

NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

INTERNET BASED AUTOMATIC CONTROL & MONITORING

**Graduation Project
COM 400**

**Submitted by: Babar Rehman &
Mohammad Nauman**

Submitted to: Asst. Prof. Dr. Dogan Ibrahim

Lefkosa 2000

TABLE OF CONTENTS

Acknowledgement	i
Abstract	ii
Abbreviations	iii
Introduction	vi

Chapter 1

Introduction to Network & Client/Server

1.1 Networking	1
1.2 Use of Network	1
1.3 Data	2
1.4 Applications	2
1.5 Network Topology	2
1.5.1 Bus	2
1.5.2 Star	2
1.5.3 Ring	3
1.6 Range of Networks	3
1.6.1 Local Area Networks (LAN)	3
1.6.2 Wide Area Networks (WAN)	3
1.6.3 Inter-Connected Networks (Inter Network)	3
1.6.4 Metropolitan Area Networks (MAN)	3
1.7 Networking Overview	4
1.7.1 Peer-to-Peer Networks	5
1.7.2 Server-Based Networks	5
1.7.3 Specialized Servers	6
1.8 Client/Server Computing	7
1.8.1 Client	7
1.8.2 Server	7
1.8.3 The Client/Server Process	8
1.9 The Advantages of Working in a Client/Server Environment	8

Chapter 2

Introduction to TCP/IP

2.1 Introduction of TCP/IP	10
2.2 The Evolution of TCP/IP & Internet	12
2.3 Architectural Model	15
2.4 IP Address	17
2.4.1 Classes of IP Addresses	17
2.4.1.1 Class A	18
2.4.1.2 Class B	18
2.4.1.3 Class C	18
2.5 Diagnosing and Solving IP Configuration Problems	19

Chapter 3

Windows Sockets (Winsock)

3.1 Background of Winsock	23
3.2 Definition of a Socket	24
3.3 The Socket Data Type	24
3.3.1 Uses for Sockets	25
3.3.2 Byte Ordering	25
3.3.3 Convert Byte Orders	26
3.3.4 When We Don't Have to Convert Byte Orders	26
3.4 Sequence of Operations for a Stream Socket Communication	27
3.5 Winsock Control	28
3.5.1 Possible Uses	28
3.5.2 Selecting a Protocol	29
3.5.3 Setting the Protocol	29
3.5.4 Determining the Name of Computer	29
3.5.5 To find the computer's name	30
3.6 TCP Connection Basics	30
3.6.1 To create a TCP server	30
3.6.2 To create a TCP Client	31
3.6.3 Accepting More than One Connection Request	31
3.7 UDP Connection Base Basics	32
3.8 Local Port Property	32

Chapter 4

TCP/IP Programming Using Visual Basic

4.1 TCP/IP Programming	33
4.2 Sockets	34
4.3 Winsock Control	35
4.3.1 Properties	35
4.3.2 Methods	37
4.3.3 Events	40
4.4 Client-Server Application	41
4.5 A Simple TCP/IP Example	42
4.5.1 One Way Communication	42
4.5.1.1 Client Computer	43
4.5.1.2 Server Computer	46
4.5.2 Two Way Communication	47
4.5.2.1 Client Computer	48
4.5.2.2 Server Computer	50
4.6 Testing the Programs	51
Conclusion	53
Bibliography	54

ACKNOWLEDGEMENTS

First of all we are happy that Almighty Allah Taa'la and Hazrat Muhammad (peace be upon him) has provided us the strength courage and knowledge to achieve the task, and also we are grateful especially to our dearest parents in our lives who have supported us, advised us, taught us that no dream is unachievable and encouraged us to follow our dreams and ambitions. We wish our mothers lives happily always as the rose lives in the spring and our fathers in the heaven are proud of us.

We wish to thank my advisor. Assistant Professor Dr. Dogan Ibrahim who is the river of knowledge we just took one drop to perform this job. His intellectual support, encouragement, patience for correcting both my stylistic and scientific errors and enthusiasm made this project possible.

Our Earnest thanks must go to our friends especially to Mr. Irfan Bashir, Mr. Nasir Durrani, Mr. Syed Jawad Ali, Mr. Naveed Mustafa and finally Mr. Shahid Islam those shared their suggestions and evaluations throughout the completion of the project and in our graduation.

We thank Allah Taa'la for giving us courage and strength to achieve aims and objectives of the life.

ABSTRACT

This project is about automatic control of external hardware over the Internet, using the TCP/IP communications protocol. The project is based on a client-server model and the principles of this model are described in detail. The project also describes the data exchange between two computers using the well-known TCP/IP protocol.

This project is theoretical and the Visual Basic language, together with the Winsock ActiveX control component is used to demonstrate the principles of socket-based programming.

ABBREVIATIONS

ARP	Address Resolution Protocol
ARPANET	Advanced Research Projects Agency Network
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
BGP	Border Gateway Protocol
BSD	Berkeley Software Development
CCITT	International Telegraph and Telephone Consultative Committee
CIX	Commercial Internet Exchange
DARPA	Defense Advanced Research Projects Agency
DNS	Domain Name System
DoD	U.S. Department of Defense
FAQ	Frequently Asked Questions lists
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
FYI	For Your Information series of RFCs
GOSIP	U.S. Government Open Systems Interconnection Profile
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IAB	Internet Activities Board
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol

IESG	Internet Engineering Steering Group
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
ISOC	Internet Society
ITU-T	International Telecommunication Union Standardization Sector
MAC	Medium (or media) access control
Mbps	Megabits (millions of bits) per second
NICNAME	Network Information Center name service
NSF	National Science Foundation
NSFNET	National Science Foundation Network
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PPP	Point-to-Point Protocol
RARP	Reverse Address Resolution Protocol
RIP	Routing Information Protocol
RFC	Request For Comments
SLIP	Serial Line IP
SMDS	Switched Multimegabit Data Service
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
STD	Internet Standards series of RFCs
TCP	Transmission Control Protocol

TLD	Top-level domain
UDP	User Datagram Protocol
WINSOCK	Windows Socket

INTRODUCTION

The Bartering, exchanging, of things is and has been a great demand of the century and still most wanted. For the sake of the barter system, as in history, there should be some sort of dealings. As we are in 21st century so the old barter system have been developed and mostly dealings or the purchase & sale is done over the internet; the biggest WAN in the world through which all most every person is connected. The finest example of the purchase & sale over Internet is e-commerce and by using this technology many persons even can sell their products without permitting their bodies to move. Over the Internet there is also necessary thing mail that have been necessity of world before establishment of the Internet. Actually the Internet made the checking and reading the mail for the people so simple that every one has now become slave of Internet. As we know mail means exchanging of the data. As ordinary snail mail follows the rule of old mailing system to put some sort of the postage stamps so there are also some protocols, rules according to which data can be moved over the Internet, through which mail can be transferred.

Our project is about automatic control using the Internet technology. The project is based upon the client-server structure and the TCP/IP socket programming. The project describes the basic principles of client-server based communication systems. Details of TCP/IP protocol are then given with examples of its usage. The project then gives some examples of using the Visual Basic language to program client-server based systems. Principles of Internet based remote control are also described with examples.

The project is theoretical and there is no hardware involved. In a real Internet based control system we would also have an interface card in the client computer and this card will monitor and control the external hardware attached to it. The server computer should then exchange data with the client over the Internet, using the TCP/IP communications protocol.

Chapter 1

Introduction to Network & Client/Server

1.1 Networking

Consisting of two computers or more than two computers are connected to each other by cable so they can share data called Network.

Performing the job of establishing a network is called networking.

All networking, no matter how sophisticated, stems from that simple system. While the idea of two computers connected by cable may not seem extraordinary, in retrospect; it was a major achievement in communication.

Networking arose from the need to share data in a timely fashion. Personal computers are wonderful business tools for producing data, spreadsheets, graphics, and other types of information, but do not allow you to quickly share the data you have produced. Without a network, the documents have to print out so that other can edit them or use them. At best, you give files on floppy disks to others to merge the changes. This was, and still is, called working in a stand-alone environment.

1.2 Use of Network

Organizations implement networks primarily to share resources and enable online communication. Resources include data, applications, and peripherals. A peripheral is a device such as an external disk drive, printer, mouse, modem, or joystick. Online communication includes sending messages back and forth, or e-mail.

Computers that are part of network can share the following.

- Data
- Messages
- Graphics
- Printers
- Fax machines
- Modems
- Other hardware resources

This list is constantly growing as new ways are found to share and communicate by means of computers.

1.3 Data

Before networks existed, people who wanted to share information were limited to:

- Telling each other the information (voice communication).
- Writing memos.
- Putting the information on a floppy disk, physically taking the disk to another computer, and then copying the data onto that computer.
- Sending Data to remote access computer e.g. File transfer(FTP)

Networks can reduce the need for paper communication and make nearly any type of data available to every user who needs it.

1.4 Applications

Networks can be used to standardize applications, such as a word processor, to ensure that everyone on the network is using the same application and the same version of that application. Standardizing on one application can simplify support. It is easier to know one application very well than to try to learn four or five different applications. It is also easier to deal with only one version of an application and to set up all computers in the same manner.

Some businesses invest in networks because of e-mail and scheduling programs (different types of databases). Managers can use these utilities to communicate quickly and effectively with large numbers of people and to organize and schedule an entire company far more easily than was previously possible and people can easily get information about the desired job e.g. stock exchange... etc.

1.5 Network Topology

The term topology, or more specifically, network topology, refers to the arrangement or physical layout of computers, cables, and other components on the network. Simply the network topology is describes the way the computers are connected.

Basically there are three topologies of network.

1.5.1 Bus

This topology is uses for the broadcast channels; all computers share the single common channel. When one host sends data all others can receive while connected to the network and all the hosts, computers, connected to a single cable called *trunk* (back bone).

1.5.2 Star

Star Topology is used for point-to-point channels. Uses for the centralized system. If the central computer is down the network fails and main advantage of this is that we

can control the network centrally but the worst drawback is we might face the too much cables when expands the network.

1.5.3 Ring

This can be used for either broadcast or point-to-point channels. The data travels around the network in the ring. This has one advantage and one disadvantage. Advantage is this that uses fewer cables and if one host is down then network will be down sounds disadvantage but this uses widely in LANs.

1.6 Range of Networks

There are four types of the networks those can be established.

1.6.1 Local Area Networks (LAN)

The computers are connected in a building or over a small area; in general this network uses over small size. In this network maximum 30 users on a maximum cable length of just over 600 feet can be connected through any topology as above mentioned.

1.6.2 Wide Area Networks (WANs)

As the geographical scope of the network grows by connecting users in different cities or different states, the LAN grows into a wide area network (WAN). The number of users in a company network can now grow from ten to thousands.

Today, most major businesses store and share vast amounts of crucial data in a network environment, which is why networks are currently as essential to businesses as typewriters and filing cabinets used to be.

These type of networks are larger than the LANs. These spans a country or even the world e.g. Internet. Actually these are interconnected LANs. Transmitting each other while using the high-speed phone lines, microwave dishes ... etc.

1.6.3 Inter-Connected Networks (Inter Network)

It contains two or more LANs interconnected. Breaking down the large networks into smaller networks can make these types of networks.

1.6.4 Metropolitan Area Networks (MANs)

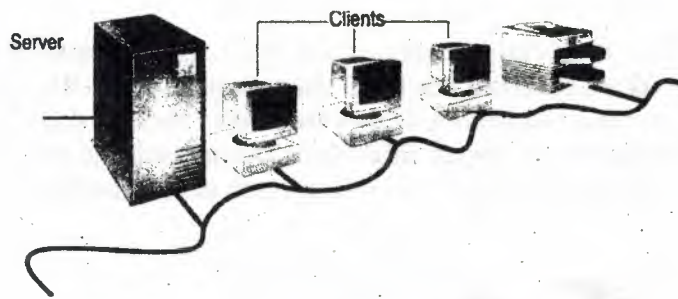
These are also interconnected LANs but larger than the inter networks. These can be within city or town e.g. large campus. And the feature of this network is usage of backbone, trunk, for interconnections.

1.7 Networking Overview

In general, all networks have certain components, functions, and features in common. These include:

- Servers—Computers that provide shared resources to network users.
- Clients—Computers that access shared network resources provided by a server.
- Media—The way those computers are connected.
- Shared data—Files provided by servers across the network.
- Shared printers and other peripherals. Other resources provided by servers.

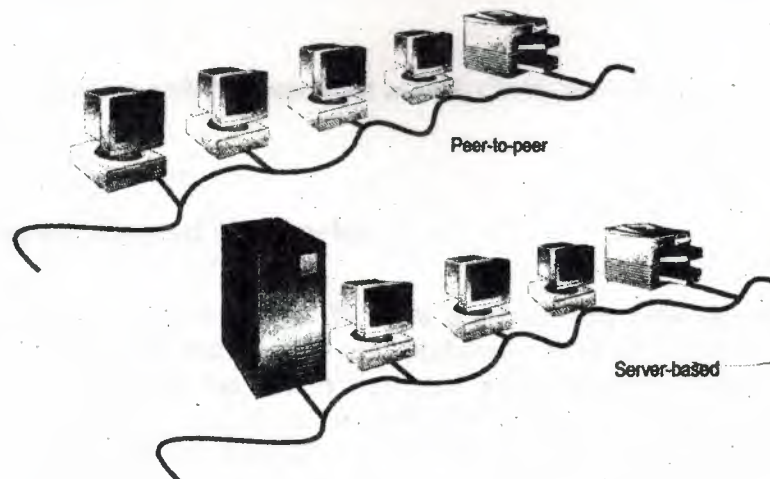
Resources—Files, printers, or other items to be used by network users.



Common network elements

Even with these similarities, networks can be divided into two broad categories:

- Peer-to-peer
- Server-based



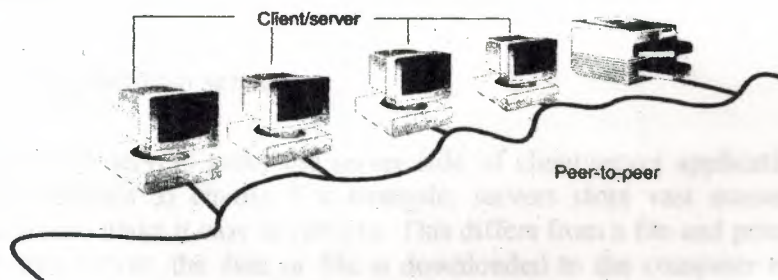
Typical peer-to-peer and server-based networks

The distinction between peer-to-peer and server-based networks is important because each has different capabilities. The type of network you implement will depend on numerous factors, including the:

- Size of the organization.
- Level of security required.
- Type of business.
- Level of administrative support available.
- Amount of network traffic.
- Needs of the network users.
- Network budget

1.7.1 Peer-to-Peer Networks

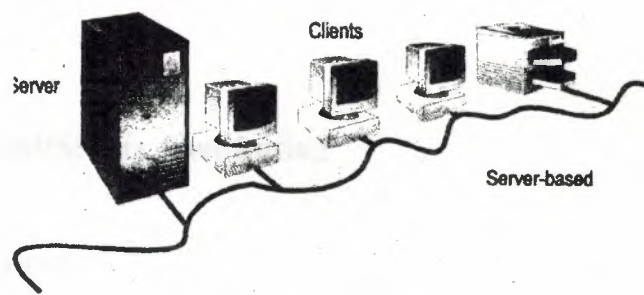
In a peer-to-peer network, there are no dedicated servers or hierarchy among the computers. All of the computers are equal and therefore are known as peers. Normally, each computer functions as both a client and a server, and there is no one assigned to be an administrator responsible for the entire network. The user at each computer determines what data on their computer gets shared on the network.



Peer-to-peer network computers act as both clients and servers
Size

1.7.2 Server-Based Networks

In an environment with more than 10 users, a peer-to-peer network—with computers acting as both servers and clients—will probably not be adequate. Therefore, most networks have dedicated servers. A dedicated server is one that only functions as a server and is not used as a client or workstation. Servers are "dedicated" because they are optimized to quickly service requests from network clients and to ensure the security of files and directories. Server-based networks have become the standard model for networking.



Server-based network

As networks increase in size and traffic, more than one server on the network is needed. Spreading the tasks among several servers ensures that each task will be performed in the most efficient manner possible.

1.7.3 Specialized Servers

The variety of tasks that servers must perform is varied and complex. Servers for large networks have become specialized to accommodate the expanding needs of users. For example, in a Windows NT Server network, the different types of servers include the following:

- **Application servers**

Application servers make the server side of client/server applications, as well as the data, available to clients. For example, servers store vast amounts of data that is structured to make it easy to retrieve. This differs from a file and print server. With a file and print server, the data or file is downloaded to the computer making the request. With an application server, the database stays on the server and only the result of a request is downloaded to the computer making the request.

A client application running locally would access the data on the application server. Instead of the entire database being downloaded from the server to your local computer, only the results of your query would be loaded onto your computer. For example, you could search the employee database for all employees who were born in November.

- **Mail servers**

Mail servers manage electronic messaging between network users.

1.8 Client/Server Computing

1.8.1 Client

The user generates a request at the front end. The client runs an application that:

- Presents an interface to the user.
- Formats requests for data.
- Displays data it receives back from the server.

In a client/server environment, the server does not contain the user interface software. The client is responsible for presenting the data in a useful form, such as with user interfaces and report writing.

The client computer accepts instructions from the user, prepares them for the server, and then sends a request for specific information over the network to the server. The server processes the request, locates the appropriate information, and sends it across the network back to the client. The client then feeds the information to the interface, which presents the information to the user.

In a client/server environment, the person at the client end uses an on-screen form; called a search key, to specify what information they are looking for.

1.8.3 Server

The server in a client/server environment is usually dedicated to storing and managing data. This is where most of the actual database activity occurs. The server is also referred to as the back end of the client/server model because it fulfills the requests of the client. The server receives the structured requests from the clients, processes them, and sends the requested information back over the network to the client.

The database software on the file server reacts to client queries by running searches. As part of a client/server system, it only returns the results of the searches.

Back-end processing includes sorting data, extracting the requested data, and sending that data back to the user.

Also, database server software manages the data in a database including:

- Updates
- Deletions
- Additions
- Protection

1.8.3 The Client/Server Process

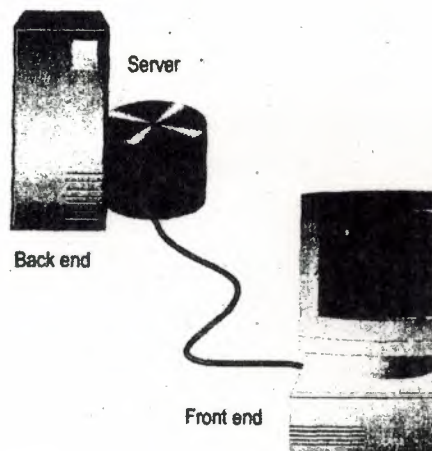
The database query is sent from the client but processed on the server. Only the results are sent across the network back to the client. The whole process of requesting and receiving information consists of six steps:

1. The client requests data.
2. The request is translated into SQL.
3. The SQL request is sent over the network to the server.
4. The database server carries out a search on the computer where the data exists.
5. The requested records are returned to the client.
6. The data is presented to the user.

In the client/server environment, there are two main components:

The application, which is often referred to as the client or the front end

The database server, which is often referred to as the server or the back end



The client is the front end and the server is the back end

1.9 The Advantages of Working in a Client/Server Environment

Client/server technology creates a powerful environment that. Offers many real benefits to organizations. Well-planned client/server systems provide relatively inexpensive platforms that provide mainframe-computing capacity while being easy to customize for specific applications. Because client/server processing only sends the results of a query across the network, it cuts down on network traffic.

It puts the file search burden on a computer that is more powerful than the client, and is better able to handle the request. On a busy network, this means that the processing will be distributed more evenly than in a traditional server-based system.

The client/server network also saves RAM in the client computer because all of the data and the file I/O logic is on the application in the server. The servers in client/server systems are capable of storing large amounts of data. This allows more space on client computers for other applications.

Because the file services and the data are on the back-end server, the servers are easier to secure and maintain in one location. Data is more secure in a client/server environment because it is centrally located on one server or on a small number of servers. When the data is in one location and managed by one authority, backups are simplified.

Chapter 2

Introduction to TCP/IP

2.1 Introduction of TCP/IP

TCP/IP stands for transport control protocol/Internet protocol. TCP/IP is actually a shorthand term for an entire family of protocols, all designed to transport information between both local area networks (LANs) and wide area networks (WANs).

Until a few years ago, it was nearly impossible for computer systems with different hardware or software to exchange data. TCP/IP was developed as a kind of "Esperanto" for computers — a universal language all systems understand. It has become the mainstay of the worldwide Internet, as well as a useful tool for smaller, proprietary intranets.

TCP/IP has become widely used because:

- Its open standards are freely available and independent of any specific vendor's hardware or operating system.
- It can run over Ethernet, ARCnet, Token-Ring, phone line, and X.25 networks and virtually any other transmission medium.
- It has a versatile addressing scheme that lets any TCP/IP device address any other device in a network — even a network of worldwide proportions.
- It can accept major upgrades and repairs while actively operating.

It can automatically route transmitted data around trouble sites. If a snowstorm blows your connection in Minneapolis, your messages are routed elsewhere and still arrive at their destination.

TCP and IP were developed by a Department of Defense (DOD) research project to connect a number different networks designed by different vendors into a network of networks (the "Internet"). It was initially successful because it delivered a few basic services that everyone needs (file transfer, electronic mail, remote logon) across a very large number of client and server systems. Several computers in a small department can

use TCP/IP (along with other protocols) on a single LAN. The IP component provides routing from the department to the enterprise network, then to regional networks, and finally to the global Internet. On the battlefield a communications network will sustain damage, so the DOD designed TCP/IP to be robust and automatically recover from any node or phone line failure. This design allows the construction of very large networks with less central management. However, because of the automatic recovery, network problems can go undiagnosed and uncorrected for long periods of time.

As with all other communications protocol, TCP/IP is composed of layers:

- **IP** - is responsible for moving packet of data from node to node. IP forwards each packet based on a four-byte destination address (the IP number). The Internet authorities assign ranges of numbers to different organizations. The organizations assign groups of their numbers to departments. IP operates on gateway machines that move data from department to organization to region and then around the world.
- **TCP** - is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate network. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received.
- **Sockets** - is a name given to the package of subroutines that provide access to TCP/IP on most systems.

An increasing number of people are using the Internet and, many for the first time, are using the tools and utilities that at one time were only available on a limited number of computer systems (and only for really intense users!). One sign of this growth in use has been the significant number of TCP/IP and Internet books, articles, courses, and even TV shows that have become available in the last several years;

2.2 The Evolution of TCP/IP & Internet

Prior to the 1960s, what little computer communication existed comprised simple text and binary data, carried by the most common telecommunications network technology of the day. Because most data traffic is burst in nature (i.e., most of the transmissions occur during a very short period of time), circuit switching results in highly inefficient

use of network resources. In 1962, Paul Baran, of the Rand Corporation, described a robust, efficient, store-and-forward data network in a report for the U.S. Air Force; Donald Davies suggested a similar idea in independent work for the Postal Service in the U.K., and coined the term packet for the data units that would be carried. According to Baran and Davies, packet switching networks could be designed so that all components operated independently, eliminating single point-of-failure problems. In addition, network communication resources appear to be dedicated to individual users but, in fact, statistical multiplexing and an upper limit on the size of a transmitted entity result in fast, economical data networks.

The modern Internet began as a U.S. Department of Defense (DoD) funded experiment to interconnect DoD-funded research sites in the U.S. In December 1968, the Advanced Research Projects Agency (ARPA) awarded a contract to design and deploy a packet switching network to Bolt Beranek and Newman (BBN). In September 1969, the first node of the ARPANET was installed at UCLA. With four nodes by the end of 1969, the ARPANET spanned the continental U.S. by 1971 and had connections to Europe by 1973.

The original ARPANET gave life to a number of protocols that were new to packet switching. One of the most lasting results of the ARPANET was the development of a user-network protocol that has become the standard interface between users and packet switched networks; namely, ITU-T (formerly CCITT) Recommendation X.25. This "standard" interface encouraged BBN to start Telenet, a commercial packet-switched data service, in 1974; after much renaming, Telenet is now a part of Sprint's X.25 service.

The initial host-to-host communications protocol introduced in the ARPANET was called the Network Control Protocol (NCP). Over time, however, NCP proved to be incapable of keeping up with the growing network traffic load. In 1974, a new, more robust suite of communications protocols was proposed and implemented throughout the ARPANET, based upon the Transmission Control Protocol (TCP) and Internet Protocol (IP). TCP and IP were originally envisioned functionally as a single protocol, thus the protocol suite, which actually refers to a large collection of protocols and applications, is usually referred to simply as TCP/IP. The original versions of both TCP and IP that are in common use today were written in September 1981, although both have had several modifications applied to them (in addition, the IP version 6, or IPv6, specification was released in December 1995). In 1983, the DoD mandated that all of their computer systems would use the TCP/IP protocol suite for long-haul communications, further enhancing the scope and importance of the ARPANET.

In 1983, the ARPANET was split into two components. One component, still called ARPANET, was used to interconnect research/development and academic sites; the other, called MILNET, was used to carry military traffic and became part of the Defense Data Network. That year also saw a huge boost in the popularity of TCP/IP with its inclusion in the communications kernel for the University of California's UNIX implementation, 4.2BSD (Berkeley Software Distribution) UNIX.

In 1986, the National Science Foundation (NSF) built a backbone network to interconnect four NSF-funded regional supercomputer centers and the National Center for Atmospheric Research (NCAR). This network, dubbed the NSFNET, was originally intended as a backbone for other networks, not as an interconnection mechanism for individual systems. Furthermore, the "Appropriate Use Policy" defined by the NSF limited traffic to non-commercial use. The NSFNET continued to grow and provide connectivity between both NSF-funded and non-NSF regional networks, eventually becoming the backbone that we know today as the Internet. Although early NSFNET applications were largely multiprotocol in nature, TCP/IP was employed for interconnectivity (with the ultimate goal of migration to Open Systems Interconnection).

In 1993, the NSF decided that it did not want to be in the business of running and funding networks, but wanted instead to go back to the funding of research in the areas of supercomputing and high-speed communications. In addition, there was increased pressure to commercialize the Internet; in 1989, a trial gateway connected MCI, CompuServe, and Internet mail services, and commercial users were now finding out about all of the capabilities of the Internet that once belonged exclusively to academic and hard-core users! In 1991, the Commercial Internet Exchange (CIX) Association was formed by General Atomics, Performance Systems International (PSI), and UNET Technologies to promote and provide a commercial Internet backbone service. Nevertheless, there remained intense pressure from non-NSF ISPs to open the network to all users.

In 1994, a plan was put in place to reduce the NSF's role in the public Internet. The new structure comprises three parts:

1. Network Access Points (NAPs), where individual ISPs would interconnect. Although the NSF is only funding four such NAPs (Chicago, New York, San Francisco, and Washington, D.C.), several non-NSF NAPs are also in operation.
2. The very High Speed Backbone Network Service, a network interconnecting the NAPs and NSF-funded centers, operated by MCI. This network was installed in 1995 and operated at OC-3 (155.52 Mbps); it was completely upgraded to OC-12 (622.08 Mbps) in 1997.
3. The Routing Arbiter, to ensure adequate routing protocols for the Internet.

In addition, NSF-funded ISPs were given five years of reduced funding to become commercially self-sufficient. This funding ended by 1998.

In 1988, meanwhile, the DoD and most of the U.S. Government chose to adopt OSI protocols. TCP/IP was now viewed as an interim, proprietary solution since it ran only on limited hardware platforms and OSI products were only a couple of years away. The

DoD mandated that all computer communications products would have to use OSI protocols by August 1990 and use of TCP/IP would be phased out. Subsequently, the U.S. Government OSI Profile (GOSIP) defined the set of protocols that would have to be supported by products sold to the federal government and TCP/IP was not included.

Despite this mandate, development of TCP/IP continued during the late 1980s as the Internet grew. TCP/IP development had always been carried out in an open environment (although the size of this open community was small due to the small number of ARPA/NSF sites), based upon the creed "We reject kings, presidents, and voting. We believe in rough consensus and running code" [Dave Clark, M.I.T.]. OSI products were still a couple of years away while TCP/IP became, in the minds of many, the real open systems interconnection protocol suite.

It is not the purpose of this memo to take a position in the OSI vs. TCP/IP debate. Nevertheless, a number of observations are in order. First, the ISO Development Environment (ISODE) was developed in 1990 to provide an approach for OSI migration for the DoD. ISODE software allows OSI applications to operate over TCP/IP. During this same period, the Internet and OSI communities started to work together to bring about the best of both worlds as many TCP and IP features started to migrate into OSI protocols, particularly the OSI Transport Protocol class 4 (TP4) and the Connectionless Network Layer Protocol (CLNP), respectively. Finally, a report from the National Institute for Standards and Technology (NIST) in 1994 suggested that GOSIP should incorporate TCP/IP and drop the "OSI-only" requirement. [NOTE: Some industry observers have pointed out that OSI represents the ultimate example of a sliding window; OSI protocols have been "two years away" since about 1986.]

2.3 Architectural Model

TCP/IP is most commonly associated with the Unix operating system. While developed separately, they have been historically tied, as mentioned above, since 4.2BSD Unix started bundling TCP/IP protocols with the operating system. Nevertheless, TCP/IP protocols are available for all widely used operating systems today and native TCP/IP support is provided in OS/2, OS/400, and Windows 95/98/NT, as well as most Unix variants.

The following figures show how the layers build upon each other and how the TCP/IP protocols described above fit.

Figure 2.1 shows the Windows NT model within the seven-layer OSI model, which has become a standard model for comparison of network protocols. Note that most actual networks do not conform precisely to the OSI model.

OSI Layer	Protocol			
Application	SMB	CSNW	GSNW	Others
Presentation				
TDI				
Session	TC/IP	NWLink	NetBEUI/NBF Logical Link Control	Others
Transport				
Network				
NDIS 3.0				
Data Link	Network Interface Card Driver(s)			
Physical	Network Interface Cards			

Figure 2.1 Networking Architectural Models

The sections below will provide a brief overview of each of the layers in the TCP/IP suite and the protocols that compose those layers. A large number of books and papers have been written that describe all aspects of TCP/IP as a protocol suite, including detailed information about use and implementation of the protocols

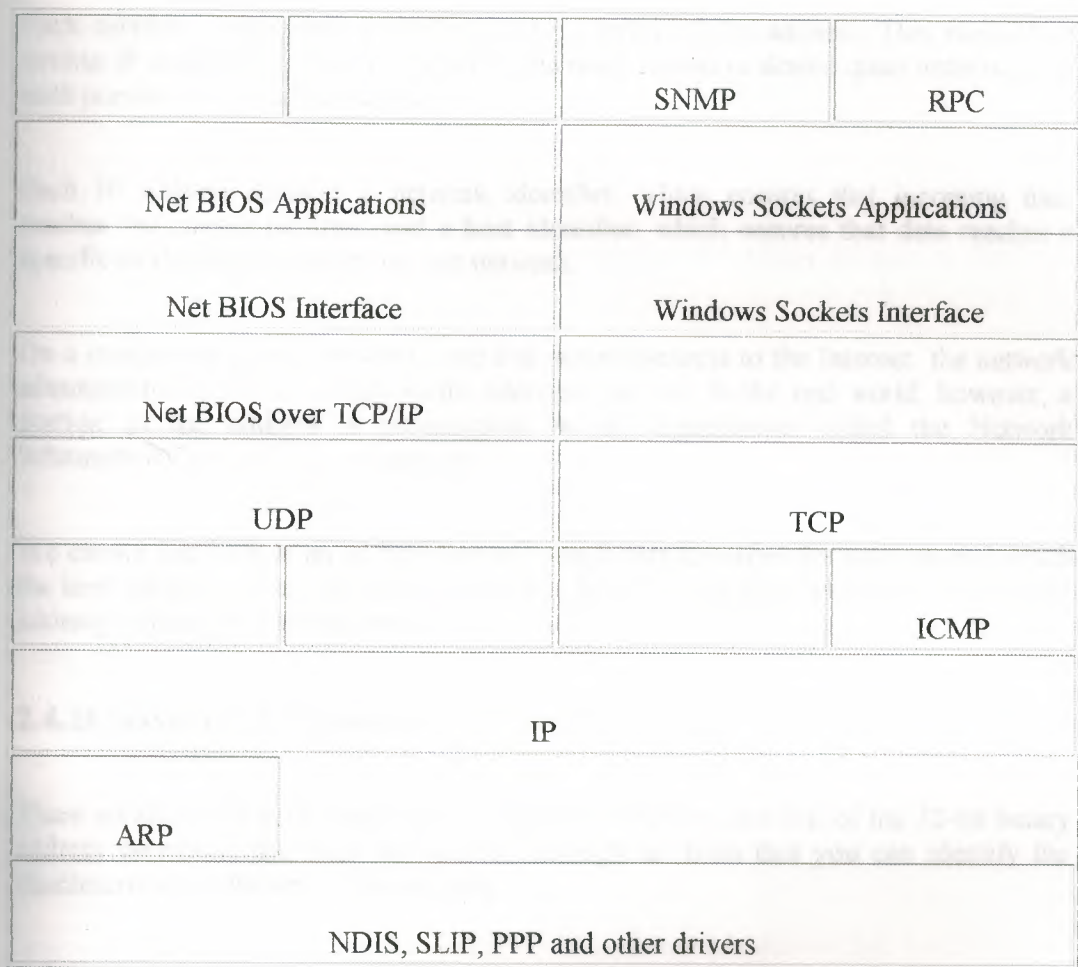


Figure 1.2 TCP/IP Architectural Models

2.4 IP Address

An IP address is a set of numbers that identifies a particular network and the individual workstations and servers on the network. Just as your street address determines where your mail goes, the IP address determines where a network delivers your data.

The address is a 32-bit value, such as 10011101 00011000 01110000 11010101. However, for ease of use, IP addresses are usually expressed as four numbers between 0 and 255, separated by dots. Here are two example IP addresses:

132.36.253.10

99.224.78.176

Each number corresponds to eight bits (one byte) of the address. This method of writing IP addresses is sometimes called the quad format or dotted quad notation, and each portion of it is called a quad.

Each IP address contains a network identifier, which ensures that incoming data reaches the correct network, and a host identifier, which ensures that data reaches a specific workstation or server on that network.

On a completely private network one that never connects to the Internet the network administrator is free to assign entire addresses at will. In the real world, however, a portion of the address is pre-assigned by an organization called the Network Information Center (NIC), or InterNIC.

We cannot just look at an address and tell which part identifies the network and which the host. Before you can do that, you need to be able to identify what class a particular address belongs to. For that, read on.

2.4.1 Classes of IP Addresses

There are three common classes of IP addresses. The first few bits of the 32-bit binary address determine the class the address belongs to; from that you can identify the characteristics of the rest of the address.

2.4.1.1 Class A

In a Class A network; the first bit of the binary IP address is always 0. The first byte (quad) is always between 1 and 126. The first byte of the number identifies the network. The final three bytes identify the hosts on the network.

There are 126 or fewer Class A networks in the world, but each can have up to 16,777,214 hosts on it. These addresses have been assigned by the InterNIC to very large organizations such as Hewlett Packard, whose network's first byte is 15.

2.4.1.2 Class B

In a Class B network, the first two bits of the binary IP address are always 10. The first byte (quad) is always between 128 and 191. The first two full bytes identify the network. The last two bytes identify the host.

Therefore, there are 16,384 Class B networks, and each of these can have up to 65,534 hosts. These networks were assigned to medium-sized companies and institutions.

2.4.1.3 Class C

In a Class C network, the first three bits of the binary IP address are always 110. The first byte (quad) is always between 192 and 223. The first three complete bytes identify the network. The last byte identifies the host.

Therefore, there can be 2,097,151 networks within Class C, but each network can contain only 254 or fewer hosts. These networks are the only ones still available and are being assigned to companies of all sizes. Microsoft, for instance, uses the Class C network 198, among others.

There are also several special, reserved classes of networks, but you do not need to worry about those now.

2.5 Diagnosing and Solving IP Configuration Problems

When diagnosing problems on TCP/IP, it is useful to remember that NetBIOS and TCP/IP is a layered environment, as indicated in the architectural models in Module 1. You can isolate a specific problem by checking various layers. The fastest way to do this is to split the architectural model in half, right at the IP layer.

Your best friend in troubleshooting TCP/IP networks is the utility program ping. You can use ping to tell you whether a problem is at the IP layer, below it, or above. First, use ping to test the IP layer of the machine you are on. Ping the loopback device. The loopback device is simply the IP address 127.0.0.1, which is designed to send any packet it gets straight back at the sender. (Actually, the entire network 127 is reserved for a loopback device, but 127.0.0.1 is the traditional address to use. The name associated with this address is "localhost.")

If you type this command line:

```
C:\users\default>ping 127.0.0.1
```

You should get output similar to this:

Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time<10ms TTL=32

Reply from 127.0.0.1: bytes=32 time<10ms TTL=32

Reply from 127.0.0.1: bytes=32 time<10ms TTL=32

Reply from 127.0.0.1: bytes=32 time<10ms TTL=32

If all is working, ping repeats the same information several times, then stops. Clearly, the computer above has a working IP layer.

If ping doesn't work, check to be sure TCP/IP is installed on the computer you're working on and that the computer rebooted after installation.

Check NT's Event Viewer for problems reported by setup or by TCP/IP.

Now, learn the IP address of the computer you're working on. Use ipconfig by typing the following command line:

C:\users\default>ipconfig

The system should return output similar to

Windows NT IP Configuration

Ethernet adapter AT17001:

IP Address : 192.0.0.4

To v Subnet Mask : 255.255.255.0

The e Default Gateway : 192.0.0.4

If the IP address or subnet mask on your planning sheet disagrees with that returned by ipconfig, fix the discrepancy, and see if the problem continues.

Now ping the computer's IP address by typing

```
G:\users\default>ping 192.0.0.4
```

You should get the following result:

Pinging 192.0.0.4 with 32 bytes of data:

Reply from 192.0.0.4: bytes=32 time<10ms TTL=32

Reply from 192.0.0.4: bytes=32 time<10ms TTL=32

Reply from 192.0.0.4: bytes=32 time<10ms TTL=32

Reply from 192.0.0.4: bytes=32 time<10ms TTL=32

If you don't get this result, something is wrong on the computer. Go over the above steps and double-check your work.

Up to now, you have been pinging your own computer. Now try pinging another host on the same physical network. A computer you know to be good, such as a server or the gateway, is an appropriate target.

If you don't know whether a particular address is actually on your network, use the `tracert` utility, which traces the route to another computer. As a byproduct of tracing the route, it tells you how many "hops" it has to make to reach its destination. If it only has to make one hop, you know the destination is on the same network.

To use the `tracert` command, type `c:\users\default>tracert [address]`

The output will look something like this:

```
C:\users\default>tracert 192.0.0.2
```


Tracing route to bad dog [192.0.0.2]

Over a maximum of 30 hops:

```
1  10 ms <10 ms <10 ms  bad dog [192.0.0.2]
```

Trace complete.

The "1" tells you it took a single hop to reach this machine and therefore that the two computers are on the same network.

Now, having traced the route to the computer, ping the computer's IP address. If that works, ping its host name. If that doesn't work, then you know you have a name resolution problem, which will be covered later in this module.

If you can ping yourself but not another host on the local net, then you may have a binding problem. Use `netstat -e` to get some Ethernet level statistics. You can use `nbtstat` to learn what a host knows about its local network. `Nbtstat -a` or `nbtstat -A` will tell you what a remote host knows about its environment. If these utilities return results that look as if there is no connection to the net, then check the MAC address (which you can learn with `nbtstat -a`) against the Ethernet card. They should agree.

You can also use ping on multiple hosts to isolate a problem. For example, if A can ping B but B cannot ping A, then most likely B has a configuration problem, probably at the IP layer. Check B's IP address and subnet mask.

You can also use the `arp -a` command to check arp resolution. Make sure the IP address and MAC address for each entry agree with what you have in your notebook of planning sheets. If they don't, you may have a mis-assigned IP address.

You also could simply have a bad connection, such as a loose connector. The best way to check for bad connections is to use Ethernet cards with "link" LEDs. If you have a good connection, the link LED will be lit. If it is off, you have a physical layer problem. Many Ethernet cards have other indicator LEDs as well.

Chapter 3

Windows Socket (Winsock)

3.1 Background of Winsock

The Windows Sockets specification defines a binary-compatible network-programming interface for Microsoft Windows. Windows Sockets are based on the UNIX® sockets implementation in the Berkeley Software Distribution (BSD, release 4.3) from the University of California at Berkeley. The specification includes both BSD-style socket routines and extensions specific to Windows. Using Windows Sockets permits your application to communicate across any network that conforms to the Windows Sockets API. On Win32, Windows Sockets provide for thread safety.

Many network software vendors support Windows Sockets under network protocols including Transmission Control Protocol/Internet Protocol (TCP/IP), Xerox® Network System (XNS), Digital Equipment Corporation's DECNet™ protocol, Novell® Corporation's Internet Packet Exchange/Sequenced Packed Exchange (IPX/SPX), and others. Although the present Windows Sockets specification defines the sockets abstraction for TCP/IP, any network protocol can comply with Windows Sockets by supplying its own version of the dynamic link library (DLL) that implements Windows Sockets. Examples of commercial applications written with Windows Sockets include X Window servers, terminal emulators, and electronic mail systems.

Note: Keep in mind that the purpose of Windows Sockets is to abstract away the underlying network so you don't have to be knowledgeable about that network and so your application can run on any network that supports sockets. Consequently, this documentation doesn't discuss the details of network protocols.

The Microsoft Foundation Class Library (MFC) supports programming with the Windows Sockets API by supplying two classes. One of these classes, CSocket, provides a high level of abstraction to simplify your network communications programming.

The Windows Sockets specification, Windows Sockets: An Open Interface for Network Computing Under Microsoft Windows, now at version 1.1, was developed as an open networking standard by a large group of individuals and corporations in the TCP/IP community and is freely available for use. The sockets programming model supports one "communication domain" currently, using the Internet Protocol Suite. The specification is available in the Win32 SDK.

Tip: Because sockets use the Internet Protocol Suite, they are the preferred route for applications that support Internet communications on the “information highway.”

3.2 Definition of a Socket

A socket is a communication endpoint an object through which a Windows Sockets application sends or receives packets of data across a network. A socket has a type and is associated with a running process, and it may have a name. Currently, sockets generally exchange data only with other sockets in the same “communication domain,” which uses the Internet Protocol Suite.

Both kinds of sockets are bi-directional: they are data flows that can be communicated in both directions simultaneously (full-duplex).

Two socket types are available:

- Stream sockets

Stream sockets provide for a data flow without record boundaries a stream of bytes. Streams are guaranteed to be delivered and to be correctly sequenced and unduplicated.

- Datagram sockets

Datagram sockets support a record-oriented data flow that is not guaranteed to be delivered and may not be sequenced as sent or unduplicated.

“Sequenced” means that packets are delivered in the order sent. “Unduplicated” means that you get a particular packet only once.

Note: Under some network protocols, such as XNS, streams can be record-oriented streams of records rather than streams of bytes. Under the more common TCP/IP protocol, however, streams are byte streams. Windows Sockets provides a level of abstraction independent of the underlying protocol.

3.3 The Socket Data Type

Each MFC socket object encapsulates a handle to a Windows Sockets object. The data type of this handle is SOCKET. A SOCKET handle is analogous to the HWND for a window. MFC socket classes provide operations on the encapsulated handle.

The SOCKET data type is described in detail in the Win32 SDK. See the topic Socket Data Type and Error Values under Windows Sockets.

3.3.1 Uses for Sockets

Sockets are highly useful in at least three communications contexts:

- Client/Server models
- Peer-to-peer scenarios, such as chat applications
- Making remote procedure calls (RPC) by having the receiving application interpret a message as a function call

Tip The ideal case for using MFC sockets is when we are writing both ends of the communication: using MFC at both ends. For more information on this topic, including how to manage the case when you're communicating with non-MFC applications.

3.3.2 Byte Ordering

Different machine architectures sometimes store data using different byte orders. For example, Intel-based machines store data in the reverse order of Macintosh (Motorola) machines. Intel's byte order, called "little-Endian," is also the reverse of the network standard "big-Endian" order. The following table explains these terms.

Table 3.1 Big- and Little-Endian Byte Ordering

Byte ordering	Meaning
Big-Endian	The most significant byte is on the left end of a word.
Little-Endian	The most significant byte is on the right end of a word.

Typically, you don't have to worry about byte-order conversion for data that you send and receive over the network, but there are situations in which you must convert byte orders.

3.3.3 Convert Byte Orders

We need to convert byte orders in the following situations:

- You're passing information that needs to be interpreted by the network, as opposed to the data you're sending to another machine. For example, you might pass ports and addresses, which the network must understand.
- The server application with which you're communicating is not an MFC application (and you don't have source code for it). This calls for byte order conversions if the two machines don't share the same byte ordering.

3.3.4 When We Don't Have to Convert Byte Orders

You can avoid the work of converting byte orders in the following situations:

- The machines on both ends can agree not to swap bytes, and both machines use the same byte order.
- The server you're communicating with is an MFC application.
- You have source code for the server you're communicating with, so you can tell explicitly whether you must convert byte orders or not.
- You can port the server to MFC.

Working with CAsyncSocket, you must manage any necessary byte-order conversions yourself. Windows Sockets standardizes the "big-Endian" byte-order model and provides functions to convert between this order and others. CArchive, however, which you use with CSocket, uses the opposite ("little-Endian") order but CArchive takes care of the details of byte-order conversions for you. By using this standard ordering in your applications, or using Windows Sockets byte-order conversion functions, you can make your code more portable.

The ideal case for using MFC sockets is when you're writing both ends of the communication: using MFC at both ends. If you're writing an application that will communicate with non-MFC applications, such as an FTP server, you'll probably need to manage byte-swapping yourself before you pass data to the archive object, using the Windows Sockets conversion routines, ntohs, ntohl, htons, and htonl. An example of these functions used in communicating with a non-MFC application appears later in this article.

3.4 Sequence of Operations for a Stream Socket Communication

Up to the point of constructing a CSocketFile object, the following sequence is accurate (with a few parameter differences) for both CAsyncSocket and CSocket. From that point on, the sequence is strictly for CSocket. The following table illustrates the sequence of operations for setting up communication between a client and a server.

Setting Up Communication Between a Server and a Client

Server	Client
// construct a socket CSocket sockSrvr;	// construct a socket CSocket sockClient;
// create the SOCKET sockSrvr.Create(nPort); ^{1,2}	// create the SOCKET sockClient.Create(); ²
// start listening sockSrvr.Listen();	
	// Seek a connection sockClient.Connect(strAddr, nPort); ^{3,4}
// construct a new, empty socket CSocket sockRecv; // accept connection sockSrvr.Accept(sockRecv); ⁵	
// construct file object CSocketFile file(&sockRecv);	// construct file object CSocketFile file(&sockClient);
// construct an archive CArchive arIn(&file, CArchive::load); -or- CArchive arOut(&file, CArchive::store); - or Both -	// construct an archive CArchive arIn(&file, CArchive::load); -or- CArchive arOut(&file, CArchive::store); - or Both -
// use the archive to pass data: arIn >> dwValue; -or- arOut << dwValue; ⁶	// use the archive to pass data: arIn >> dwValue; -or- arOut << dwValue; ⁶

Table 3.2

1. Where *nPort* is a port number
2. The server must always specify a port so clients can connect. The Create call sometimes also specifies an address. On the client side, use the default parameters, which ask MFC to use any available port.
3. Where *nPort* is a port number and *strAddr* is a machine address or an Internet Protocol (IP) address.
4. Machine addresses can take several forms: "ftp.microsoft.com", "ucsd.edu". IP addresses use the "dotted number" form "127.54.67.32". The Connect function checks to see if the address is a dotted number (although it doesn't check to ensure the number is a valid machine on the network). If not, Connect assumes a machine name of one of the other forms.
5. When you call Accept on the server side, you pass a reference to a new socket object. You must construct this object first, but do not call Create for it. Keep in mind that if this socket object goes out of scope, the connection closes. MFC connects the new object to a SOCKET handle. You can construct the socket on the stack, as shown, or on the heap.
6. The archive and the socket file are closed when they go out of scope. The socket object's destructor also calls the Close member function for the socket object when the object goes out of scope or is deleted.

3.5 Winsock Control

A WinSock control allows you to connect to a remote machine and exchange data using either the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP). Both protocols can be used to create client and server applications. Like the Timer control, the WinSock control doesn't have a visible interface at run time.

The Winsock control, invisible to the user, provides easy access to TCP and UDP network services. Microsoft Access, Visual Basic, Visual C++, or Visual FoxPro developers can use it. To write client or server applications we need to understand the details of TCP or to call low-level Winsock APIs. By setting properties and invoking methods of the control, you can easily connect to a remote machine and exchange data in both directions.

3.5.1 Possible Uses

- Create a client application that collects user information before sending it to a central server.
- Create a server application that function as a central collection point for data from several users.
- Create a "chat" application.

3.5.2 Selecting a Protocol

When using the WinSock control, the first consideration is whether to use the TCP or the UDP protocol. The major difference between the two lies in their connection state:

- The TCP protocol control is a connection-based protocol, and is analogous to a telephone — the user must establish a connection before proceeding.
- The UDP protocol is a connectionless protocol, and the transaction between two computers is like passing a note: a message is sent from one computer to another, but there is no explicit connection between the two. Additionally, the network determines the maximum data size of individual sends.

The nature of the application you are creating will generally determine which protocol you select. Here are a few questions that may help you select the appropriate protocol:

1. Will the application require acknowledgment from the server or client when data is sent or received? If so, the TCP protocol requires an explicit connection before sending or receiving data.
2. Will the data be extremely large (such as image or sound files)? Once a connection has been made, the TCP protocol maintains the connection and ensures the integrity of the data. This connection, however, uses more computing resources, making it more "expensive."
3. Will the data be sent intermittently, or in one session? For example, if you are creating an application that notifies specific computers when certain tasks have completed, the UDP protocol may be more appropriate. The UDP protocol is also more suited for sending small amounts of data.

3.5.3 Setting the Protocol

To set the protocol that your application will use: at design-time, on the Properties window, click Protocol and select either `sckTCPProtocol`, or `sckUDPProtocol`. We can also set the Protocol property in code, as shown below:

```
Winsock1.Protocol = sckTCPProtocol
```

3.5.4 Determining the Name of Computer

To connect to a remote computer, you must know either its IP address or its "friendly name." The IP address is a series of three digit numbers separated by periods (xxx.xxx.xxx.xxx). In general, it's much easier to remember the friendly name of a computer.

3.5.5 To find the computer's name

1. On the **Taskbar** of your computer, click **Start**.
2. On the **Settings** item, click the **Control Panel**.
3. Double-click the **Network** icon.
4. Click the **Identification** tab.
5. The name of your computer will be found in the **Computer name** box.

Once you have found your computer's name, it can be used as a value for the RemoteHost property.

3.6 TCP Connection Basics

The Transfer Control Protocol allows you to create and maintain a connection to a remote computer. Using the connection, both computers can stream data between themselves.

If you are creating a client application, you must know the server computer's name or IP address (Remote Host property), as well as the port (Remote Port property) on which it will be "listening." Then invoke the Connect method.

If you are creating a server application, set a port (Local Port property) on which to listen, and invoke the Listen method. When the client computer requests a connection, the Connection Request event will occur. To complete the connection, invoke the Accept method within the Connection Request event.

Once a connection has been made, either computer can send and receive data. To send data, invoke the Send Data method. Whenever data is received, the Data Arrival event occurs. Invoke the Get Data method within the Data Arrival event to retrieve the data

The following steps create a rudimentary server:

3.6.1 To create a TCP server

1. Create a new Standard EXE project.
2. Change the name of the default form to frmServer.
3. Change the caption of the form to "TCP Server."
4. Draw a Winsock control on the form and change its name to tcpServer.

5. Add two TextBox controls to the form. Name the first txtSendData, and the second txtOutput.
6. Add the code

3.6.2 To create a TCP Client

1. Create a new Standard EXE project.
2. Change the name of the default form to frmClient.
3. Change the caption of the form to "TCP Client."
4. Draw a Winsock control on the form and change its name to tcpClient.
5. Add two TextBox controls to the form. Name the first txtSendData, and the second txtOutput.
6. Add the code

3.6.3 Accepting More than One Connection Request

The basic server outlined above accepts only one connection request. However, it is possible to accept several connection requests using the same control by creating a control array. In that case, you do not need to close the connection, but simply create a new instance of the control (by setting its Index property), and invoking the Accept method on the new instance.

The code below assumes there is a Winsock control on a form named sockServer, and that its Index property has been set to 0; thus the control is part of a control array. In the Declarations section, a module-level variable intMax is declared. In the form's Load event, intMax is set to 0, and the LocalPort property for the first control in the array is set to 1001. Then the Listen method is invoked on the control, making it the "listening control. As each connection request arrives, the code tests to see if the Index is 0 (the value of the "listening" control). If so, the listening controls increments intMax, and uses that number to create a new control instance. The new control instance is then used to accept the connection request.

3.7 UDP Connection Base Basics

The User Datagram Protocol (UDP) is a connectionless protocol. Unlike TCP operations, computers do not establish a connection. Also, a UDP application can be either a client or a server.

To transmit data, first set the client computer's Local Port property. The server computer then needs only to set the Remote Host to the Internet address of the client computer, and the Remote Port property to the same port as the client computer's Local Port property, and invoke the Send Data method to begin sending messages. The client computer then uses the Get Data method within the Data Arrival event to retrieve the sent messages.

3.8 Local Port Property

Returns or sets the local port to use. Read/Write and available at design time.

- For the client, this designates the local port to send data from. Specify port 0 if the application does not need a specific port. In this case, the control will select a random port. After a connection is established, this is the local port used for the TCP connection.
- For the server, this is the local port to listen on. If port 0 is specified, a random port is used. After invoking the Listen method, the property contains the actual port that has been selected.

Chapter 4

TCP/IP Programming Using Visual Basic

4.1 TCP/IP PROGRAMMING

TCP/IP is currently the accepted standard protocol used when two or more computers wish to exchange information over a network. Several hardware and software components must be in place before the data can actually be sent and received.

First of all, the physical hardware connection must exist between the computers. This is either a network interface card (NIC), or a serial communications port for dial-up type networking connections.

Computers also need to be software configured to use the TCP/IP protocol over the selected transmission medium. As part of this configuration, each computer on a network should be given a unique Internet Protocol (IP) address and an IP name. Each computer is assigned a 32-bit number, which can be used to identify it over the network. This address is broken down into four 8-bit numbers, separated by dots. This is called *dot-notation* and looks something like "170.45.23.1" (see Chapter 3).

When a system sends data over the network using the TCP/IP protocol, it is sent in discrete units called *datagrams*, also known as packets. A datagram consists of a header followed by application dependent data. TCP/IP offers a reliable, full-duplex byte streaming type data communication. Data packets are re-transmitted if they do not reach their destinations reliably. TCP/IP is also connection-orientated protocol. This means that before two computers can begin to exchange data they must establish a connection with each other. One computer usually assumes the role of a server while the other computer the client. The server normally waits for a connection. The client is responsible for establishing the connection, while the server has the option of accepting (or rejecting) a connection request. If the connection is accepted then the two computers can begin to exchange data with each other.

There are two general approaches that you can take when creating TCP/IP based programs. One is to code at the lower level using the Windows Application Programming Interface (API) calls. The other option, which provides a higher-level and much simpler approach, is to use an *activex* component. The *Winsock* *activex* component provided with the Visual Basic distribution kit is a very powerful tool and enables you to develop TCP/IP based application in a very short time.

In this chapter you'll learn about:

- Windows sockets
- Winsock *activex* component
- Client-server based TCP/IP communications

Several working examples are given in this chapter on the programming of TCP/IP, using the Winsock *activex* component.

4.2 SOCKETS

A socket is basically a logical interface between two computers, which is created when using the TCP/IP protocol. A socket is a communications end-point. However, just creating a socket by itself is not enough for exchanging information. You have to assign a port number to direct the data to a specific place. A port is a virtual link between two systems, which communicate with each other. Several applications can use a socket, each transmitting and receiving data through a selected port. It is important that both sides of the communication ends must use the same port number. The port numbers are divided into three ranges: the Well Known Ports, the Registered Ports, and the Dynamic or Private Ports. The Well Known Ports are those from 0 to 1023. The registered Ports are those from 1024 to 49151, and the Dynamic Ports are those from 49152 to 65535. The Internet Assigned Numbers Authority (IANA) assigns the Well Known Ports and these should be used with care when the system is connected to the Internet. When writing TCP/IP based applications using the Winsock control, you have to make sure to stay away from any ports that are used by other applications.

Port	Service	Description
1	Tcpmux	TCP Port Service multiplexer
7	Echo	Echo
11	Systat	Systat
13	Daytime	Daytime
15	Netstat	Netstat
17	Qotd	Qotd
18	Msp	Message send protocol
19	Chargen	Ttytst source
20	Ftp	Ftp default data
21	Ftp	Control
22	Ssh	SSH remote login protocol
23	Telnet	Telnet
37	Time	Time
42	Nameserver	Host name server
43	Whois	Who is
53	Domain	Domain name server
57	Bootps	Bootp server
58	Bootpc	Bootp client
70	Gopher	Internet gopher
79	Finger	Finger
80	Http	World wide web http

10	PoP3	Post office protocol 3
23	Ntp	Network time service
61	Snmp	Snmp
62	Snmp	Snmp trap

Table 4.1 Some of the important Well-known Ports

4.3 WINSOCK CONTROL

Before we actually begin programming in TCP/IP, let's take a look at the Winsock activex control, which shall form the core of our programs. We shall be using this control nearly in all of our programs.

Just like the other standard activex components (e.g. dialog boxes, push-buttons, text-boxes, list-boxes etc.), Winsock activex control has a number of *properties*, *methods*, and *events* as described below.

4.3.1 Properties

The Winsock's most interesting properties are listed in Table 4.2 and are described here briefly. In all the examples given below, the Winsock control is named as Winsock.

Property	Description
Local Hostname	IP name for local machine
Local IP	IP address of local machine
Local Port	Local port number to use for data exchange
Protocol	Protocol to use for transmission
Remote Host	Remote machine IP name or IP address
State	Current state of Winsock control

Table 4.2 Winsock control important properties

LocalHostName returns a string, which is filled with the IP name of the local machine.

Example use:

```
Dim myname as string
```



```
myname = winsock1.LocalHostName
```

Assuming the local IP name is Charlie, variable my name will be assigned to string Charlie.

LocalIP returns a string, which is filled with the IP address of local machine in dotted string format

Example use:

```
Dim myaddress as string  
My address = winsock1.localIP
```

Assuming the local IP address is "125.23.45", variable my address will be assigned to string "125.23.45.2".

Protocol is used to set the protocol type used. Valid options are `sckTCP` protocol for TCP based protocols and `sckUDPProtocol` for UDP based protocols.

Example use:

```
Winsock1.Protocol = sckTCPProtocol
```

This code will set the protocol to TCP which is what we will be using in this chapter.

RemoteHost Name is used to set or return the remote machine to which a control sends or receives data. Either the IP name or the IP address of remote machine can be specified.

Example use:

```
winsock1.RemoteHost = "Charile"
```

This code will define the remote machine IP name as "Charile"

State property returns the state of the control, expressed as an enumerated type. The setting for the state property are listed in Table 4.3

Satus	Description
SckClosed	Closed (default)
SckOpen	Open
SckListening	Listening
SckCinnectionPending	Connection Pending
SckResolvingHost	Resolving Host
SckHostResolved	Host Resolved
SckConnecting	Connecting
SckConnected	Connected
SckClossing	Peer is Closing the Connection
SckError	Error

Table 4.3 Winsock State Property constant

Example use:

```

If Winsock1.state = sckOpen Then
    MsgBox "Socket is Open "
End if

```

This code will display the message "Scket is Open " if the socket is open.

4.3.2 Methods

The Winsock's most interesting methods are listed in Table 5.4 and are described here briefly. In all the examples below, the Winsock control is named as Winsock1.

Method	Description
Accept	Accept incoming connection (TCP/IP only)
Bind	Specify local port and IP in multiple connection
Close	Close connection (TCP only)
Listen	Listen for the connection (TCP only)
Get Data	Retrieve received data packet
Send Data	Send data packet

Table 4.4 Winsock Control Important Methods

Accept is used for TCP server applications only. This method is used to accept an incoming connection in a ConnectionRequest event (see section on events).

Example use:

```
Private Sub Winsock1_ConnectionRequest (Byval RequestID as Long)
    '
    'Accept the connection
    Winsock1.Accept requested
EndSub
```

The RequestID parameter identifies the request. The Accept method should be used on a new control instance and not on the one that is in the listening state.

Bind specifies the LocalPort and LocalIP address to be used for TCP connections. This method is used if there are multiple protocol adapters.

Example use:

```
Winsck1.Bind 500,"120.12.34.45"
```

Binds port 500 to local address "120.12.34.45". Bind is used before the Listen

Constant	Description
VbByte	Byte
VbInteger	Integer
VbLong	Long
VbSingle	Single
VbDouble	Double
VbCurrency	Currency
VbDate	Date
VbBoolean	Boolean
VbError	SCODE
VbArray + VbByte	Byte array

Table 4.5 Data types that can be used with GetData

Send Data sends data to a remote computer.

Example use:

```
Dim mydata as string
Mydata="This is a test string"
Winsck1.SendData mydata
```

Will send the contents of string mydata to the remote computer

4.3.3 Events

The Winsock's most interesting events are listed in Table 5.6 and are described here briefly. In all the examples below, the winsock control is named as winsck1.

method.

Close method closes a TCP connection.

Example use:

```
Winsckl.close
```

Listen is used by the server TCP protocols. The machine waits in listen mode waiting for a connection request from a client.

Example use:

```
Winsckl.Listen
```

GetData retrieves the received block of data and stores it in a variable. The general format is:

```
Object.GetData data,[type,][maxLen]
```

Where,

Data is where the retrieved data will be stored

Type is the type of data to be retrieved. This parameter is optional (see table 4.5 for a list of data types)

MaxLen is the size to read. This parameter is optional and if omitted, all Available data will be retrieved.

Example use:

```
Dim mydata as string  
Winsckl.GetData mydata
```

Will retrieve all available data and store in string mydata

Methods	Description
Close	Connection is Closed
Connect	A connection is established
Connection Request	A connection request is received
Error	An error has occurred
Data Arrival	Data is being received from remote computer

Table 4.6 Winsock Control important events

Close event occurs when the remote computer closes the connection. This event should be used to close a TCP connection properly.

Connect event occurs when a connection is established to the remote computer.

ConnectionRequest event occurs when a remote computer requests a connection. This event is available for TCP based server applications. The server should accept the connection request within this event procedure.

Error event occurs whenever an error occurs in background processing. For example, if failure is detected in receiving or sending data.

DataArrival event occurs when new data is received from the remote computer. GetData method should be used within this event procedure to retrieve the received data.

4.4 Client-Server Applications

Programs written to use the TCP/IP protocol are developed using the client-server model. In this model one computer assumes the role of the client, while the other computer must assume the role of the server. The server computer creates a socket and listens for connection requests from the client computer on a specified port. The client computer creates a socket and initiates a connection request to the server on the same port. The server computer then accepts (or rejects) the connection request and the two sides can start to exchange data. By accepting the connection, the server completes the virtual circuit between the two machines. It is important to note that a new socket is created by the act of accepting a connection and the original socket still continues to listen for additional connection requests. When the server computer no longer wishes to communicate, it closes the socket connection and this act breaks the virtual connection between the two machines.

The steps in establishing a connection, sending and receiving data, and closing a connection are shown diagrammatically in Figure 4.1

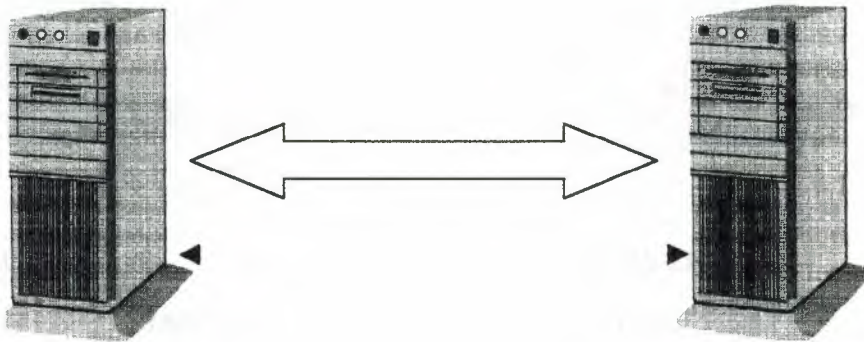


Figure 4.1 TCP/IP communication steps

CLIENT

Create a socket
Specify IP name or IP address of Server
Establish Port Number of Server
Establish connection with the Server
Send and received data
Close the Socket

SERVER

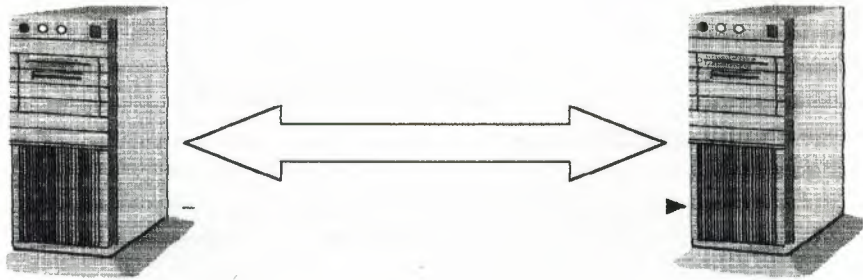
Create a socket
Listen for Connection
Accept the connection
Send and received data
Close the Socket

4.5 A SIMPLE TCP/IP EXAMPLE

4.5.1 One Way Communication

In this section we shall be looking at a simple client-server based TCP/IP example. In this example, assume that two computers with the network names and addresses as shown in Figure 4.2 are connected to each other using a network interface card. Two programs shall be developed, one for the server (SIERRA) and one for the client (CHARLIE). The server computer shall create a socket and listen for a connection. The client computer shall initiate a connection and then connect to the server. Data will then be sent from the

client to the server. The server computer shall display the received data in a text-box.



CLIENT

IP name: CHARILE

IP address: 170.100.16.1

SERVER

IP name: SIERRA

IP address: 170.100.16.2

Figure 4.2 Client and server computer of the simple example

The steps in creating the two programs are given below. Note that although two computers are used in this example it is possible to develop both the client and the server programs on the same computer by invoking two Visual Basic sessions. This should enable you to test the programs on one computer only without requiring a second one. The TCP/IP protocol should of course be installed and configured on this computer.

4.5.1.1 Client Computer

1. Load up Visual Basic and create a new project.
2. Go to Projects, then Components and select *Microsoft Winsock Control 6.0* by placing a tick in the box on the left hand side, as shown in Figure 4.3.

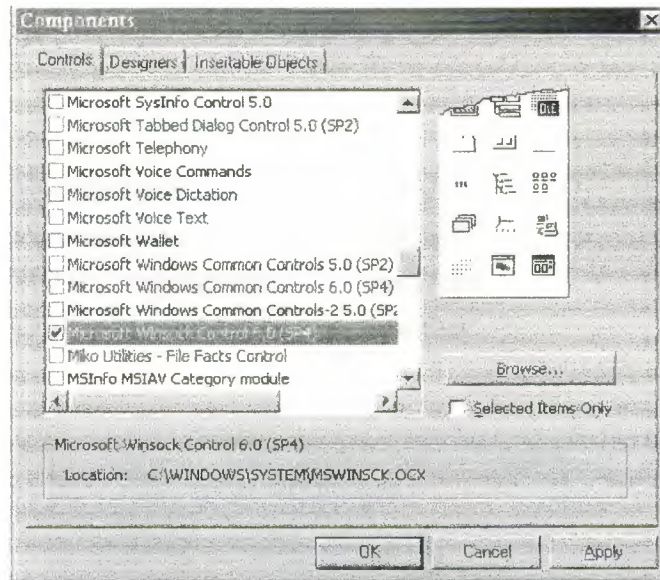


Figure 4.3 Selecting the Winsock control

3. Place the control on your form. Note that the form is not visible during run-time.
4. Click on the Winsock control. You should see the properties window as in Figure 4.4 Note that the name of the control is set to Winsock 1.

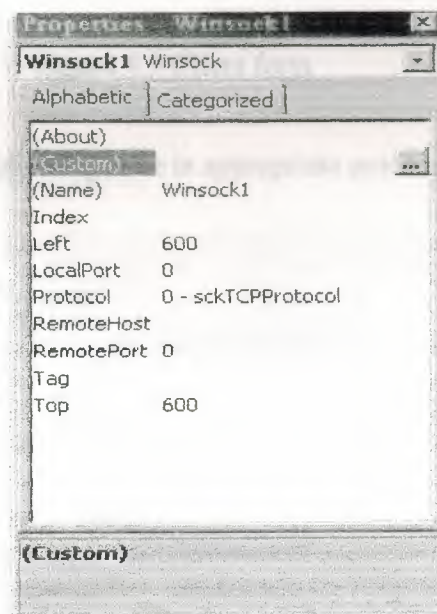


Figure 4.4 Winsock properties Window

5. Create the text-box and the command-buttons on your form as shown in Figure 4.5 and name them as follows:

Message label	lblMsg
Message text-box	txtMsg
SEND command-button	cmdSend
EXIT command-button	cmdExit

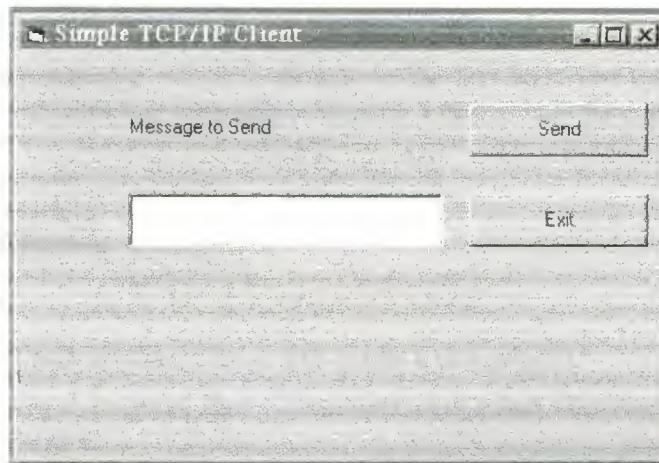


Figure 4.5 Client form

- 6 Enter the following code in appropriate procedures:

```

' CLIENT Program
' Connect to Server and send messages
'
Option Explicit

```

```

Private Sub Form_Load ()
Winsock1.Protocol = sckTCPProtocol
Winsock1.RemoteHost = "SIERRA"
Winsock1.RemotePort = 1112
Winsock1.connect
End Sub

```

```

Private Sub CmdSend_Click ()
Winsock1.SendData txtMsg.Text
TxtMsg.Text = ""
End Sub

```

```
Private Sub cmdExit_Click()
Winsock1.Close
End
End Sub
```

Note that in procedure *Form_Load*, the protocol type is specified as TCP/IP, the remote computer name is specified as SIERRA, and the port number is set to 1112 (any other unused port number can be used). When the Send button is pressed, procedure *cmdSend_Click* is activated and this procedure

sends the message, which is already in the text-box. The text-box is then cleared ready for the next message. Procedure *cmdExit_Click* closes the connection.

4.5.1.2 Server Computer

1. Load up Visual Basic and create a new project.
2. Go to Projects, then Components and select *Microsoft Winsock Control 6.0* as before.
3. Place the winsock control on the form and name the control as Winsock 1.
4. Create the text-box and the command-buttons as shown in **Figure 4.6** and name them as follows:

Message label Received	lblMsg
Data text-box EXIT	txtReceived
Command-button	cmdExit

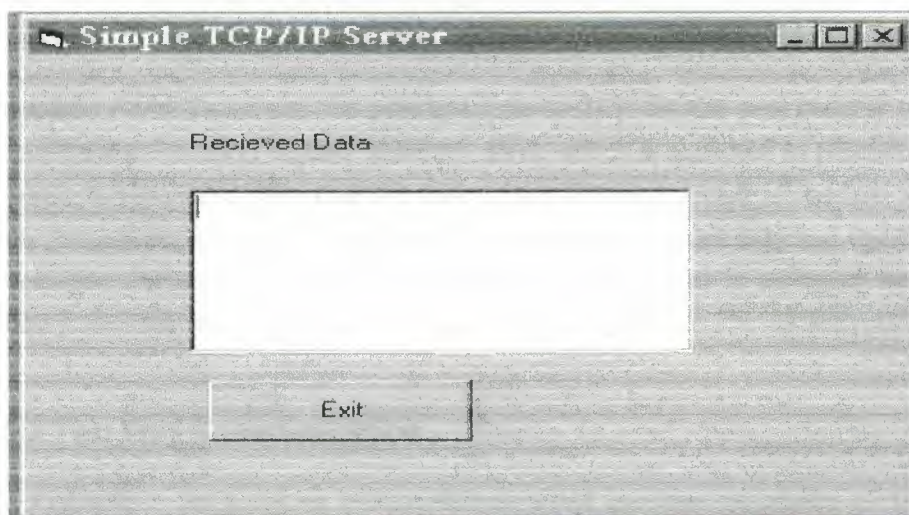


Figure 4.6 Server form

- 5 Enter the following code in the appropriate procedures:

```
' SERVER Program  
' Accept connction and display received data  
'  
Option Explicit
```

```
Private Sub Form_Load()  
Winsock1.protocol = sockTCPProtocol  
Winsock1.LocalPort = 1112  
Winsock1.listen  
End Sub
```

```
Private Sub Winsock1_ConnectionRequest(ByVal requestID As Long)  
If Winsock1.State <> sockClosed Then  
Winsock1.close  
End If  
Winsock1.Accept requested  
End Sub
```

```
Private Sub cmdExit_Click()  
Winsock1.Close  
End  
End Sub
```

Procedure *FormJLoad* sets the protocol type to TCP/IP, the port number to 1112, and listens for a connection request from the client, which is accepted in procedure *Winsock1_ConnectionRequest*. Note that the state of the listening socket is checked and closed if open. Procedure *Winsock1_DataArrival* retrieves the received data and displays this data in text-box *txtReceived*. Finally, procedure *cmdExit_Click* is activated when the exit button is pressed and this procedure closes the TCP/IP link.

4.5.2 Two Way Communication

The example in the previous section demonstrated the basic principles of TCP/IP based communication. We shall now develop a more complicated example where both the

server and the client can send and receive data. In addition, you should be able to specify the remote computer name at run time so that the program can be used to connect to any server.

Since you are now familiar with the steps of using the Winsock control, only the project forms and the code shall be given in this section.

4.5.2.1 CLIENT COMPUTER

Create the form details shown in Figure 4.7 and name the Winsock control as Winsock1. Name the various controls as follows:

Data to be sent label	lblSend
Received data label	lblReceived
Remote host name label	lblHost
Send command-button	cmdSend
Connect command-button	cmdConnect
Exit command button	cmdExit

The following code should be entered into the appropriate procedures:

```
' CLIENT Program
' Connect to Server and send and receive messages
'
Option Explicit
```

```
Private Sub Form_Load()
Winsock1.protocol = sckTCPProtocol
CmdSend.Enabled = False
```

```
Private Sub CmdConnect_Click()
Winsock1.Connect txtRemote, 112
End Sub
```

```
Private Sub cmdExit_Click()
Winsock1.Close
End
End Sub
```

```
Private Sub CmdSend_Click()
Winsock1.SendData txtSend.Text
```

```
TxtSend.Text = ""  
End Sub
```

```
Private Sub Winsock1_Connect()  
CmdSend.Enabled = True  
CmdConnect.Enabled = False  
  
End Sub
```

```
Private Sub Winsock1_DataArrival(ByVal bytes Total As Long)  
Dim msg As string  
Winsock1.GetData msg  
TxtReceived.Text = msg  
End Sub
```

Procedure *Form_Load* defines the protocol type as TCP/IP and disables the Send command-button on the form. This button will be enabled after a connection is made to the client. Procedure *cmdConnect_Click* initiates a connection to the server on port number 1112. Procedure *cmdExit_Click* closes the connection and terminates the program. Procedure *cmdSend_Click* sends the data in text-box *txtSend* to the remote machine. Procedure *Winsock1_Connect* is a winsock event driven procedure and is activated whenever a connection is established. The Send command-button is enabled and the Connect command-button is disabled in this procedure. Procedure *Winsock1_DataArrival* is activated when a data packet is received by the program. The received data is displayed in the text-box called *txtReceived*.

The screenshot shows a Windows-style application window titled "TCP Client". Inside the window, there are three text input fields and four buttons. The first text field is labeled "Data to be Send" and has a "Send" button to its right. The second text field is labeled "Received data". The third text field is labeled "Remote Host Name". Below the "Remote Host Name" field is a "Connect" button. To the right of the "Connect" button is an "Exit" button. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Figure 4.7 Client form

4.5.2.2 Server Computer

Create the form details shown in Figure 4.8 and load two identical Winsock controls on the form, both named Winsock1. Set the Index of one of the controls to 0 and the other one to 1. Name the various controls on the form as follows:

Data to be sent label	lblSend
Data to be Sent Text-Box	txtSend
Received data label	lblReceived
Received Text-Box	txtReceived
Send command-button	cmdSend
Exit command button	cmdExit

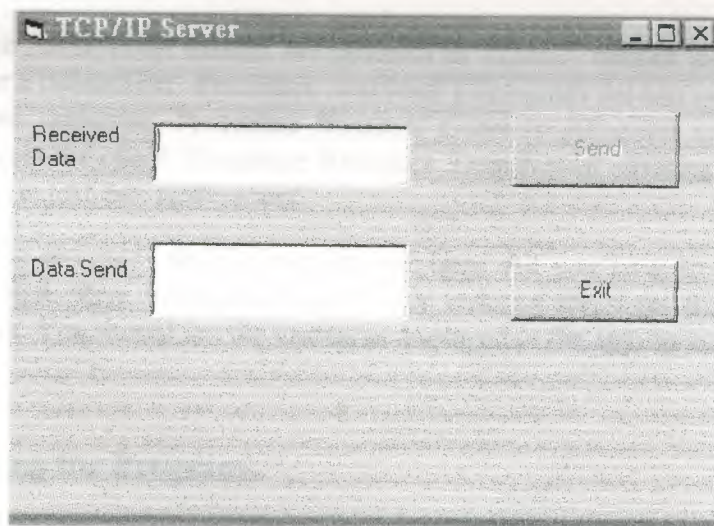


Figure 4.8 Server form

The following code should be entered into the appropriate procedures:


```
' SERVER Program
' Accept connection from the Client and display received data
' Option Explicit
```

```
Private Sub form_Load()
CmdSend.Enabled = False
Winsock1.Protocol = sockTCPProtocol
Winsock1.LocalPort = 1112
Winsock1.Listen
End Sub
```

```
Private Sub cmdExit_Click()
Winsock1.Close
End
End Sub
```

```
Private Sub cmdSend_Click()
Winsock1.SendData txtSend.Text
txtSend.Text = ""
End Sub
```

```
Private Sub Winsock1_DataArrival(ByVal bytes Total As Long)
Dim msg As String
Winsock1.GetData msg
txtReceived.Text = msg
End Sub
```

Procedure *Form_Load* defines the protocol as TCP/IP, defines the local port and then listens for a connection. Procedure *cmdSend_Click* sends the text in text-box *cmdSend* to the remote computer. Procedure *Winsock1_ConnectionRequest* accepts the connection request from the client. Procedure *Winsock1_DataArrival* retrieves the received data and displays it in text-box *txtReceived*.

Testing the programs should be relatively easy. All you have to do is run the server Program first, then the client program. Try entering messages from both sides of the connection. You should see the messages displayed in the appropriate text-boxes.

4.6 Testing the Programs

Testing the simple client-server programs you have created is easy. Run the server program first. Then run the client program either on the same computer or on another computer. Remember to change the IP names to match the TCP/IP configurations of your computer. Now enter a message on the client computer and press the Send button. The message should be displayed on the server computer. Press the exit button to terminate

the application.

CONCLUSION

We have described the structure of the system and the way the data is stored over the network. We have shown how the system can be used to manage the data of a business application. The system is designed to be used in a business environment and it is designed to be used in a business environment.

We have shown how the system can be used to manage the data of a business application. The system is designed to be used in a business environment and it is designed to be used in a business environment. We have shown how the system can be used to manage the data of a business application. The system is designed to be used in a business environment and it is designed to be used in a business environment.

We can recommend that the project be expanded in the future by adding hardware to the client computer. This way, we can physically control and monitor external equipment from a local server computer.

CONCLUSION

In this project we have described the client-server based communications using the TCP/IP protocol over the Internet. We have shown that client-server communication is very well suited to remote control based applications. We have used the Visual Basic language in our project with the Winsock control. This has made the programming very easy and well structured that sends the simple text to each other.

As Client requests to Server after accepted request by Server both computers can communicate each other and in this way Client can share the other resources of the network e.g. data storage over the Server. Where it connected. Then connections are made through the winsock control and text can be sent over the Internet while curing the protocols of Internet. While taking the concept of this simple and basic structure of the client-server based application we can develop highly civilized and widely used applications e.g. famous chat applications icq and mirc ,e-mail and finally the file attachments with e-mails.

We can recommend that the project be expanded in the future by adding hardware to the client computer. This way, we can physically control and monitor remote equipment from a local server computer.

BIBLIOGRAPHY

- A guide for writing the base of Internet:

ARPANET to INTERNET and beyond... by Peter Salus (Addison-Wesley, 1995) and Where Wizards Stay Up Late: The Origins of the Internet by Katie Hafner and Mark Lyon (Simon & Schuster, 1997).

- A guide for writing on the basic concept of the Networking and client/sever Microsoft's Networking Essentials from Microsoft Press

Referred to Internetworking with TCP/IP,
Vol. I: Principles, Protocols, and Architecture,
2/e, by D. Comer (Prentice-Hall, 1991)

TCP/IP: Architecture, Protocols, and Implementation with IPv6 and IP Security,
2nd. Edition . by S. Feit (McGraw-Hill, 1997),
"TCP/IP Tutorial" by T.J. Socolofsky and C.J. Kale (RFC 1180),
TCP/IP Illustrated, Volume I:
The Protocols by W.R. Stevens (Addison-Wesley, 1994).

- A guide for writing the program on Visual basic.

www.vbcode.com

www.programmersheaven.com

<http://www.hill.com> or <http://www.sover.net/~kessfam>