savage, "Sting: a TCP-based Network Measurement Tool," Department of Computer Science and Engineering, University of Washington. http://www.cs.washington.edu/homes/savage/papers/Usits99.pdf (current Aug. 2002).

[12] M. Luckie, A. McGregor, and H. Braun, "Towards Improving Packet Probing Techniques," *ACM SIGCOMM*, San Francisco, CA, November 2001, pp 145-147.

[13] P. Dykstra, "ttcp/nttcp/nuttcp/iperf versions," March, 2001, http://sd.wareonearth.com/~phil/net/ttcp/ (current Jan. 2003).

[14] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf Version 1.6.5," *NLANR*, http://dast.nlanr.net/Projects/Iperf/ (current Jan. 2003).

[15] M. Coates, R. Nowak, and Y. Tsang, "Nonparametric Estimation of Internal Delay Densities from Unicast End-to-end Measurement," Technical Report TREE-010, Department of Electrical and Computer Engineering, Rice University, Oct, 2001, http://spin.rice.edu/PDF/tree0106.pdf (current Jan. 2003).

[16] Microsoft Corporation, "Common Executable Format for eMbedded Visual C++ in Microsoft Windows CE 3.0," June 2000, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnce30/html/cef2.asp (current Feb. 2003).

[17] Microsoft Corporation, "Windows CE 3.0 Add-On Pack Press Pass," September 25, 2000, http://www.microsoft.com/presspass/press/2000/Sept00/AddOnPackPR.asp (current Feb. 2003).

[18] D. Boling, *Programming Microsoft Windows CE,* Microsoft Press, Redmond, WA, 2001

[19] Microsoft Corporation, "Embedded Visual Tools," http://msdn.microsoft.com/vstudio/device/embedded/default.asp (current Jan. 2003).

[20] Syware Intuitive Database Technology, http://www.syware.com/prodlib/win_ce/vce/vce.htm (current Jan. 2003).

[21] NS Basic Corporation webpage, http://nsbasic.com/ce/info/nsbce/Overview.html (current Jan. 2003).

[22] Kevin Cao, "Programming language and applications for Windows CE PocketC" *Pocket PC Magazine*, November 13, 2001, http://www.pocketpcmag.com/may99/PocketCArticle.htm (current Oct. 2002).

[23] Pocket PC Developer Network, http://www.pocketpcdn.com/tools/index.html (current Sept. 2002).

[24] ANS Forth for Win32, http://sourceforge.net/projects/spf/ (current Sept. 2002).

[25] Perl 5.6 for Windows CE, http://www.rainer-keuchel.de/wince/perlce.html (current Oct. 2002).

[26] Wabasoft Incorporation, http://www.wabasoft.com (current Sept. 2002).

[27] B. Quinn and D. Shute, *Windows Sockets Network Programming*, Addison Wesley, Reading, MA, 1998, p. 306.

[28] J. Harris, "What is GRE?" May 28, 2001, http://www.sans.org/rr/securitybasics/GRE.php (current April 15, 2003).

# Appendix A. Server and Client Users' Manuals

This appendix provides the server and the client users' manuals that contain instructions to set up the network measurement tool.

## A.1 Server Users' Manual

This section describes how to use the server application that was developed in this research. The server interface is displayed in Figure 28. The "Configure" button brings up the "Configuration" dialog box as shown in Figure 29. The "Port Number" parameter is the transport layer port used for establishing connections between the client and the server. Only the "Port Number" parameter needs to be configured in the server since the server only receives data and accepts all types of protocol connections. However, both the client and the server need to have the same port number. The default port number is 5555 for data transfer, and the control information is sent through a port that is one number higher than the data transfer port number. For example, if 5000 is defined as the port number, 5000 will be used to receive data and 5001 will be used to transfer control information. Click "OK" when finished or "Cancel" to quit the configuration process. The new defined port number will be used if "OK" is clicked. To start listening for incoming connections from a client, click the "Start" button in the main window. The "Exit" button is used to quit the program.

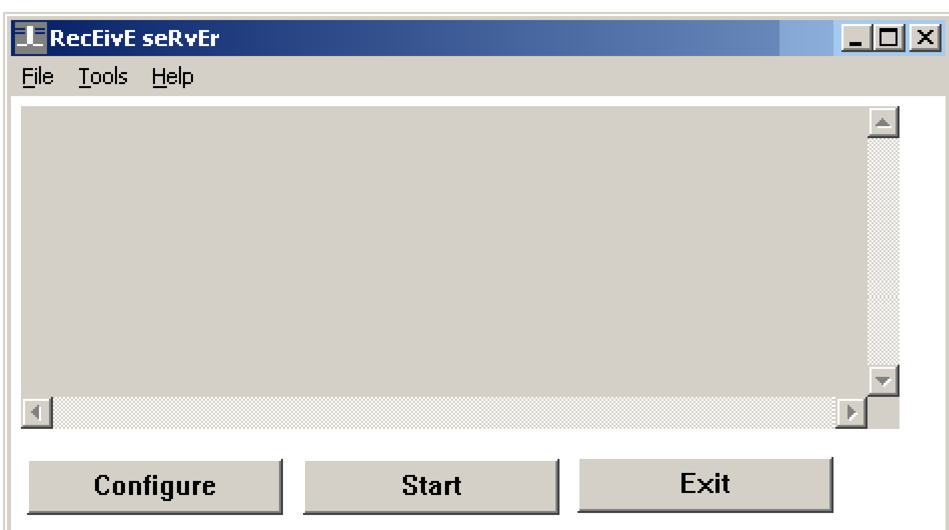



**Figure 28.** The server interface.

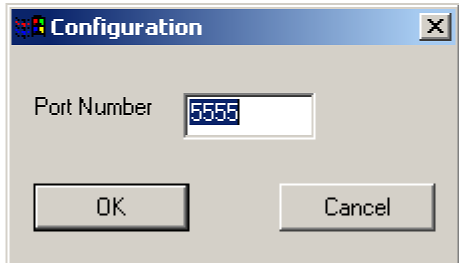



**Figure 29.** The server "Configuration" dialog box.

Under the "File" menu, the "Save to log file" option prompts for a file name and saves the connection status information to the specified file. By default, the file name is "data.txt." Under the "Tool" menu, the "Configure" and the "Start" submenus have the same functions as the "Configure" button and the "Start" button in the main window. The "Help" menu has the "About" option that displays the host name and the IP address of the server itself as shown in Figure 30.


**Figure 30.** The "About" dialog box.

After choosing the "Start" command, the server waits for connections from a client. When a connection is established, the server starts receiving data from the client. Figure 31 shows example experimental results saved in the "data.txt" file. Figure 32 shows a sample screen shot of the server when receiving data from a client.

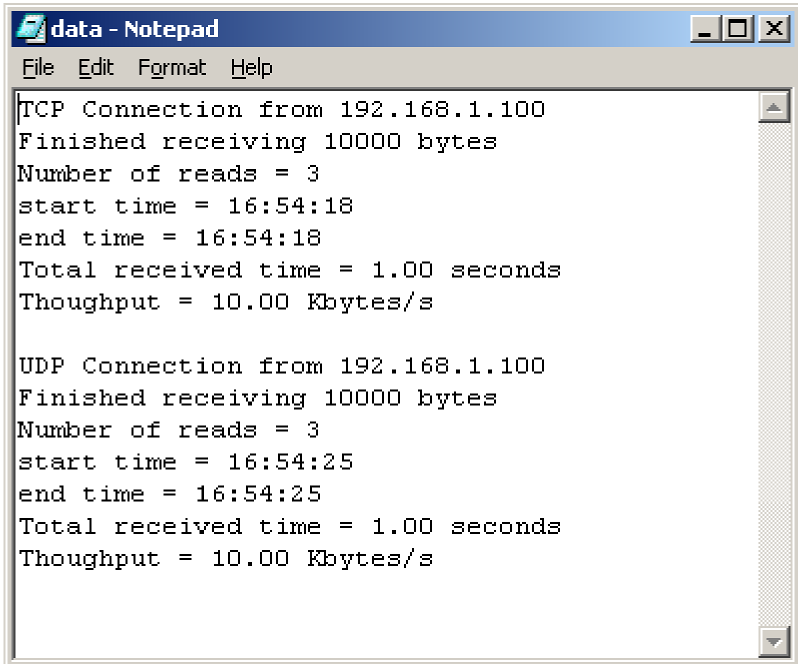



**Figure 31.** The server performance as saved in “data.txt” file.

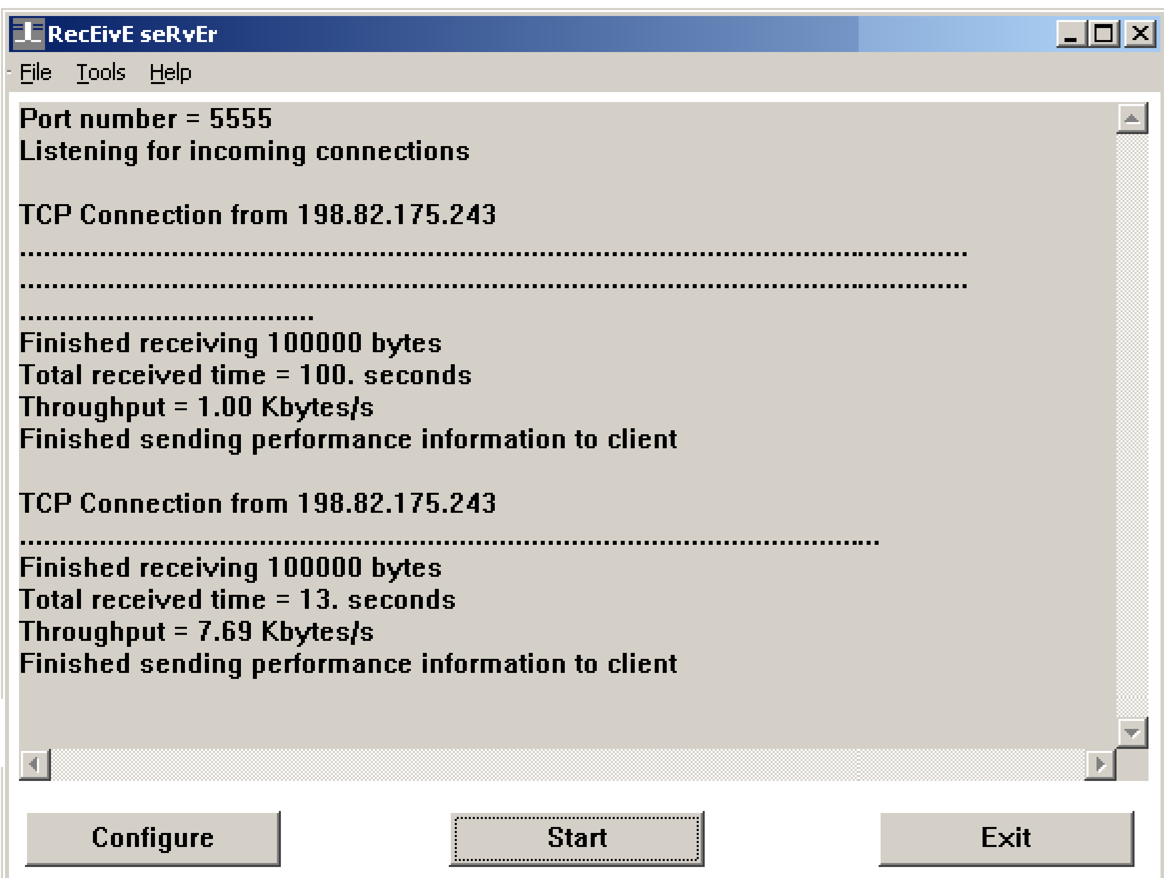



**Figure 32.** The server receives data from the client (PDA).

## A.2 Client Users' Manual

The client interface is shown in Figure 33 and the "Configuration" dialog box is shown in Figure 34. The client has more parameters to be defined than the server. The "Host" entry can be entered as host name or IP address of the server. There are three variations of transport protocols that can be chosen to setup a connection. "TCP" specifies a standard TCP connection, "TCP_NoDelay" specifies a TCP connection with the TCP_NODELAY option set, and "UDP" specifies use of standard UDP as the transport protocol. "Total bytes to send" denotes how much data, in bytes, the user would like to send to the server. "Block size" specifies the size of each data block to be sent, in bytes. The block size of each packet has to be smaller than or equal to the total number of bytes to send and cannot be larger than 65,535 bytes when UDP is the transport protocol. The "Port Number" value needs to be the same as the server's port number.

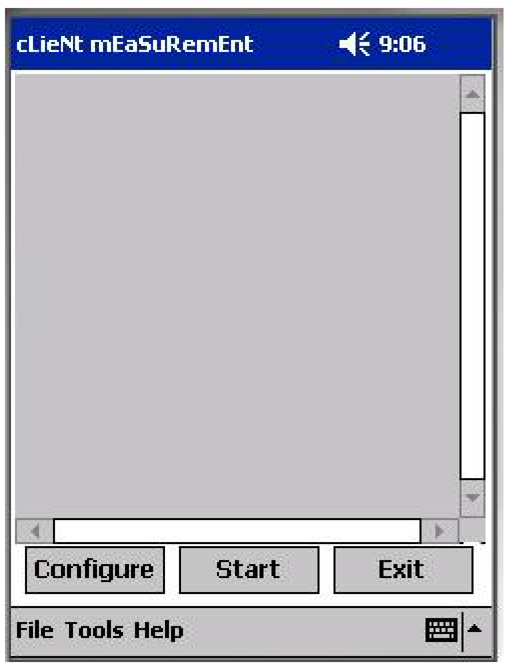



**Figure 33.** The client interface.

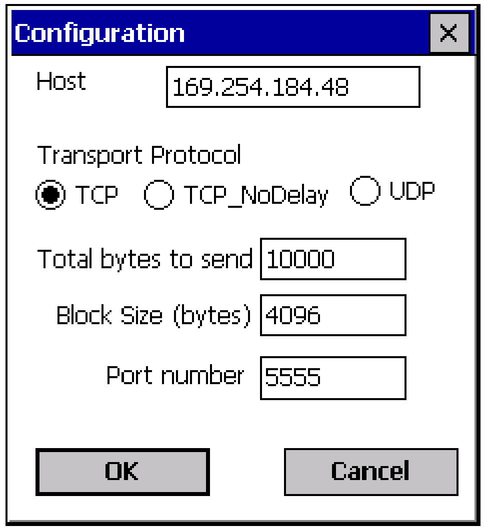



**Figure 34.** The client "Configuration" dialog box.

Click the "OK" button to save the configuration information or click "Cancel" otherwise. After finished setting up the connection information, select the "Start" button to start transmitting data to the server. Choosing the "Start" option under the "Tools" menu also performs the same operation. The client menu is the same as the server menu. Figure 35 displays the status of a sample test running between the client and the server, as seen on the PDA screen. Figure 36 displays the test results that are saved in the default "data.txt" file.

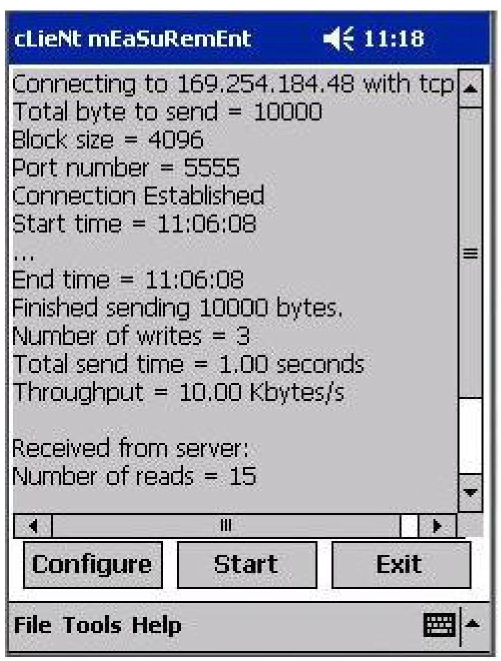
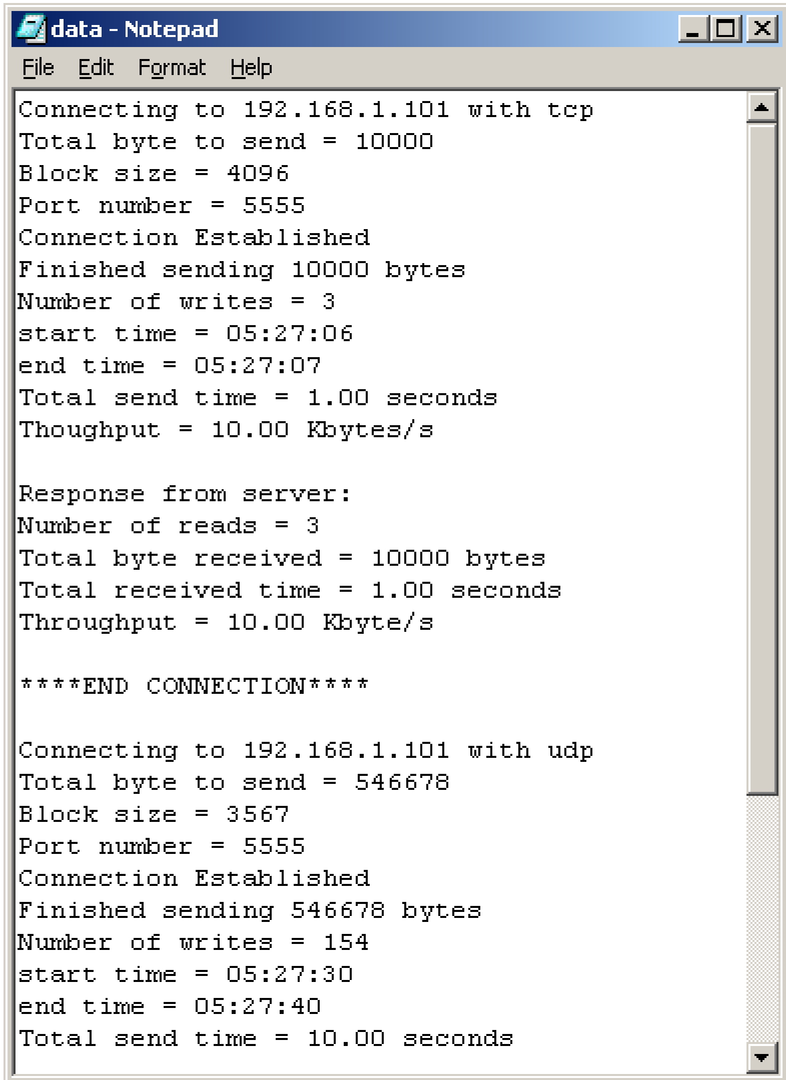



**Figure 35.** Displaying TCP connection information.




```
Connecting to 192.168.1.101 with tcp
Total byte to send = 10000
Block size = 4096
Port number = 5555
Connection Established
Finished sending 10000 bytes
Number of writes = 3
start time = 05:27:06
end time = 05:27:07
Total send time = 1.00 seconds
Thoughput = 10.00 Kbytes/s

Response from server:
Number of reads = 3
Total byte received = 10000 bytes
Total received time = 1.00 seconds
Throughput = 10.00 Kbyte/s

****END CONNECTION****

Connecting to 192.168.1.101 with udp
Total byte to send = 546678
Block size = 3567
Port number = 5555
Connection Established
Finished sending 546678 bytes
Number of writes = 154
start time = 05:27:30
end time = 05:27:40
Total send time = 10.00 seconds
```

**Figure 36.** The client and the server performance saved in “data.txt” file.

# Appendix B. Design Details

This section described the client and the server designs in details. Both programs are written in C/C++. The client program is developed using Microsoft's Embedded Visual Tools and the server program is developed using the Microsoft Visual Studio.

## B.1 Client Design

Programming a Windows CE application requires programming a Windows application. A Windows program uses a “push” model in which the program must be written to react to notifications from the operating system, such as that a key has been pressed or that a command has been typed. Windows applications do not ask for input from the operating system; the operating system notifies the application that an input has occurred. The operating system achieves these notifications by sending messages to an application window. Messages are sent for events, such as when a window is being moved or resized or when a key is pressed.

The Pocket PC architecture uses Unicode instead of ASCII for input and display data. Unicode is a standard representation for a character as a 16-bit value, as opposed to the ASCII standard of encoding a character into a single 7- or 8-bit value. Unicode allows for simple porting of programs to different international markets because all the known characters can be represented in one of the 65,536 available Unicode values. Instead of the standard char data type, TCHAR should be defined as char for the Windows 95/98 applications and unsigned short as char for application running on all other Windows operating system. Thus, conversions are needed to change Unicode to ASCII and vice-versa when parsing parameters through the socket API. The mbstowcs() function converts ASCII to Unicode and the wcstombs() function reverses the conversion. An example of how to use these functions is shown in Figure 37. The mbstowcs() function changes the “host” variable in ASCII to Unicode and stores it in the “uhost” variable, while the wcstombs() function converts the “szText” variable in Unicode to ASCII and stores it in the “host” variable.

```
// convert ASCII (host) to UNICODE (uhost)
#ifdef _UNICODE
        mbstowcs (uhost, host, strlen(host)+1);
#else
        lstrcpy (uhost, host);
#endif

// convert UNICODE (szText) to ASCII (host)
#ifdef _UNICODE
        wcstombs (host, szText, outchars);
#else
        lstrcpy (host, szText);
#endif
```

**Figure 37.** Unicode and ASCII conversion example.

A special feature of the iPAQ Pocket PC device is the soft input keyboard, which acts as a regular keyboard for inputting information using a "stylus" pen. The SipShowIM() command can be used to show and hide this soft input keyboard, as shown in Figure 38. As indicated, the SIPF_OFF parameter hides the soft input keyboard, while SIPF_ON shows the soft input keyboard.

```
SipShowIM (SIPF_OFF);     //hide soft input keyboard
SipShowIM (SIPF_ON);      //show soft input keyboard
```

**Figure 38.** Show and hide the soft input keyboard.

Unlike the server program, the client only creates two sockets, one is the data socket and one is the control socket. The data socket is set according to the user-defined protocol and port number. The control socket uses a port number that is one number higher than the data socket and always uses TCP as the transport protocol. For example, if the defined port number is 5555, the control port number will be 5556. The control socket is set in the Control() routine, and the data socket is set in the connectsock() routine. To create a TCP socket with TCP_NODELAY option selected, the client calls the setsockopt() API function with the TCP_NODELAY option enabled. In the transmit() routine, the client initializes the connection to the server by sending a notification message with the total number of bytes to be sent to the server through the control socket. After receiving an acknowledgment from the server, the client starts transmitting data in the user-specified block size until the total number of bytes to send is reached. After the client finishes sending all data, it waits for 2 seconds and then sends a "query" message to the server asking for its performance. The response packet is extracted using the strtok() command to obtain the server's performance data. Performance data from both the client and the server are shown on the screen and saved to a log file. The log file contains information including the total number of bytes sent and received, the block size, the elapsed times, the number of writes and reads, and the throughput. The client then tears down the connection after the transaction is completed. For each measurement, the client establishes a new connection to the server.

The elapsed times for sending the data are recorded to compute the client throughput. A time() routine is defined to calculate the elapsed time since the time format returns from the GetTimeFormat() function is HH:MM:SS, where HH is hour, MM is minutes, and SS is seconds.

## B.2 Server Design

The server program is also a Windows application, but runs on a Windows NT/2000/XP computer. Therefore, the programming is done using the Microsoft Visual C++. Data conversion is not needed for parameters used with the socket API functions and there is no soft input keyboard.

The server is a multiprotocol server that uses asynchronous I/O to handle connections over TCP and UDP. The server defines asynchronous sockets to wait and listen for incoming connection. Since the Windows program uses a "push" model, the operating system sends a notification message to the application window when an input has occurred. Thus, when the client connects to the server, a notification message is sent to the program to notify the corresponding listening socket to accept the connection. The WSAAsyncSelect() function is used to create three asynchronous sockets, a control socket, a TCP socket, and a UDP socket. All sockets are bound to an address and a port number. The listen() function is called to place the control socket and the

TCP sockets in passive mode. Depending on the transport protocol, a socket is notified to accept a connection or to read data from the buffer. FD_ACCEPT is used to accept incoming requests, FD_READ to read data, and FD_CLOSE to close the socket. When an FD_READ message is sent to the program, a thread is created to receive data. When all data has been read from the buffer, the thread is terminated.

The time when the first byte of data is read is recorded. The end time depends on either the server has received all data that the client has sent or when the server has received the "query" message from the client. Upon receiving the "query" message, the server sends its performance results to the client immediately even though it might not have finished receiving all data that the client has sent. The performance packet is constructed with the sprintf() function, where the number of reads, the elapsed time, the total number of bytes received, and the throughput are written to a string. For TCP, the server closes the newly created socket, but keeps the listening socket open.

# Appendix C. Performance data

This appendix shows the performance data collected from the experiments.

**Table 15.** TCP Performance Measured in Randolph Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 7 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 4 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 14 | 0.71 | 200 | 13 | 0.77 | 62 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 28 | 10000 |
| 10000 | 500 | 2 | 5 | 20 | 1 | 10 | 13 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 8 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 6 | 10000 |
| 100000 | 50 | 218 | 0.46 | 2000 | 218 | 0.46 | 996 | 100000 |
| 100000 | 100 | 98 | 1.02 | 1000 | 99 | 1.01 | 450 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 13 | 7.69 | 131 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 2 | 50 | 20 | 1 | 100 | 80 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 70 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 73 | 100000 |

**Table 16.** TCP Performance with the TCP_NODELAY Option Measured in Randolph Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 7 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 4 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 14 | 0.71 | 200 | 13 | 0.77 | 62 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 5 | 2 | 28 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 10 | 13 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 8 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 220 | 0.45 | 2000 | 220 | 0.45 | 1006 | 100000 |
| 100000 | 100 | 102 | 0.98 | 1000 | 102 | 0.98 | 468 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 13 | 7.69 | 132 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 7 | 14.29 | 100 | 100000 |
| 100000 | 5000 | 2 | 50 | 20 | 1 | 100 | 79 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 70 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 73 | 100000 |

**Table 17.** UDP Performance Measured in Randolph Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 20 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 10 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 13 | 0.77 | 200 | 14 | 0.71 | 200 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 100 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 10 | 20 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 2 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 1 | 10000 |
| 100000 | 50 | 224 | 0.45 | 2000 | 225 | 0.44 | 2000 | 100000 |
| 100000 | 100 | 103 | 0.97 | 1000 | 102 | 0.98 | 1000 | 100000 |
| 100000 | 500 | 13 | 7.69 | 200 | 14 | 7.14 | 200 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 20 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 10 | 100000 |
| 100000 | 50000 | 1 | 100 | 2 | 1 | 100 | 2 | 100000 |

**Table 18.** TCP Performance with VPN Measured in Randolph Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 3 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 2 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 14 | 0.71 | 200 | 14 | 0.71 | 23 | 10000 |
| 10000 | 100 | 5 | 2 | 100 | 6 | 1.67 | 15 | 10000 |
| 10000 | 500 | 2 | 5 | 20 | 2 | 5 | 15 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 9 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 7 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 218 | 0.46 | 2000 | 218 | 0.46 | 298 | 100000 |
| 100000 | 100 | 101 | 0.99 | 1000 | 101 | 0.99 | 159 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 14 | 7.14 | 145 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 6 | 16.67 | 99 | 100000 |
| 100000 | 5000 | 2 | 50 | 20 | 1 | 100 | 75 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 71 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 72 | 100000 |

**Table 19.** TCP Performance with the TCP_NODELAY Option and VPN Measured in Randolph Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 2 | 0.5 | 4 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 2 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 1 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 13 | 0.77 | 200 | 13 | 0.77 | 29 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 19 | 10000 |
| 10000 | 500 | 2 | 5 | 20 | 2 | 5 | 15 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 7 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 217 | 0.46 | 2000 | 219 | 0.46 | 343 | 100000 |
| 100000 | 100 | 100 | 1 | 1000 | 101 | 0.99 | 179 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 14 | 7.14 | 145 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 7 | 14.29 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 76 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 70 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 72 | 100000 |

**Table 20.** UDP Performance with VPN Measured in Randolph Hall

| Total bytes send | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 20 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 10 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 14 | 0.71 | 200 | 13 | 0.77 | 200 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 100 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 10 | 20 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 2 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 1 | 10000 |
| 100000 | 50 | 223 | 0.45 | 2000 | 222 | 0.45 | 2000 | 100000 |
| 100000 | 100 | 102 | 0.98 | 1000 | 101 | 0.99 | 1000 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 13 | 7.69 | 200 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 2 | 50 | 20 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 10 | 100000 |

**Table 21.** TCP Performance Measured in Torgersen Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 2 | 0.5 | 3 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 4 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 13 | 0.77 | 200 | 13 | 0.77 | 61 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 28 | 10000 |
| 10000 | 500 | 2 | 5 | 20 | 1 | 10 | 13 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 7 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 214 | 0.47 | 2000 | 214 | 0.47 | 971 | 100000 |
| 100000 | 100 | 100 | 1 | 1000 | 100 | 1 | 456 | 100000 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.69 | 128 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 2 | 50 | 20 | 2 | 50 | 77 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 69 | 100000 |
| 100000 | 50000 | 1 | 100 | 2 | 1 | 100 | 72 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 72 | 100000 |

**Table 22.** TCP Performance with the TCP_NODELAY Option Measured in Torgersen Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 6 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 4 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 13 | 0.77 | 200 | 13 | 0.77 | 61 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 28 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 10 | 13 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 8 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 214 | 0.47 | 2000 | 214 | 0.47 | 978 | 100000 |
| 100000 | 100 | 97 | 1.03 | 1000 | 97 | 1.03 | 445 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 13 | 7.69 | 127 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 78 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 71 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 72 | 100000 |

**Table 23.** UDP Performance Measured in Torgersen Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 20 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 10 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 13 | 0.77 | 200 | 13 | 0.77 | 200 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 100 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 10 | 20 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 2 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 1 | 10000 |
| 100000 | 50 | 216 | 0.46 | 2000 | 216 | 0.46 | 1998 | 99900 |
| 100000 | 100 | 100 | 1 | 1000 | 100 | 1 | 1000 | 100000 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.69 | 200 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 20 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 10 | 100000 |
| 100000 | 50000 | 1 | 100 | 2 | 1 | 100 | 2 | 100000 |

**Table 24.** TCP Performance with VPN Measured in Torgersen Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 4 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 3 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 13 | 0.77 | 200 | 14 | 0.71 | 30 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 15 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 10 | 15 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 7 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 220 | 0.45 | 2000 | 221 | 0.45 | 428 | 100000 |
| 100000 | 100 | 101 | 0.99 | 1000 | 104 | 0.96 | 192 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 14 | 7.14 | 145 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 7 | 14.29 | 99 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 68 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 73 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 70 | 100000 |

**Table 25.** TCP Performance with the TCP_NODELAY Option and VPN Measured in Torgersen Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 3 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 2 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 14 | 0.71 | 200 | 14 | 0.71 | 32 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 18 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 10 | 14 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 6 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 220 | 0.45 | 2000 | 220 | 0.45 | 398 | 100000 |
| 100000 | 100 | 102 | 0.98 | 1000 | 103 | 0.97 | 226 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 14 | 7.14 | 144 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 2 | 50 | 74 | 100000 |
| 100000 | 10000 | 2 | 50 | 10 | 1 | 100 | 68 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 72 | 100000 |

**Table 26.** UDP Performance with VPN Measured in Torgersen Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 20 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 10 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 14 | 0.71 | 200 | 14 | 0.71 | 200 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 100 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 10 | 20 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 2 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 1 | 10000 |
| 100000 | 50 | 222 | 0.45 | 2000 | 222 | 0.45 | 2000 | 100000 |
| 100000 | 100 | 102 | 0.98 | 1000 | 102 | 0.98 | 1000 | 100000 |
| 100000 | 500 | 14 | 7.14 | 200 | 14 | 7.14 | 200 | 100000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 20 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 10 | 100000 |
| 100000 | 40000 | 1 | 100 | 3 | 1 | 100 | 3 | 100000 |

**Table 27.** TCP Performance Measured in Pamplin Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 2.00 | 0.50 | 20 | 1.00 | 1.00 | 7 | 1000 |
| 1000 | 100 | 1.00 | 1.00 | 10 | 1.00 | 1.00 | 4 | 1000 |
| 1000 | 500 | 1.00 | 1.00 | 2 | 1.00 | 1.00 | 2 | 1000 |
| 1000 | 1000 | 1.00 | 1.00 | 1 | 1.00 | 1.00 | 1 | 1000 |
| 10000 | 50 | 13.00 | 0.77 | 200 | 13.00 | 0.77 | 62 | 10000 |
| 10000 | 100 | 7.00 | 1.43 | 100 | 6.00 | 1.67 | 29 | 10000 |
| 10000 | 500 | 1.00 | 10.00 | 20 | 2.00 | 5.00 | 13 | 10000 |
| 10000 | 1000 | 1.00 | 10.00 | 10 | 1.00 | 10.00 | 10 | 10000 |
| 10000 | 5000 | 1.00 | 10.00 | 2 | 1.00 | 10.00 | 8 | 10000 |
| 10000 | 10000 | 1.00 | 10.00 | 1 | 1.00 | 10.00 | 7 | 10000 |
| 100000 | 50 | 220.00 | 0.45 | 2000 | 219.00 | 0.46 | 1004 | 100000 |
| 100000 | 100 | 101.00 | 0.99 | 1000 | 101.00 | 0.99 | 463 | 100000 |
| 100000 | 500 | 13.00 | 7.69 | 200 | 13.00 | 7.69 | 130 | 100000 |
| 100000 | 1000 | 6.00 | 16.67 | 100 | 6.00 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1.00 | 100.00 | 20 | 2.00 | 50.00 | 80 | 100000 |
| 100000 | 10000 | 1.00 | 100.00 | 10 | 1.00 | 100.00 | 71 | 100000 |
| 100000 | 100000 | 1.00 | 100.00 | 1 | 1.00 | 100.00 | 72 | 100000 |

**Table 28.** TCP Performance with the TCP_ NODELAY Option Measured in Pamplin Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1.00 | 1.00 | 20 | 1.00 | 1.00 | 6 | 1000 |
| 1000 | 100 | 1.00 | 1.00 | 10 | 1.00 | 1.00 | 4 | 1000 |
| 1000 | 500 | 1.00 | 1.00 | 2 | 1.00 | 1.00 | 2 | 1000 |
| 1000 | 1000 | 1.00 | 1.00 | 1 | 1.00 | 1.00 | 1 | 1000 |
| 10000 | 50 | 13.00 | 0.77 | 200 | 13.00 | 0.77 | 62 | 10000 |
| 10000 | 100 | 6.00 | 1.67 | 100 | 6.00 | 1.67 | 29 | 10000 |
| 10000 | 500 | 2.00 | 5.00 | 20 | 1.00 | 10.00 | 13 | 10000 |
| 10000 | 1000 | 1.00 | 10.00 | 10 | 1.00 | 10.00 | 10 | 10000 |
| 10000 | 5000 | 1.00 | 10.00 | 2 | 1.00 | 10.00 | 8 | 10000 |
| 10000 | 10000 | 1.00 | 10.00 | 1 | 1.00 | 10.00 | 7 | 10000 |
| 100000 | 50 | 221.00 | 0.45 | 2000 | 221.00 | 0.45 | 1010 | 100000 |
| 100000 | 100 | 101.00 | 0.99 | 1000 | 101.00 | 0.99 | 463 | 100000 |
| 100000 | 500 | 14.00 | 7.14 | 200 | 13.00 | 7.69 | 131 | 100000 |
| 100000 | 1000 | 6.00 | 16.67 | 100 | 6.00 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1.00 | 100.00 | 20 | 2.00 | 50.00 | 77 | 100000 |
| 100000 | 10000 | 1.00 | 100.00 | 10 | 1.00 | 100.00 | 69 | 100000 |
| 100000 | 100000 | 1.00 | 100.00 | 1 | 1.00 | 100.00 | 72 | 100000 |

**Table 29.** UDP Performance Measured in Pamplin Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1.00 | 1.00 | 20 | 1.00 | 1.00 | 20 | 1000 |
| 1000 | 100 | 1.00 | 1.00 | 10 | 1.00 | 1.00 | 10 | 1000 |
| 1000 | 500 | 1.00 | 1.00 | 2 | 1.00 | 1.00 | 2 | 1000 |
| 1000 | 1000 | 1.00 | 1.00 | 1 | 1.00 | 1.00 | 1 | 1000 |
| 10000 | 50 | 14.00 | 0.71 | 200 | 13.00 | 0.77 | 200 | 10000 |
| 10000 | 100 | 6.00 | 1.67 | 100 | 6.00 | 1.67 | 100 | 10000 |
| 10000 | 500 | 1.00 | 10.00 | 20 | 1.00 | 10.00 | 20 | 10000 |
| 10000 | 1000 | 1.00 | 10.00 | 10 | 1.00 | 10.00 | 10 | 10000 |
| 10000 | 5000 | 1.00 | 10.00 | 2 | 1.00 | 10.00 | 2 | 10000 |
| 10000 | 10000 | 1.00 | 10.00 | 1 | 1.00 | 10.00 | 1 | 10000 |
| 100000 | 50 | 221.00 | 0.45 | 2000 | 222.00 | 0.45 | 2000 | 100000 |
| 100000 | 100 | 101.00 | 0.99 | 1000 | 100.00 | 1.00 | 1000 | 100000 |
| 100000 | 500 | 13.00 | 7.69 | 200 | 13.00 | 7.69 | 200 | 100000 |
| 100000 | 1000 | 6.00 | 16.67 | 100 | 6.00 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1.00 | 100.00 | 20 | 1.00 | 100.00 | 20 | 100000 |
| 100000 | 10000 | 1.00 | 100.00 | 10 | 1.00 | 100.00 | 10 | 100000 |
| 100000 | 50000 | 1.00 | 100.00 | 2 | 1.00 | 100.00 | 2 | 100000 |

**Table 30.** TCP Performance with VPN Measured in Pamplin Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1.00 | 1.00 | 20 | 1.00 | 1.00 | 3 | 1000 |
| 1000 | 100 | 1.00 | 1.00 | 10 | 1.00 | 1.00 | 3 | 1000 |
| 1000 | 500 | 1.00 | 1.00 | 2 | 1.00 | 1.00 | 2 | 1000 |
| 1000 | 1000 | 1.00 | 1.00 | 1 | 1.00 | 1.00 | 1 | 1000 |
| 10000 | 50 | 13.00 | 0.77 | 200 | 14.00 | 0.71 | 23 | 10000 |
| 10000 | 100 | 6.00 | 1.67 | 100 | 6.00 | 1.67 | 15 | 10000 |
| 10000 | 500 | 1.00 | 10.00 | 20 | 1.00 | 10.00 | 15 | 10000 |
| 10000 | 1000 | 1.00 | 10.00 | 10 | 1.00 | 10.00 | 10 | 10000 |
| 10000 | 5000 | 1.00 | 10.00 | 2 | 1.00 | 10.00 | 7 | 10000 |
| 10000 | 10000 | 1.00 | 10.00 | 1 | 1.00 | 10.00 | 7 | 10000 |
| 100000 | 50 | 220.00 | 0.45 | 2000 | 220.00 | 0.45 | 365 | 100000 |
| 100000 | 100 | 101.00 | 0.99 | 1000 | 100.00 | 1.00 | 176 | 100000 |
| 100000 | 500 | 14.00 | 7.14 | 200 | 14.00 | 7.14 | 140 | 100000 |
| 100000 | 1000 | 6.00 | 16.67 | 100 | 7.00 | 14.29 | 99 | 100000 |
| 100000 | 5000 | 1.00 | 100.00 | 20 | 1.00 | 100.00 | 74 | 100000 |
| 100000 | 10000 | 1.00 | 100.00 | 10 | 1.00 | 100.00 | 72 | 100000 |
| 100000 | 50000 | 1.00 | 100.00 | 2 | 1.00 | 100.00 | 69 | 100000 |
| 100000 | 100000 | 1.00 | 100.00 | 1 | 1.00 | 100.00 | 71 | 100000 |

**Table 31.** TCP Performance with the TCP_NODELAY Option and VPN Measured in Pamplin Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1.00 | 1.00 | 20 | 1.00 | 1.00 | 3 | 1000 |
| 1000 | 100 | 1.00 | 1.00 | 10 | 1.00 | 1.00 | 2 | 1000 |
| 1000 | 500 | 1.00 | 1.00 | 2 | 1.00 | 1.00 | 2 | 1000 |
| 1000 | 1000 | 1.00 | 1.00 | 1 | 1.00 | 1.00 | 1 | 1000 |
| 10000 | 50 | 13.00 | 0.77 | 200 | 14.00 | 0.71 | 32 | 10000 |
| 10000 | 100 | 6.00 | 1.67 | 100 | 6.00 | 1.67 | 16 | 10000 |
| 10000 | 500 | 1.00 | 10.00 | 20 | 1.00 | 10.00 | 13 | 10000 |
| 10000 | 1000 | 1.00 | 10.00 | 10 | 1.00 | 10.00 | 10 | 10000 |
| 10000 | 5000 | 1.00 | 10.00 | 2 | 1.00 | 10.00 | 7 | 10000 |
| 10000 | 10000 | 1.00 | 10.00 | 1 | 1.00 | 10.00 | 7 | 10000 |
| 100000 | 50 | 218.00 | 0.46 | 2000 | 219.00 | 0.46 | 314 | 100000 |
| 100000 | 100 | 101.00 | 0.99 | 1000 | 101.00 | 0.99 | 174 | 100000 |
| 100000 | 500 | 14.00 | 7.14 | 200 | 14.00 | 7.14 | 140 | 100000 |
| 100000 | 1000 | 7.00 | 14.29 | 100 | 6.00 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 2.00 | 50.00 | 20 | 1.00 | 100.00 | 73 | 100000 |
| 100000 | 10000 | 1.00 | 100.00 | 10 | 1.00 | 100.00 | 71 | 100000 |
| 100000 | 50000 | 1.00 | 100.00 | 2 | 1.00 | 100.00 | 72 | 100000 |
| 100000 | 100000 | 1.00 | 100.00 | 1 | 1.00 | 100.00 | 72 | 100000 |

**Table 32.** UDP Performance with VPN Measured in Pamplin Hall

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1.00 | 1.00 | 20 | 1.00 | 1.00 | 20 | 1000 |
| 1000 | 100 | 1.00 | 1.00 | 10 | 1.00 | 1.00 | 10 | 1000 |
| 1000 | 500 | 1.00 | 1.00 | 2 | 1.00 | 1.00 | 2 | 1000 |
| 1000 | 1000 | 1.00 | 1.00 | 1 | 1.00 | 1.00 | 1 | 1000 |
| 10000 | 50 | 14.00 | 0.71 | 200 | 13.00 | 0.77 | 200 | 10000 |
| 10000 | 100 | 6.00 | 1.67 | 100 | 6.00 | 1.67 | 100 | 10000 |
| 10000 | 500 | 1.00 | 10.00 | 20 | 1.00 | 10.00 | 20 | 10000 |
| 10000 | 1000 | 1.00 | 10.00 | 10 | 1.00 | 10.00 | 10 | 10000 |
| 10000 | 5000 | 1.00 | 10.00 | 2 | 1.00 | 10.00 | 2 | 10000 |
| 10000 | 10000 | 1.00 | 10.00 | 1 | 1.00 | 10.00 | 1 | 10000 |
| 100000 | 50 | 223.00 | 0.45 | 2000 | 222.00 | 0.45 | 2000 | 100000 |
| 100000 | 100 | 102.00 | 0.98 | 1000 | 101.00 | 0.99 | 1000 | 100000 |
| 100000 | 500 | 14.00 | 7.14 | 200 | 13.00 | 7.69 | 200 | 100000 |
| 100000 | 1000 | 6.00 | 16.67 | 100 | 6.00 | 16.67 | 100 | 100000 |
| 100000 | 5000 | 1.00 | 100.00 | 20 | 2.00 | 50.00 | 20 | 100000 |
| 100000 | 10000 | 1.00 | 100.00 | 10 | 1.00 | 100.00 | 10 | 100000 |
| 100000 | 40000 | 1.00 | 100.00 | 3 | 1.00 | 100.00 | 3 | 100000 |

**Table 33.** TCP Performance for Off-campus Scenario

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 6 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 4 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 13 | 0.77 | 200 | 13 | 0.77 | 48 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 21 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 4 | 2.5 | 10 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 8 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 8 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 216 | 0.46 | 2000 | 216 | 0.46 | 859 | 100000 |
| 100000 | 100 | 100 | 1 | 1000 | 100 | 1 | 412 | 100000 |
| 100000 | 500 | 15 | 6.67 | 200 | 15 | 6.67 | 110 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 7 | 14.29 | 73 | 100000 |
| 100000 | 5000 | 2 | 50 | 20 | 2 | 50 | 56 | 100000 |
| 100000 | 10000 | 2 | 50 | 10 | 1 | 100 | 59 | 100000 |
| 100000 | 50000 | 1 | 100 | 2 | 1 | 100 | 54 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 6 | 100000 |

**Table 34.** TCP Performance with the TCP_NODELAY option for Off-campus Scenario

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 6 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 4 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 12 | 0.83 | 200 | 12 | 0.83 | 53 | 10000 |
| 10000 | 100 | 5 | 2 | 100 | 4 | 2.5 | 16 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 2 | 5 | 12 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 8 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 216 | 0.46 | 2000 | 216 | 0.46 | 844 | 100000 |
| 100000 | 100 | 99 | 1.01 | 1000 | 99 | 1.01 | 397 | 100000 |
| 100000 | 500 | 12 | 8.33 | 200 | 12 | 8.33 | 109 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 7 | 14.29 | 82 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 63 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 100 | 61 | 100000 |
| 100000 | 50000 | 1 | 100 | 2 | 1 | 100 | 59 | 100000 |
| 100000 | 100000 | 1 | 100 | 1 | 1 | 100 | 65 | 100000 |

**Table 35.** UDP Performance for Off-campus Scenario

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 20 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 10 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 0.5 | 1 | 500 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 12 | 0.83 | 200 | 13 | 0.71 | 184 | 9200 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.63 | 98 | 9800 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 9 | 18 | 9000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 2 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 1 | 10000 |
| 100000 | 50 | 219 | 0.46 | 2000 | 222 | 0.43 | 1902 | 95100 |
| 100000 | 100 | 100 | 1 | 1000 | 101 | 0.94 | 951 | 95100 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.15 | 186 | 93000 |
| 100000 | 1000 | 6 | 16.67 | 100 | 6 | 15.67 | 94 | 94000 |
| 100000 | 5000 | 2 | 50 | 20 | 2 | 42.5 | 17 | 85000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 70 | 7 | 70000 |
| 100000 | 50000 | 1 | 100 | 2 | 2 | 25 | 1 | 50000 |

**Table 36:** TCP Performance with VPN for Off-campus Scenario

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 3 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 2 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 12 | 0.83 | 200 | 11 | 0.91 | 14 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 17 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 10 | 15 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 7 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 218 | 0.46 | 2000 | 219 | 0.46 | 372 | 100000 |
| 100000 | 100 | 99 | 1.01 | 1000 | 99 | 1.01 | 197 | 100000 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.69 | 126 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 7 | 14.29 | 88 | 100000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 100 | 73 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 3 | 33.33 | 57 | 100000 |
| 100000 | 50000 | 3 | 33.33 | 2 | 5 | 20 | 64 | 100000 |
| 100000 | 100000 | 8 | 12.5 | 1 | 5 | 20 | 66 | 100000 |

**Table 37.** TCP Performance with the TCP_NODELAY option and VPN for Off-campus Scenario

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 2 | 0.5 | 20 | 2 | 0.5 | 4 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 1 | 3 | 1000 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 12 | 0.83 | 200 | 9 | 1.11 | 25 | 10000 |
| 10000 | 100 | 6 | 1.67 | 100 | 6 | 1.67 | 15 | 10000 |
| 10000 | 500 | 1 | 10 | 20 | 9 | 1.11 | 14 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 7 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 7 | 10000 |
| 100000 | 50 | 217 | 0.46 | 2000 | 217 | 0.46 | 374 | 100000 |
| 100000 | 100 | 100 | 1 | 1000 | 100 | 1 | 224 | 100000 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.69 | 116 | 100000 |
| 100000 | 1000 | 7 | 14.29 | 100 | 7 | 14.29 | 91 | 100000 |
| 100000 | 5000 | 6 | 16.67 | 20 | 6 | 16.67 | 66 | 100000 |
| 100000 | 10000 | 1 | 100 | 10 | 3 | 33.33 | 61 | 100000 |
| 100000 | 50000 | 4 | 25 | 2 | 5 | 20 | 64 | 100000 |
| 100000 | 100000 | 4 | 25 | 1 | 5 | 20 | 64 | 100000 |

**Table 38.** UDP Performance with VPN for Off-campus Scenario

| Total bytes send (bytes) | Block size | Client send time (second) | Client throughput (Kbytes/s) | Number of writes | Server receive time (second) | Server throughput (Kbytes/s) | Number of reads | Total bytes received |
|---|---|---|---|---|---|---|---|---|
| 1000 | 50 | 1 | 1 | 20 | 1 | 1 | 20 | 1000 |
| 1000 | 100 | 1 | 1 | 10 | 1 | 0.7 | 7 | 700 |
| 1000 | 500 | 1 | 1 | 2 | 1 | 1 | 2 | 1000 |
| 1000 | 1000 | 1 | 1 | 1 | 1 | 1 | 1 | 1000 |
| 10000 | 50 | 13 | 0.77 | 200 | 13 | 0.68 | 176 | 8800 |
| 10000 | 100 | 5 | 2 | 100 | 6 | 1.38 | 83 | 8300 |
| 10000 | 500 | 1 | 10 | 20 | 1 | 10 | 20 | 10000 |
| 10000 | 1000 | 1 | 10 | 10 | 1 | 10 | 10 | 10000 |
| 10000 | 5000 | 1 | 10 | 2 | 1 | 10 | 2 | 10000 |
| 10000 | 10000 | 1 | 10 | 1 | 1 | 10 | 1 | 10000 |
| 100000 | 50 | 222 | 0.45 | 2000 | 223 | 0.41 | 1822 | 91100 |
| 100000 | 100 | 102 | 0.98 | 1000 | 102 | 0.9 | 917 | 91700 |
| 100000 | 500 | 13 | 7.69 | 200 | 13 | 7.31 | 190 | 95000 |
| 100000 | 1000 | 5 | 20 | 100 | 6 | 15.17 | 91 | 91000 |
| 100000 | 5000 | 1 | 100 | 20 | 1 | 85 | 17 | 85000 |
| 100000 | 10000 | 1 | 100 | 10 | 1 | 90 | 9 | 90000 |
| 100000 | 40000 | 3 | 100 | 3 | 2 | 20 | 1 | 40000 |

# Vita

SiewYeen Agnes Tan was born in Ipoh, Perak of Malaysia. She graduated from Ave Maria Convent Secondary School in December of 1995. Agnes came to U.S. in June 1996 to pursue her education at Virginia Tech. She received her B.S. degree in Computer Engineering in May 2001 and continued her study as a Master student at Virginia Tech. This thesis completes her M.S. degree in Computer Engineering from Virginia Tech.