

CLIENT-SERVER COMMUNICATION IN REMOTE CONTROL

**A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**

OF

CANKAYA UNIVERSITY

120 859
BY

ROYA CHOUPANI

**IN PARTIAL FULLFILMENT OF THE REQUIRMENTS FOR THE
DEGREE OF MASTER OF SCIENCE**

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

**TC. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ**

JULY 2002

120859

Approval of the Graduate School of Science and Engineering



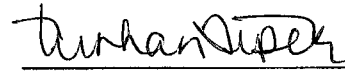
Prof. Dr. Üstün DİKEÇ
Director

I certify that this thesis satisfies all requirements as a thesis for the degree of Master of Science in Computer Engineering



Prof. Dr. Turhan ALPER
Chairman of the Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Computer Engineering.



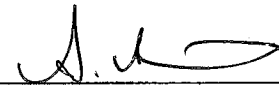
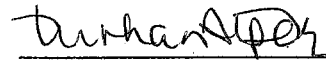
Prof. Dr. Turhan ALPER
Supervisor

Examining Committee Members

Prof. Dr. Turhan ALPER

Assist. Prof. Dr. Gülsün TÖRECI

Assist. Prof. Dr. Alırıza AŞKUN



Abstract

The client server communication model has been used in remote controlling of devices. The main feature in this study is that the Internet has been used as the common media to transmit controlling data and receive information. Client server model on the Internet restricts the access to client computers and has the disadvantage of unknown platform on the client side. This problem has been solved by means of platform independent programming and java applets. Socket interface available in application layer of TCP/IP protocol suit has been used to establish reliable connection between clients and server. Security issues have been dealt with in the server side by checking the IP addresses of requesting clients.

ÖZ

Cihazların uzaktan kontrolunda istemci sunucu iletişim modeli kullanılmıştır. Bu çalışmanın temeli, kontrol verilerinin transferi için internet ortamının kullanılmasına dayanmaktadır. İnternet üstündeki istemci sunucu modeli, istemci bilgisayara bağlantıyı sınırlamaktadır ve istemci tarafındaki platformun bilinmemesi bu modelin dezavantajıdır.

Bu problem platform bağımsız programlama ve Java Applet'leri aracılığı ile çözülmüştür.

TCP/IP protokolünün uygulama katmanında bulunan soket arayüzü kullanılarak istemci ve sunucu arasında güvenilir bağlantı kurulmuştur. Sunucu tarafında, istekte bulunan istemci IP adreslerini kontrole dayanan bir güvenlik metodu kullanılmıştır

To my husband and my dear son Mazdak



Acknowledgment

I wish to express my deep gratitude to my advisor Prof. Dr. Turhan ALPER for his valuable advices, guidance and encouragement. Also sincere thanks to Dr. Süleyman KONDAKCI for his help and advices. Also I would like to thank Gökhan YILMAZ for his help to complete and test the implemented code.

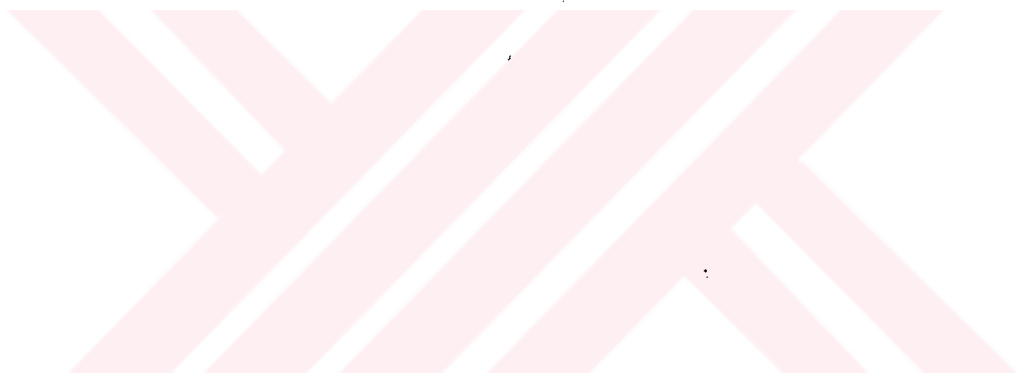


Table of contents

CHAPTER 1. Introduction.....	1
CHAPTER 2. TCP/IP Protocol suite	3
2.1 Introduction.....	3
2.2 Protocol Suite.....	3
2.3 User Datagram Protocol (UDP).....	5
2.4 Transmission Control Protocol (TCP).....	7
2.5 Packet delivery and routing	9
2.6 Application layer and client server model.....	13
The client program.....	14
The server program.....	14
2.7 Client-Server Interaction: Socket programming.....	14
CHAPTER 3. Platform independent programming and Java language.....	18
3.1 Introduction.....	18
3.2 Java Language.....	19
3.3 Java Applets.....	21
CHAPTER 4. Remote Control System.....	23
4.1 Introduction.....	23
4.2 System Hardware.....	23
4.3 Software architecture: Client-Server Model.....	25
CHAPTER 5. Experimental results and discussion.....	31
References:.....	34

List of figures

Figure 2-1 Direct delivery of a packet in a network	10
Figure 2-2 Default routing	12
Figure 2-3 The order of calling socket library functions in client and server.....	17
Figure 3-1 Compilation process in java virtual machine.....	20
Figure 3-2 The JIT compiler in the client system.....	21
Figure 4-1 Overall structure of the remote control system.....	24
Figure 4-2 The rotation axes of the camera.....	24
Figure 4-3 User interface of the client program.....	30



List of tables

Table 1 Differences between java applets and applications	22
Table 2 Sample commands sent to server form a client	26



Chapter 1

Introduction

Remote control of devices has obtained a wide application recently. This is due to the need for automating the control that has traditionally been administered by human operator on one hand and on the other hand the enormous improvement in the underlying communication technology. Today, the Internet is part of our life and accessing remote data and information is a trivial and simple matter. This large facility has invoked the idea that other than a huge pile of information resource, the Internet may be considered as communication mechanism transferring controlling information and commands. Remote control of a device first of all needs the current state of the device to be controlled. This means that there should be kind of state data transferring from the device to the controlling system. Similar systems have been reported in [1] and [2].

In [1] the method has been applied in a teleconference. They have reported the occurrence of simultaneous access to the device and trying to control it by several users. This problem which is one the intrinsic properties of all conferences, can frustrate it from its main goal. Videoconferencing doesn't allow for moderated discussions. For multi-participant meetings, there is no way to stop everyone from talking at once. The **goal of the floor control** project is to allow an individual to moderate meetings and control "who has the floor".

The **objectives** of the floor control are:

- Integrate mechanisms for scalable floor control into the tool architecture,
- Allow meetings to occur over the Web,
- Develop two prototype coordination tools:
 - Moderated meeting tool

- Consensus meeting .
- Integrate floor management into session management

In addition, videoconferencing often requires the meeting's host to run cameras and other equipment to allow remote users to participate. This disrupts the meeting's flow and frustrates remote users. The **goal of the remote camera** project is to provide remote collaborators with the ability to receive and control media at any time, a sense of telepresence, media tools (video/audio), and the ability to "walk around" the remote space.

In [2] the authors argue that growth of network technology makes it smooth to communicate between long distant places. Ability of controlling remote devices contributes to improve the quality of communication. However, there are few discussions about the standardization of a protocol for controlling remote devices under such environments. In their paper, they design and evaluate a network protocol called common remote control protocol (CRCP), which is for the purpose of being a common method. Then they develop a system to control remote devices used in teleconference systems. They also apply it to the real teleconferences and discuss the problem and the measures devised to deal with.

In this system we have used a *java* based application to control a camera from a client connected to the Internet. Since the underlying protocol in the Internet is TCP/IP we have used socket programming facility provided by this protocol and a reliable connection oriented data transmission to realize the system, This report has the following structure. Chapter 2 is a review of TCP/IP protocol and its facilities. In this chapter I will also explain socket programming interface and its functions. Chapter 3 is about platform independent programming and java language. This chapter discusses Applets as the main feature of java language in writing Internet based applications. Chapter 4 explains the details of the system in two different hardware and software subsections. Chapter 5 has the discussion and experimental results.

Chapter 2

TCP/IP Protocol suite

2.1 Introduction

In this chapter we will briefly review the methods used in TCP/IP protocol suite. This review together with the review of java programming language and java applets covered in the next chapter will help me to justify my own algorithm and make it clear why I have used a connection oriented socket application using java applets. In this chapter I will cover different layers of TCP/IP protocol and connection oriented and connection less transport protocols. Then routing and delivery of IP packets to their final destinations will follow. Client Server application model is the topic of next subsection and finally I will show how socket interface facility works in TCP/IP protocol.

2.2 Protocol Suite

TCP/IP is a hierarchical protocol made up of interactive modules each one providing a specific functionality, but the modules are not necessarily interdependent [3,4]. Whereas the Open System Interconnection (OSI) model specifies which functions belong to each of its layers. The layers of the TCP/IP protocol suite contain relatively independent protocols that can be mixed and matched depending on the need of the system. The term *hierarchical* means that each upper level protocol is supported by one or more lower level protocols. At the transport level, TCP/IP defines two protocols: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). At the

network layer, the main protocol defined by TCP/IP is the Internetworking Protocol (IP) although there are some other protocols that support data movement in this layer.

Physical and data link layers

At the physical and data link layer, TCP/IP does not define any specific protocol. It supports all the recommended standard and proprietary protocols. A network in a TCP/IP internetwork can be a local area network (LAN), a metropolitan area network (MAN), or a wide area network (WAN)

- ***Network Layer***

At the network layer, TCP/IP provides the Internetworking Protocol (IP). IP, in turn contains four supporting protocols: Address Resolution Protocol (ARP), Reverse Address Resolution Protocol (RARP), Internet Control Message Protocol (ICMP), and Internet Group Message Protocol (IGMP). IP is an unreliable and connectionless datagram protocol. This means that IP supports no error checking or tracking. IP relies on the upper layer protocols and assumes the unreliability of the underlying layers and does its best to get a transmission through to its destination, but with no guarantee. IP transports data in packets called ***datagram***, each one transported separately. Datagrams can travel along different routes and can arrive out of sequence or be duplicated. IP does not keep track of the route and has no facility for recording datagrams once they arrive at their destination. IP uses ARP and RARP to establish a relation between physical addresses and logical addresses, ICMP is used to send notifications of datagrams problems and IGMP is used to facilitate simultaneous transmission of a message to a group of recipients.

- ***Transport Layer***

The transport layer is represented in TCP/IP by two protocols: TCP and UDP which are transport level protocols responsible for delivery of a message from a process to another process whereas IP is a host-to-host protocol, meaning that it can deliver a packet from one

physical device to the other.. The UDP is the simpler of the two standards TCP/IP transport protocols. It is a process-to-process protocol that adds only port addresses, checksum error control, and length information to the data from the upper layer. The TCP provides full transport layer services to applications. TCP is a reliable stream transport protocol. The term stream here means connection oriented where a connection must be establish between both ends of a transmission before either of nodes can transmit data. At the sending end of the each transmission, TCP divides a stream of data into smaller units called segments. Each segment includes a sequence number for recording after receipt, together with an acknowledgment number for the segment received. Segments are carried across the internet inside of IP datagrams. At the receiving end, TCP collects each datagram as it arrives in, and reorders the transmission based sequence numbers.

- ***Application Layer***

The application layer in TCP/IP is equivalent to the combination of session, presentation, and application layers itself in the OSI model. Many protocols are defined at this layer. For example Simple Network Management Protocol (SNMP) and Hypertext Transfer Protocol (HTTP) are two protocols defined in this layer.

2.3 User Datagram Protocol (UDP)

UDP is a connection less, unreliable transport protocol. It does not add anything to the services of IP except for providing process-to-process communication instead of host-to-host communication. Also, it provides very limited error checking and simple integrity checking. The main advantages of UDP are its simplicity and minimum overhead added to the system. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message using UDP takes much less interaction between the sender and receiver than it does using TCP. The IP is responsible for communication at computer level. As a network layer protocol, IP can deliver

the messages only to the destination computer. However, this is an incomplete delivery. The message still needs to be handed to the correct process. This is where a transport layer protocol such as UDP takes over. UDP is responsible for delivery of the messages to the appropriate process. Although there are few ways to achieve process-to process communication, the most common one is through the client-server method. A process on a local host, called a client, needs services usually on a remote host, called server. Both processes (client and server) might have the same name. For example, to get the day time from a remote machine, we need a Daytime client process running on the local host and a Daytime server process running on the remote machine. However, operating systems today support both multi-user and multiprogramming environments. A remote computer can run several server programs concurrently, just as several local computers can run one or more client programs at the same time. For any communication we must define the

- Local host,
- Local process,
- Remote host,
- Remote process.

The local host and the remote host are represented by their IP addresses. To define the processes, we need other identifiers, which are called *port numbers*. In the TCP/IP suite, the port numbers are integers between 0 and 65535. The client program defines itself with a port number, chosen randomly by the UDP software running on the client host. The server process must serve through a port number. This port number (service identifier), however, cannot be chosen randomly. If the computer at the server side runs a server process and assigns a random number as the port number, the process at the client side that wants to access that server and use its services will not know the port number.

Since UDP is a connectionless service, each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are coming

from the same source process and going to the same destination process. The user datagrams are not numbered. Also, there is no connection establishment and no connection terminations we will see in the case of the TCP protocol. This means that each user datagram can travel via different paths. This also means that we can not send a stream of data using UDP protocol and expect UDP to chop them into different related user datagrams. Instead each request must be small enough to fit into one user datagram. Only those processes sending short messages should use UDP.

UDP is a very simple, unreliable transport protocol, there is no flow control, and hence no windowing mechanism. The receiver may overflow with incoming messages. There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error using the checksum, the user datagram is silently discarded. The lack of flow control and error control means that the process using UDP should provide for these mechanisms.

2.4 Transmission Control Protocol (TCP)

TCP offers services that are not offered by its counterpart UDP. Among these services we may mention some as:

- TCP is considered a stream transport layer service, which means the sending TCP accepts a stream of characters from the sending application program, creates packets, called segments, of appropriate size extracted from the stream, and sends them across the network. The receiving TCP receives segments, extracts data from them, orders them if they have arrived out of order, and delivers them as a stream of characters to the receiving application program. For stream delivery, the sending and receiving TCPs use buffers. The sending TCP uses a sending buffer to store the data coming from the sending application program. The sending program delivers data at the rate it is created. For example, if the user is typing the data on a keyboard, the data is delivered to the sending TCP character by character. If

the data is coming from a file, data may be delivered to the sending TCP line by line or block by block. The sending application program writes data to the buffer of the sending TCP. However, the sending TCP does not create a segment of data for each write operation issued from the sending application program. TCP may choose to combine the result of several write operations into predefined segments to make transmission more efficient. The receiving TCP receives the segment and stores them in a receiving buffer. The receiving application program uses the read operation to read the data contained in one segment in one operation, but since the rate of reading can be slower than the rate of receiving data, the received data is kept in a buffer until the receiving application reassembles it completely.

- TCP offers full-duplex service, where data can flow in both directions at the same time. After two application program are connected to each other, they can both send and receive data. One TCP connection can carry data from application A to B and, at the same time, data from B to A. when a packet is going from A to B, it can also carry an acknowledgement of the packets received form B.
- TCP is a reliable transport protocol. It uses an acknowledgement mechanism to check the safe and sound arrival of data.
- TCP is a connection-oriented protocol. A connection-oriented protocol establishes a virtual path between the source and destination. All of the segments belonging to a message are then sent over this virtual path using a single virtual pathway for the entire message facilitates the acknowledgement process as well as retransmission of damaged or lost segments.
- Connection establishment should be achieved before any data transfer, each side should initialize a communication and get approval from the other side for it. This procedure needs four steps to follow:
 - Host A sends a segment to announce its request

**Y.C. YÜKSEKÖĞRETİM KURULU
DOKÜMANTASYON MERKEZİ**

- Host B sends a segment to acknowledge the request from A
- Host B sends a segment that includes its initialization information about traffic from A to B
- Host A send a segment to acknowledge the request of B
- Any of two sides involving in exchanging data can also close the connection. When the connection in one direction is terminated, the other side can still continue sending data.

Therefore the termination of the connection will have the following steps:

- Host A sends a segment announcing its request for connection termination.
- Host B sends a segment acknowledging the request of A. after this, the connection is closed in one direction, but not in the other.
- When host B has finished sending its own data, it sends a segment to indicate that it wants to close the connection.
- Host A acknowledges the request of B, and the connection is terminated.

2.5 Packet delivery and routing

By delivery, we mean the physical forwarding of data packets and by routing, we also mean finding the rout (next hop) for a datagram [5]. Concepts such as connectionless and connection oriented services, and direct and indirect delivery are discussed in this chapter.

Delivery of a packet in the network layer is accomplished using either a connection oriented or a connectionless network service. In a connection oriented situation, the network layer protocol first makes a connection with the network layer protocol at the remote side before sending packets. When the connection is established, the sequence of packets from the same source to the same destination can be sent one after another. In this case, there is a relationship between packets. They are sent on the same path where they follow each other. A packet is logically connected to the packet traveling before it and to the packet traveling after it. When all packets of a message have

been delivered, the connection is terminated. In a connection oriented protocol, the decision about the route of a sequence of packets with the same source and destination addresses can be made only once when the connection is established. Routers do not have to recalculate the route for each individual, packet. In a connectionless situation, the network protocol treats each packet independently, with each packet having no relationship to any other packet. The packets in a message may not travel the same path to their destination. The IP protocol is a connectionless protocol. It is designed this way because IP, as an internetwork protocol, may have to deliver the packets through several heterogeneous networks. If IP were to be connection oriented, all of the networks in the internet should also be connection oriented, which is not the case. The delivery of a packet to its final destination is accomplished using two different methods of delivery: direct and indirect[6]. In a direct delivery, the final destination of the packet is a host connected to the same physical network as the source. Direct delivery occurs when the source and destination of the packets are located on the same physical network or if the delivery is between the last router and the destination host (Figure 2.1).

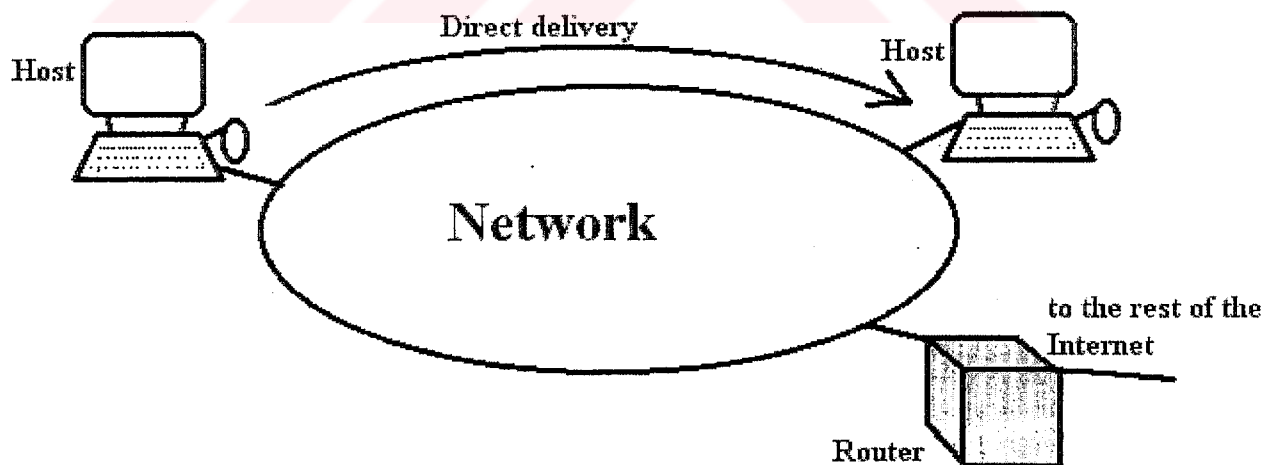


Figure 2-1 Direct delivery of a packet in a network

The sender can easily determine if the delivery is direct. It can extract the network address of the destination packet (setting the hosted part to all 0s, broadcast) and compare this address of the networks to which it is connected. If a match is found, the delivery is direct. In direct delivery, the sender uses the destination IP address to find the destination physical address. The IP software then delivers the destination IP address with the destination physical address to the data link layer for actual delivery. This process is called mapping the IP address to the physical address. This is done by the aid of address resolution protocol (ARP). Although, this mapping can be done by finding the match in a routing table if the destination host is not on the same network as the deliverer, the packet is delivered indirectly. In an indirect delivery, packets go from the router until it reaches the one connected to the same physical network as its final destination. A delivery always involves one direct delivery but zero or more indirect deliveries. Also last delivery is always a direct delivery. In an indirect delivery, the sender uses the destination IP address and a routing table to find the IP address of the next router to which the packet should be delivered. The sender then uses the ARP protocol to find the physical address of the next router. In direct delivery the address mapping is between the IP address of the final destination and the physical address of the final destination. In an indirect delivery, the address mapping is between the IP address of the next router and physical address of the next router.

Routing requires a host or a router to have a routing table. When a host has a packet to send or when a router has received a packet to be forwarded, it looks its table to find the next router to the final destination. However, this simple solution is impossible today in the internetworks such as the Internet because the number of entries in the routing table make table lookups inefficient several techniques can make the size of the routing table manageable and handle such issues as security. One technique to make the contents of a routing table smaller is called next hop counting. In this technique, the routing table holds only the address of the next hop instead of holding information about the complete route. Routing tables are thereby consistent with each other.

A second technique to make the routing table smaller and searching process simpler is called network specific routing. Here, instead of an entry for every host connected to the same physical network, we have only one entry to define the address of the network itself. In other words, we treat all hosts connected to the same network as one single entity. For example if 1000 hosts are attached to the same network, only one entry exists in the routing table instead of 1000 entries. Next method is host specific routing, in which the host address is given in the routing table. The idea of host specific routing is inverse of network specific routing. Here, efficiency is sacrificed for other advantages. Although, it is not efficient to put the host address in the routing table, there are occasions in which the administrator wants to have more control over routing. For example if the administrator wants all packets arriving for host B delivered to router R3 instead of R1, one single entry in the routing table of host A can explicitly define the route. Host specific routing is a good choice for certain purposes such as checking the route or providing security measures.

Another technique used to simplify routing is default routing. As shown in figure 2.2, host A is connected to a network with two routers. Router R1 is used to route packets to hosts connected to network N2. However, for the rest of the Internet, router R2 should be used. Bearing this in mind, instead of listing all networks in the entire Internet, host A can just have one entry called default routing.

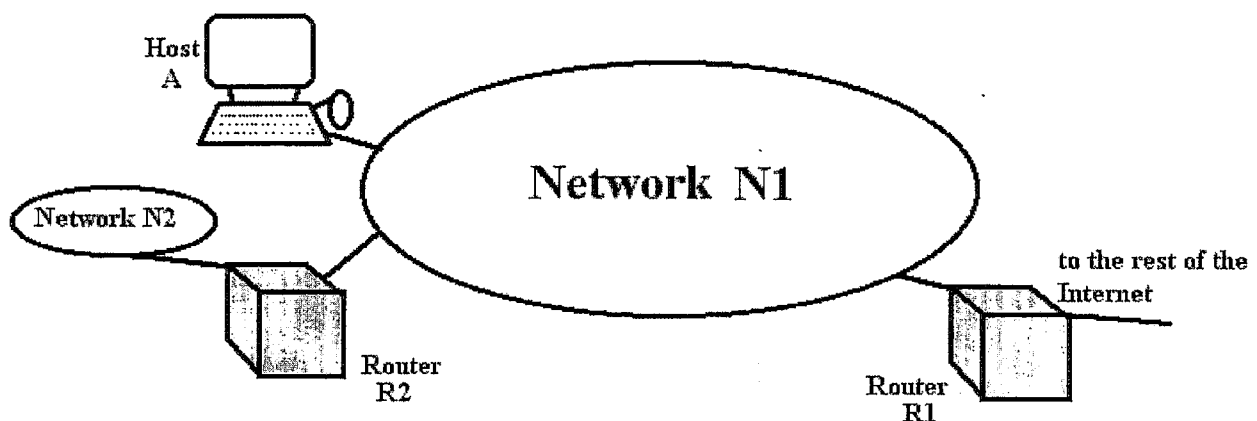


Figure 2-2 Default routing

2.6 Application layer and client server model

The purpose of a network, or an internetwork, is to provide services to users. A user at a local site wants to receive a service from a computer at a remote site. There is only one way for a computer to do the job: it must run a program. A computer runs a program to request a service from another program and also to provide a service to another computer. This means that two computers, connected by an internet, should each run a program, one to provide a service and the other to request a service. Therefore if we want to use the services available on the Internet, application programs running at two end computers and communicating with each other are needed. In other words, in the Internet, the application programs are the entities that communicate with each other not with the computer or users. There are some considerations in writing a service providing and a service requesting system. These considerations may be as following:

- Should both application programs be able to request services and provide services or should the application program just do one or the other? One solution is to have an application program called the *client* running on the local machine, requesting a service from another application program, called the *server* running on the remote machine. In other words, the tasks of requesting a service and providing a service are separated from each other. An application program is either a requester (client) or a provider (server). If a machine needs to request a service and provide a service, two application programs must be installed.
- Should an application program provide service only to one specific client or any application program requesting this service? The most common solution is a server providing services for any client, not a particular one. In other words the client-server relationship is a many-to-one relationship.
- When should an application program be running? Continuously or just when there is a need for service? Generally a client program will run when it needs a service but the server should be available at any time, because it doesn't know when its service will be needed.

- Should there be a universal application program that can provide any type of service a user wants? In TCP/IP, the solution is that services needed frequently and by many users have specific client-server application programs.

The client program

A *client* is a program running on a local machine requesting service from a server. A client program is finite, which means it is started by the user (or another application program) and terminates when the service is complete. A client opens the communication channel using the IP address of the remote host and its port number. After a channel of communication is opened, the client sends its request and receives a response. Although, the request-response part may be repeated for several times, the whole process is finite and eventually comes to an end. At this moment, the client closes the communication channel.

The server program

A *server* is a program running on the remote machine providing service to its clients. When it starts, it opens the door for the incoming requests from clients, but it never initiates a service until it is requested to do so. This is a well known passive property of a server program. A server program is an infinite program. When it starts it runs infinitely unless a problem arises. It waits for incoming requests from clients. When a request arrives it responds the requests and waits for the next one.

2.7 Client-Server Interaction: Socket programming

In a client-server model, two application programs, one running on the local system and the other running on the remote system need to communicate with one another. To standardize network programming, application programming interfaces (APIs) have been developed. An API is a set of declarations, definitions, and procedures used by programmers to write client-server programs. Among the most common APIs are the *Socket Interface* [6,9], the Transport

Layer Interface (TLI), and The Remote Procedure Call (RPC). The socket interface was first developed as a part of UNIX BSD. It is based on the earlier UNIX versions and defines a set of system calls (procedures) that are an extension of system calls used in all UNIX based operating systems to access files.

The communication structure that is necessary in socket programming is a *socket*. A socket acts as an end point to a process. Two processes need a socket at each end to communicate with each other. A socket, in its lowest level, is defined in the operating system as a structure. The fields of this structure have details like:

- **Type** This field defines the type of socket. Socket type can be stream socket for connection oriented communication, datagram socket for connection-less, or raw socket which is used by ICMP protocol for direct access to the services of IP protocol.
- **Protocol.** This field is usually set to zero for UDP and TCP.
- **Local socket address.** This field defines the local host address.
- **Remote socket address.** This field defines the remote host address

Socket system calls have been implemented as system functions that can be called by application program to communicate with another application programs. Some of these functions are as follows:

- **Socket** This function is used to create a socket. The function gets address family, connection type and protocol as parameters.
- **Bind** this function binds a socket to a local address by adding the host socket address to an already created socket. The function gets the socket id and local address as input.
- **Listen** this function informs the operating system that the server is ready to accept client connections through this socket. The function gets the socket id and number of requests that can be queued for this socket as input.

- **Accept** The accept function is called by a TCP server to remove the first connection request from the corresponding queue. If there is no request the caller of the accept function (TCP server) will be put to sleep.
- **Connect** The connect function is called by a client to establish an active connection to a remote process (normally a server).
- **Read** The read function is used by a process to receive data from another process running on a remote machine. The function assumes that there is already an open connection between two machines, therefore it can only be used by TCP processes.
- **Write** The function is used by a process to send data to another process running on a remote host and using TCP protocol. It also assumes that a connection is already open between the two processes.

The order of using these functions is very important. The server first opens a socket which is a request to the operating system to create a socket. Then it binds it to the local address. It then calls listen function which converts the process to a passive one waiting for request from other sides. Server process finally uses accept function to establish the connection with the first request from a client available in its queue.

Client, on the other hand, opens a socket, then uses connect function to send a request to the server and after establishment of the connection, starts sending and/or receiving data. Figure 2-3 shows the steps followed in a socket based data transmission.

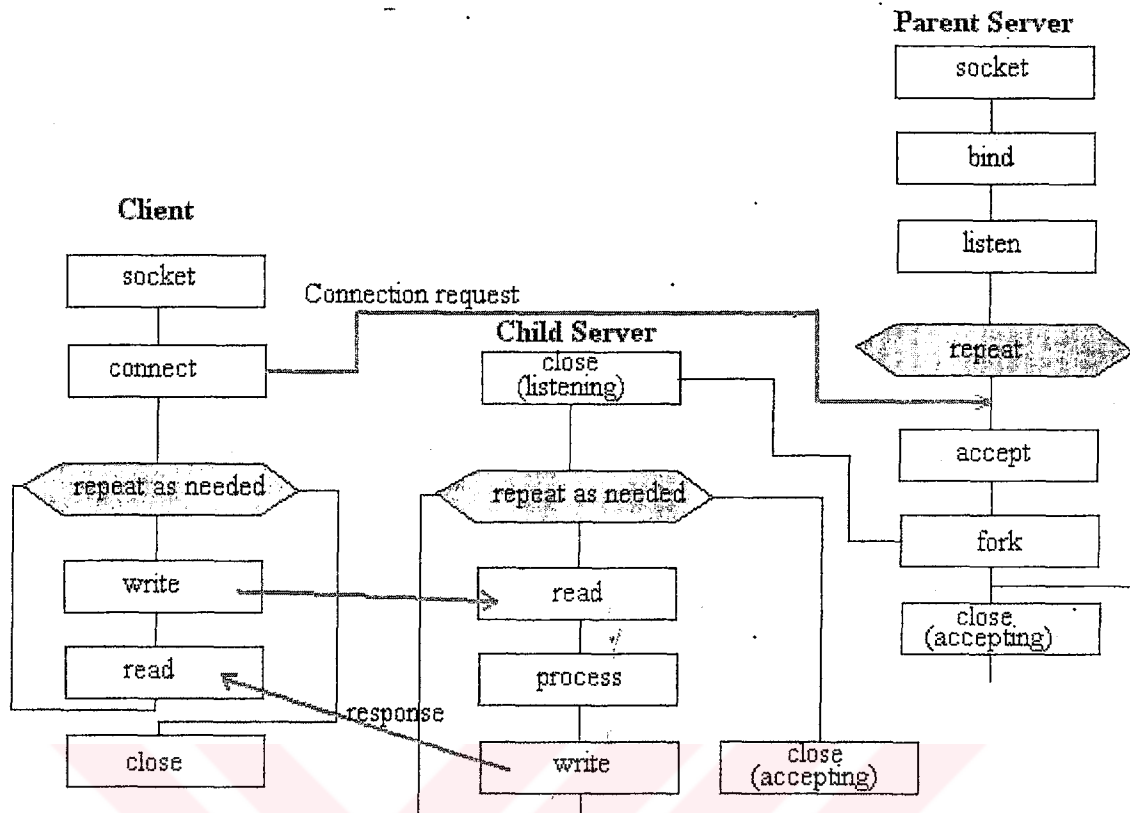


Figure 2-3 The order of calling socket library functions in client and server

Chapter 3

Platform independent programming and Java language

3.1 Introduction

java is a technology that makes it easy to build distributed applications, which are programs executed by multiple computers across a network [10]. This means by expanding the network programming, java expands the role of the Internet from an arena for communication to a network on which full-fledged applications can be run. This technology will allow business to deploy full-scale transaction services and real-time, interactive information on the Internet. Before java, the Internet was primarily used for information-sharing. Though, the Internet was created in the 1960s, it only started to realize its business potential in the 1990s, thanks to the World Wide Web. The Web is a technology that treats Internet resources as linked documents, and it has revolutionized the way we access and share information. The reason why so much attention has been paid to java is:

- Using java development time of application programs can be reduced. This increases programmers efficiency.
- Build an application on almost any platform, and run that application on any other supported platform without recompilation.
- Distribute applications over a network in a secure fashion
- Software reuse by object oriented approach

In particular, java programs can be embedded into Web documents, turning static pages into applications that run on the user's computer. No longer is online documentation limited to articles,

like a printed book. With java, the documentation can include simulations, working models, and even specialized tools. This means that java has the potential to expand the function of the Internet. It is also important to know that java is not appropriate in some cases and for some applications.

Among the problems that java poorly handles are:

- Performance critical problems.
- Large problems with large memory or I/O operations require application to take more active roles in managing memory or I/O. java is not optimized to solve such problems.
- Platform specific problems. Java takes great pain to achieve platform independence so you can not develop a code with java to create a symbolic link, develop an X Window manager, read UNIX environment variables, or identify the owner of a file. That is, java is not developed to go into the kernel level operation. It is a very high level programming language.
- GUIs. Of course java has GUI features but GUI performance needs a great deal of attention if java is to be used as a serious GUI platform.

3.2 Java Language

As with many other programming languages, java uses a compiler to convert human readable source code into executable programs. Traditionally compilers produce code that can be executed by specific hardware. For example, a Windows 95 C++ compiler creates executable programs that work with Intel x86 compatible processors. In contrast, the java compiler generates architecture independent *bytecodes*. The bytecodes can be executed by only a java virtual machine (JVM), which is an idealized java processor chip usually implemented in software rather than hardware. Java bytecode files are called class files. To execute java bytecodes, the JVM uses a class loader to fetch the bytecode from a disk or from the network. Each class file is fed to a bytecode verifier that ensures that the class is formatted correctly and that the class will not corrupt the memory when it is executed. The bytecode verification adds to the time it takes to load a class, but it actually allows

the program to run faster because the class verification is performed only once, not continuously as the program runs. The execution unit of the JVM carries out the instructions specified in the bytecodes. The simplest execution unit is an interpreter, which is a program that reads the bytecodes, interprets them, and then performs the associated operation. Interpreters are generally much slower than native code compilers because they continuously need to look up the meaning of each bytecode during execution. Another alternative to interpreting code is just-in-time compilation (JIT). The JIT compiler converts the bytecode to native code instructions on the user's machine immediately before execution. Traditional native code compilers run on the developer's machine, are used by programmers, and produce non-portable executables. JIT compilers run on the user's machine and are transparent to the user. The resulting native code instructions do not need to be ported because they are already at their destination. The compilation process in java virtual machine is shown in figure 3-1, and figure 3-2 illustrates JIT method of compilation.

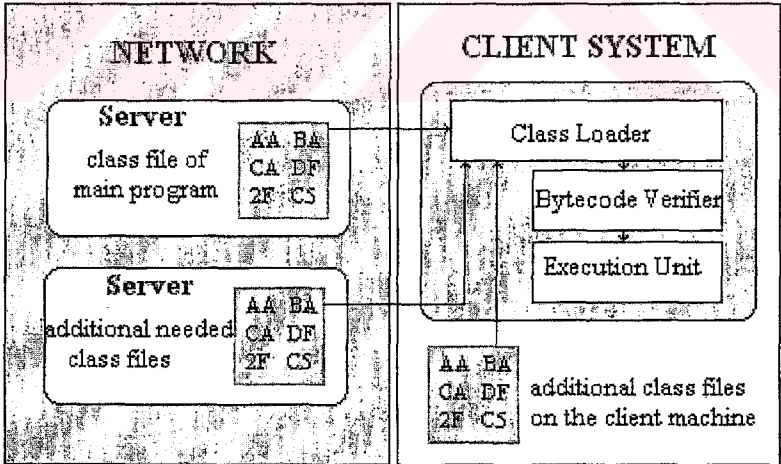


Figure 3-1 Compilation process in java virtual machine

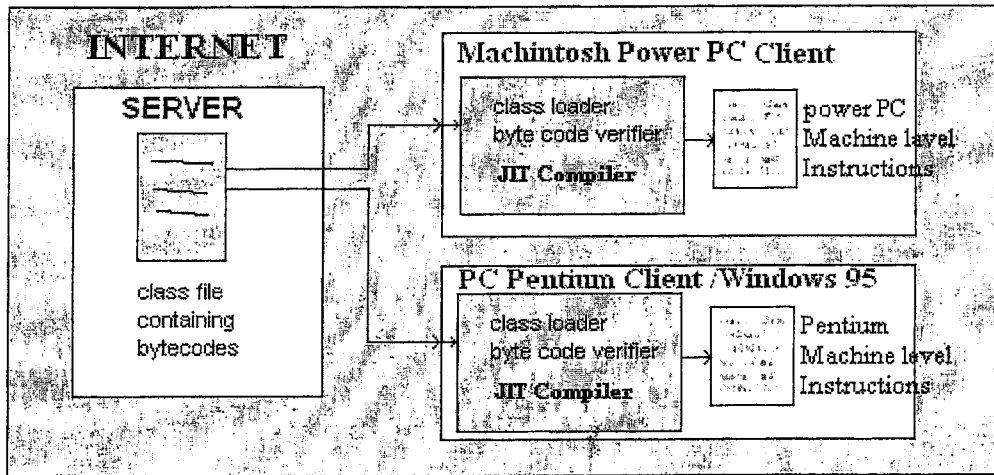


Figure 3-2 The JIT compiler in the client system

3.3 Java Applets

Traditionally the word applet bears the meaning of any small application. In Java, an applet is any java program that is launched from a Web document; that is, from an HTML file. Java applications, on the other hand, are programs that run from a command line, independent of a Web browser. There is no limit to the size or complexity of a java applet. In fact java applets are in some ways more powerful than java applications. However, with the Internet, where communication speed is limited and download times are long, most Java applets are intentionally kept small. The technical differences between applets and applications stem from the context in which they run. A Java application runs in the simplest possible environment. Its only input from the outside world is a list of command line parameters. On the other hand, a java applet receives its input from the Web browser. It needs to know when it is initialized, when and where to draw itself in the browser window, and when it is activated or deactivated. As a consequence of these two very different execution environments, applets and applications have different base requirements. The decision to

write a program as an applet versus an application depends on the context of the program and its delivery mechanism. Because java applets are always presented in the context of a Web browser's graphical interface, java applications are preferred over applets when graphical displays and direct access to server resources are unnecessary. For example a HyperText Transfer Protocol (HTTP) server written in java needs no graphical display; it requires only file and network access. The convenience of Web protocols for applet distribution makes applets the preferred program type for Internet applications, although java applications can easily be used to perform many of the same tasks. With java, developing Internet-based software, either as applets or applications is extremely easy. Non-networked systems and systems with small amounts of memory are more likely to be written as Java applications rather than as Java applets. Table 1 summarizes the differences between the two types of java programs.

	Java application	Java applet
Uses graphics	Optional	Inherently graphical
Memory requirements	Minimal java application requirements	Java application requirements Plus Web browser requirements
Distribution	Loaded from the file system or by a custom class loading process	Linked to a HTML file and Transported via HTTP

Table 1 Differences between java applets and applications

Chapter 4

Remote Control System

4.1 Introduction

Controlling devices from a Web page is a convenient, safe, and cost-effective alternative to managing them locally or controlling them via private networks. For example, scientists in remote locations can "meet" on a Web page at a laboratory, taking turns controlling an experiment such as a microscope and discussing the results via telephone. Companies with unsafe environments such as hazardous materials or underwater work can protect employees by letting them operate equipment remotely. Security companies can trig alarms or lock doors in response to intrusions. With our system, we are trying to control a device which in this case is a camera, through a web browser and view the images taken by this camera. By means of small modifications, the system is applicable to other devices. The main advantage of the system is that it doesn't require any special equipment on the user side and an Internet connection and a web browser is quite sufficient. We are applying a client-server model based on TCP/IP protocol which is the de facto protocol in the Internet. In the following sections I will describe the hardware and software of the system and in the next chapter the results and possible improvements will be discussed.

4.2 System Hardware

As mentioned previously, the system is modeled as a client-server architecture. The server is running on an Intel Pentium microprocessor with *linux* operating system. The client can be any computer connected to the Internet with any platform. The protocol used between the client and the

server is TCP/IP and the connection is inherently reliable one. A Z8 micro controller is used to control the device. A parallel port of Z8 is used for interfacing the server to the Z8 micro controller. The overall structure of the system is given in figure 4-1.

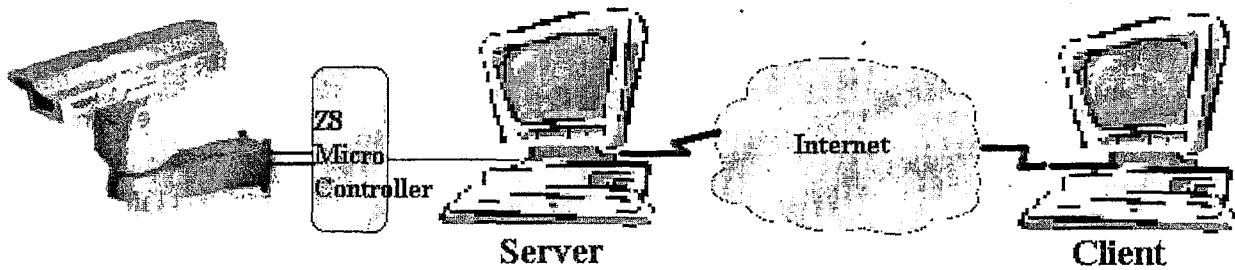


Figure 4-1 Overall structure of the remote control system

The camera is a web-cam which is connected to two stepper motors. As shown in figure 4.2, these stepper motors can rotate the camera about the vertical axis (left – right turn) and a horizontal axis which are normal to the camera.

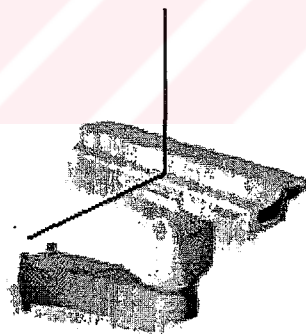


Figure 4-2 The rotation axes of the camera

These rotations are sufficient to move the camera head upward or downward and turn it to left or right which cover the complete space. In each stepper motor, a complete rotation is 200 steps. The details of the hardware interfacing and controlling the camera are given in [8].

4.3 Software architecture: Client-Server Model

As mentioned in the previous section, the system is based on a client-server model. In a client-server model we are providing a service in the server side, which the clients may use by sending request to the server [Sec. 2.6]. This means that, we should have two running programs, one in the client and the other in the server machines. The service provided by the server is controlling camera by turning it to left or right and up or down. Also, the server will provide the client with the images taken by the camera from the scene. On the server side we have four main problems to solve.

- How to send the control commands to the motors rotating the camera,
- how to get status information about the current orientation of camera,
- how to get the image from the camera,
- how to communicate with clients.

First two points are related to the interface of camera and the server computer. This part is completely hardware and operating system dependant. Since we are using linux operating system on an Intel based Pentium processor, we developed some device drivers to read data from and write data to the serial port and control the motors. Also because we are using stepper motors in changing the orientation of the camera, we have a reference point problem to overcome. The device drivers and the solution to reference point problem may be found in [3]. Getting images from the camera also depends on the model and interface of the camera and the computer. For example, for a Webcam which is connected through USB port, the software should consider the resolution of the camera, number of frames per second, handshaking controls, image format, number of colors available, USB interface and so on. All of these parameters may change if we use a different type of camera, so we decided to concentrate on the data transferring part and use the device drivers provided by the manufacturer. This solution has the disadvantage of getting the image from the device driver in a file, and because of that we will have additional I/O overhead.

For data transferring I have used the socket programming interface [Sec. 2.8]. As we want to control the camera through the Internet, it was necessary to use the facilities provided in the TCP/IP protocol. This protocol provides two different types of data communication which are, connection-oriented transmission and connectionless transmission [Sec. 2.4]. The main difference between these two methods is the reliability provided by the connection-oriented method. Since in a control system, reliability is of high importance, we decided to use this type of connection. Also, because the device drivers are simpler to develop in C programming language, we preferred to use this language in writing the server program. Server program, therefore, starts by creating a socket and binding it to the local address and declaring its readiness to accept the requests from the clients. Then it acts as a passive application waiting for the requests from the clients [Sec. 2.7]. The requests have the format of <command, parameters>. For example, to read the current status of the camera, the command part will be 'g' which stands for *get* and to change its direction it will be 'p' which is the first letter of *put*. Parameters in this case are a string consisting of motor number and amount of rotation. Since we are using only two motors, the first number can be either 1 or 2. For the second number we may have four different values. If the motor rotation is going to be very small then this value is either 1 or -1. But if it should rotate quickly, the value is 5 or -5. Each of these values will cause as many steps of rotation as the absolute value of that number. The following list of sample commands will be interpreted as shown in the table below.

<i>Command</i>	<i>Motor</i>	<i>Direction</i>	<i>Steps</i>
<i>P 1 1</i>	<i>First</i>	<i>Clockwise</i>	<i>One</i>
<i>P 1 -1</i>	<i>First</i>	<i>Anti-clockwise</i>	<i>One</i>
<i>P 2 5</i>	<i>Second</i>	<i>Clockwise</i>	<i>Five</i>
<i>P 2 -1</i>	<i>Second</i>	<i>Anti-clockwise</i>	<i>One</i>

Table 2 Sample commands sent to server form a client

The server program has the capability of serving more than one client at the same time. In current implementation, the server can accept up to five connection request from the clients. The server before accepting a request from a client, compares its IP address with a list of addresses which are considered as eligible to enter the system. This step which is added as a security mechanism inhibits all unknown users from entering and using the system. After accepting a request from a client, the server creates a new process to serve the client by calling *fork* system call which is available in *linux/UNIX* operating system. The server assigns the responsibility of responding the requests of that specific client to the newly created child process. The child process will create a new socket for this communication and close its accept facility to avoid any interrupt by other clients. The server, on the other hand, will continue in a loop waiting for other requests from other clients. This model of servicing the clients has been illustrated in figure 2-3. The second responsibility of server application is sending the image to the client whenever it is updated. This operation is being done by another child process. The reason for creating the new child process is handling the control operation in parallel with image transferring job.

The client side should be an application running on the client's machine. The responsibilities of this application are as follows:

- Communicating with user through a graphical and friendly user interface,
- Communicating with server machine and getting or sending necessary data/information,
- Showing the image taken by the camera on client's screen and updating this image whenever it has been changed on the server side.

Similar to server application, here also we are using a socket interface to establish the connection with the server. On the client side the connection is established by creating a socket and then connecting to the server by sending a *connect* request [Sec. 2.7 & Fig 2.3]. our main problem in client side is that if we are going to use the system in the Internet, it would be impossible to install and run client program on the client machines. Also since each client has a different type of

hardware and operating system, the client application will not run on all of them (platform dependency). We tried to solve this problem by means of using some features available in the infrastructure of the Internet and HTTP protocol. The World Wide Web is a huge collection of interconnected hypertext documents on the Internet. A hypertext document is a document that contains hot links to other documents. The Web is based on two standards: the HTTP protocol and the HTML language. HTTP stands for HyperText Transfer Protocol, and describes the way that hypertext documents are fetched over the Internet. HTML is the abbreviation for HyperText Markup Language, and it specifies the layout and linking commands present in the hypertext documents themselves. Resources on the Web are specified with a *Uniform Resource Locator (URL)*. A URL specifies the protocol used to fetch a document as well as its location. The HTTP protocol is implemented in software on the server and on the client machines. The server software is called a Web server or HTTP server and the client software is called a Web browser. To open an HTML document, the Web browser sends an HTTP command to the server requesting the document by its URL. The Web sever responds by sending the HTML document to the client. The client then displays the document on the user's screen. If the HTML document contains graphics, the Web browser makes additional requests for the graphics files to be sent, and then displays the graphics with the text. Much of the power of the World Wide Web stems from its platform independence. But it also has its drawbacks. It is very difficult to extend the Web protocols without leaving many Web users behind. For example, Web content developers are constantly trying to extend the capability of the Web by integrating new types of media, like 3D worlds and animation, but these developers then face the problem of excluding people without those viewing capabilities, which limits their audience. Java has begun to address the protocol problem by using Java applets [Sec 3.3]. An applet is a java program that appears embedded in a Web document, just as a graphics would be. The Java applet runs when it is loaded by a java-enabled Web browser. The running applet then draws itself in the user's browser window according to the programmer's

instructions. This seems to be the best solution to our client side problem. We have developed the client program using Java language and as an applet. The applet is embedded in an HTML file and is loaded together with it. The applet creates a graphical user interface and waits for user to start the connection by pressing on the connect button. Pressing connect button causes the program to send a connect request to the server and then a reliable TCP/IP connection is established. The user interface makes it possible for the user to turn the camera to left or right and up or down by clicking on the arrows shown in figure 4-3 or by using arrow keys. The image sent by the server is displayed on the right hand side. This image is taken from a jpg file on the server computer which is assumed to be created by the device driver of the camera. Whenever the file is updated, the applet fetches it and updates its screen. Similar to the structure used in the server application, a new thread in client side is responsible to check updates in the image file and transfer it to the client machine.

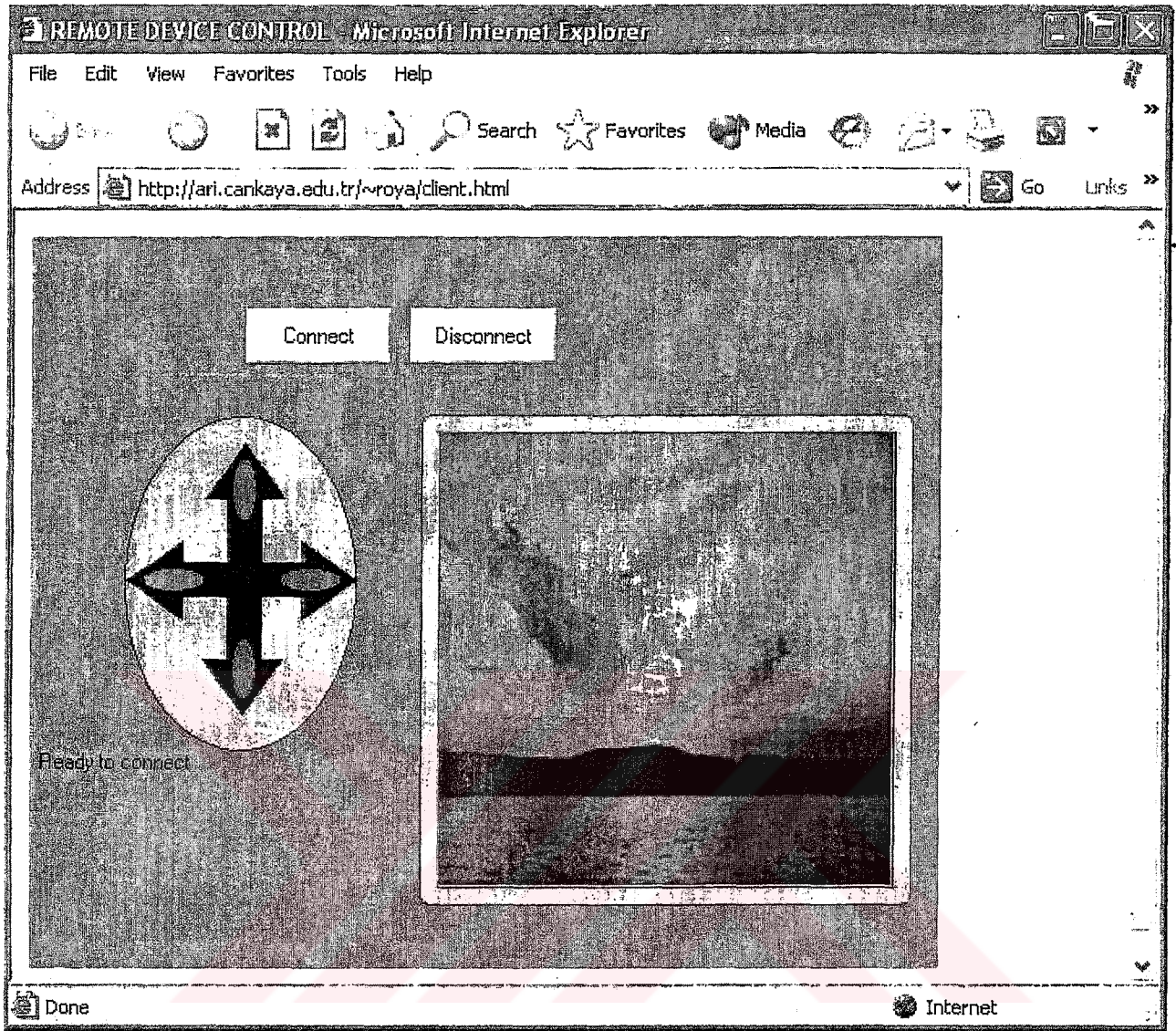


Figure 4-3 User interface of the client program

Chapter 5

Experimental results and discussion

The server program was implemented using C programming language. The client is a Java applet but the connection between these programs established without any problem. The port number used in my application is 8000 but the user is not aware of this value. In my first trial I let the user to enter the port number from a text box but since this number is always constant in my program, hiding it from the user makes it more easy to use and user friendly. Also I tried to establish a security mechanism in the server side by checking the IP addresses of requesting machines and comparing them with an available list. This method can be replaced by a login/account checking one. The disadvantage of using IP addresses in security issues is that if dynamic IP addresses are being used in a LAN the method will not work well. Turning the camera can be done by either clicking on arrows on the screen or by using arrow keys from the keyboard. At the time being the images used in our system are 256 X 256 pixels and the graphics file are in JPG format. For some applications like teleconferencing, a lower resolution together with a lossy compression can give a more realistic and on-line feeling to the users. An improvement to this system can be as follows:

- Instead of using the graphics files created by device drivers, the image can be got directly from the camera. This needs developing some device drivers.
- This system should not be restricted to transferring images. For some applications sound is as important as image.

- At the time being the system has no solution for conflicts resulting from simultaneous commands coming from different clients.
- Security may be enhanced by adding a login/password mechanism.



References:

1. NISHIMURA Kouji , MAEDA Kaori , KOHNO Eitaro , AIBARA Reiji IPSJ
SIGNotes Distributed System and internet Management technology, June 2001
2. H. FUJIWARA, Camera Recorder Control Protocol, Graphics Communication
Laboratories, March 1998.
3. TCP/IP protocol suite, Behrouz A. Forouzan, McGraw-Hill 2000,
ISBN:0-07-116268-2.
4. S. Kondakci, "*UNIX Networking I: TCP/IP Networking*", University of Oslo,
Norway, 1993.
5. Computer Networks, Andrew S. Tanenabum, Prentice-Hall, ISBN:0-13-166836-6.
6. S. Kondakci, "*UNIX Networking II & III: Autonomous Networks and Routing*",
University of Oslo, Norway, 1994.
7. S. Kondakci, Textbook: "*Advanced Unix and Shell Programming*", Scandinavian
University Press, Oslo, Norway, 1991.
8. PC and Embedded Web Server Based Remote Control, Erdem Gokhan Yilmaz, Ms
thesis, Cankaya University, July 2002.
9. Unix system programming, Keith Haviland, Addison-Wesley, 1999,
ISBN: 0-201-87758-9.
10. Advanced java networking, Prashant Sridharan, Prentice-Hall 1997,
ISBN: 0-13-749136-0

Appendix A

Source codes



```

// Server program

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

#define SIZE sizeof(struct sockaddr_in)
#define MAX 3
void catcher(int sig);
int newsockfd;

main()
{
    int sockfd;
    char com, resp[10];
    struct sockaddr_in server = {AF_INET, 8000, INADDR_ANY}, clients;
    char Valid_IPs[MAX]={"192.168.1.1", "192.168.1.2", "192.168.1.3"}
;
    static struct sigaction act;
    act.sa_handler = catcher;
    sigfillset(&(act.sa_mask));
    server.sin_port=htons(8000);
    sigaction(SIGPIPE, &act, NULL);
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
    {
        printf("Error creating SOCKET\n");
        return(0);
    }
    if( bind(sockfd, (struct sockaddr *)&server, SIZE)==-1)
    {
        printf("bind failed\n");
        return(0);
    }
    if( listen(sockfd, 5)==-1)
    {
        printf("listen failed\n");
        return(0);
    }
    while(1)
    {
        newsockfd=accept(sockfd, &clients, SIZE);
        valid=0;
        for(i=0; i<MAX; i++)
            if(strcmp(Valid_IPs, clients.sa_data)==0)
                valid=1;
        if(valid)
        {
            if(fork()==0)

```

```

        for(;;)
        {
            recv(newsockfd,&com,1,0);
            if(com=='g')
            {
                fflush(in);
                fseek(in,0,SEEK_SET);
                fscanf(in,"%s",resp);
                resp[3]=0;
                send(newsockfd,resp,4,0);
            }
            else if(com=='s')
            {
                recv(newsockfd,resp,4,0);
                printf("The command received fr
om client is : %s\n",resp);
            }
            else if(com=='e')
            {
                close(newsockfd);
                exit(0);
            }
        }
    }
}

void catcher(int sig)
{
    close(newsockfd);
    exit(0);
}

```

```

/// Client program
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.awt.Graphics;
public class clnt extends java.applet.Applet
{
    int PORT; // the port num
ber
    String st;
    Socket s;
    TextField txtOut,txtIn,Port; // Text fields
on the screen
    Button sends,getsb,portb; // Buttons
    BufferedReader input; // Input object

```

```

PrintWriter output; // Output Objec
t
InputStream i1; // Input stream
OutputStream o1; // Output Strea
m

public void init() // Initializing
the program
{
    st="Ready to connect";
    setLayout( new GridLayout(3,2,10,10)); // lay out of t
he screen(3 rows and 2 columns)
    Port = new TextField("21",20); // add user int
erface elements
    add(Port);
    portb = new Button("Connect");
    add(portb);
    txtOut = new TextField("txtOut", 20);
    add(txtOut);
    sends = new Button("Send");
    add(sends);
    txtIn = new TextField("txtIn",20);
    add(txtIn);
    getsb = new Button("Read");
    add(getsb);
} // init
public Insets insets() // determine boundaries of
the screen
{
    return new Insets(10,10,210,50);
}
public void paint( Graphics g) // put a message on the s
creen
{
    g.drawString(st,5,300);
}
public void stop() // stop the connection
{
    try
    {
        s.close();
    }
    catch (IOException e) {}
} // stop end
public boolean action(Event e, Object o) // respond to user action
s
{
    String t,s1;
    t=(String)o;
    String host ="cankaya.edu.tr";
    if(e.target instanceof Button)

```

TC YÜKSEKÖĞRETİM KURULU
İNÖNÜ ÜNİVERSİTESİ
İNÖNÜ İKTİSADİ VE İŞLETİM BİLİMLERİ FAKÜLTESİ
İKTİSADİ İSTATİSTİK ANABİLİM DALI
 Host URL

```

    {
        if( t.equals("Connect"))    // if the button pressed is con
nect
        {

            try {
                PORT = Integer.parseInt(Port.getText());
                s = new Socket(host,PORT);    // create new connectio
n
                il=s.getInputStream();    // create the I/O strea
ms
                ol=s.getOutputStream();
                input = new BufferedReader(new InputStreamReader(il
));
                output = new PrintWriter( new OutputStreamWriter(ol
));
                st="Connection successful";
            }
            catch (IOException x) {st="Connection Failed"; }
            repaint();
        }
        else if( t.equals("send"))    // if the button pressed is
send
        {
            String tem1;
            tem1=txtOut.getText();
            output.println(tem1);
        }
        else    // otherwise it is rea
d button
        {
            try{
                s1=input.readLine();
                txtIn.setText(s1);
            }
            catch(IOException ex){st="Error sending data";repaint();}
        }
        return true;
    }
    return false;
} //action end
} //client end

```