

Computer Networks and Network Programming - Lecture Notes

Mehmet Gençer

Contents

1	Digital Communications	2
1.1	Transmission media	3
1.2	Transmission of digits	3
2	Basics of data packet transmission	3
2.1	Exercises	3
3	The TCP/IP protocol architecture	4
3.1	Network access layer	4
3.2	Exercises	5
3.3	Medium access strategies	5
3.4	Exercises	5
3.5	The Internet Protocol (IP)	5
3.6	IP and Local Area Networks (LANs)	5
3.7	Routing and switching equipment and strategies	6
3.8	Exercises	6
3.9	The Domain Name Service (DNS)	6
3.10	Backbone technologies	6
3.11	Nature and control of Internet traffic flow	6
4	User level data exchange: TCP and UDP	7
4.1	Exercises	7
5	Programming primitives: sockets and connections	8
5.1	UDP: Connectionless datagram sockets	8
5.2	TCP: connection oriented sockets	8
6	TCP/UDP addressing and control	9
6.1	Exercises	9
7	The content of networks: Application protocols	9
8	Security mechanisms	10
8.1	Secrecy with symmetric encryption	11
8.2	Secrecy with asymmetric encryption	12
8.3	Authenticity with hash functions	12
9	Applications of security mechanisms	12
9.1	IP security	12
9.2	Digital signatures	13
9.3	TCP security with SSL	13
10	SSL Sockets in Java	13

11 Exercises	14
12 Distributed programming	14
13 Remote Method Invocation (RMI)	15
13.1 RMI Concurrency	16
13.2 RMI over SSL Sockets	16
14 Architectures for distributed programming	16
15 XML-RPC	17
15.1 XMLRPC Introspection	18
15.2 Asynchronous XMLRPC calls	18
A Code examples	21
A.1 UDP Broadcast	21
A.2 TCP Echo Server	23
A.3 TCP Echo Client	24
A.4 UDP Multicast	25
A.5 SSL TCP Echo Server (which does not work!)	27
A.6 SSL TCP Echo Client	27
A.7 SSL TCP Echo Server which works	28
A.8 Simple HTTP-S Server	30
A.9 RMI Example - Interface	33
A.10 RMI Example - Server	33
A.11 RMI Example - Client	34
A.12 RMI Example - SSL Server	36
A.13 XMLRPC Example - The shared object	39
A.14 XMLRPC Example - Server	39
A.15 XMLRPC Example - Client	40
A.16 XMLRPC Example - A Python client	42
A.17 XMLRPC Example - An Asynchronous Client	43

1 Digital Communications

Communication media is analog in nature. In cases such as FM or AM radio, the analog electromagnetic signals correspond somewhat directly to the analog data that is intended to be transferred (i.e. sound of program host or music). Similarly the tracks in a music record correspond to sound waves it is intended to reproduce at its destination. In AM (amplitude modulation) radio broadcast, for example, the amplitude outline of fixed frequency electromagnetic waves carry correspond to the audio signal which is to be transmitted (see http://en.wikipedia.org/wiki/Amplitude_modulation for an animation of how this is done). A wide variety of today's communications systems still use analog data transmission: in addition to radio broadcast, TV broadcast, conventional telephone lines and mobile phones still transmit audio visual data using analog techniques, although supported by some digital techniques such as in the case of TV-teletext and phone dialing.

Analog data and its transmission have some substantial problems. When a music record is deformed or radio broadcast has noise, it is not possible to separate the original data and noise, thus the information is irreversibly lost.

Digital data, in contrast, consists of binary data, i.e. 1s and 0s. Experimentation with transmission of digital data started in mid 20th century, following success of digital computers and as demanded by computer technology. Indeed, the telegraph can be considered as a primitive example of digital transmission, as it relies on the Morse alphabet which uses short and long signals, and pauses in between. Most important advantage of digital content is that even if the analog representation is deteriorated –to a certain level–, the original data can be recovered (Include Figure here). For example if we represent 0s with low voltage and 1s with

high voltage on a pair of cables, the receiving end can understand what is meant even when the high voltage is imprecise (a little higher or lower than usual). As a result the digital data has a higher chance of being received correctly.

In attempt to take advantage of digital data, most analog data today is coded in analog form. Well known examples are digital video and music. Digitization is inherently a reduction, in the sense that precision (commonly referred as resolution) is limited to a certain number of digits. Nevertheless, abundance of today's storage capacity enables us to increase the precision in digitization of analog data to levels sufficient for most use contexts.

1.1 Transmission media

Some means of transmission are air borne, such as radio waves and light. These have the advantage of being already available: one does not need to install any equipment to transfer radio waves. However there are also some substantial disadvantages: (1) transmission range is limited, (2) transmission is broadcast and publicly accessible, hence lacking any secrecy, (3) transmission from different sources overlaps and can impede one another. Despite these advantages wireless radio transmission is used for very short range (e.g. office wireless) and very long range (e.g. satellite) communication.

Second major kind of transmission media is over cables. Although this method requires expensive installation for cabling, it has serious advantages such as having a longer range and ease to provide secrecy. Long range cabling has been used for both cross-continental and national communication backbones. Recent development of fiber-optic cables enabled us to transmit light waves over cables, having a much higher data capacity.

1.2 Transmission of digits

First attempts to transmit a series of bits (digits in binary number system) over analog media encountered problems regarding the clocking of events at the two ends of transmission. Development of techniques such as Manchester coding replaced early solutions based on using separate clock and data lines (see Chapter 5 in Stallings [2] or http://en.wikipedia.org/wiki/Manchester_code).

Despite strength of digit transmission to noise in the transmission media, errors are still possible. For this reason digital transmission techniques usually transfer digits in a certain length packets, where parts of each packet is used to convey information to detect errors in transmission. Error detection techniques vary from use of a single parity bit to more extensive CRC codes (see Section 6.1 in Stallings [2]).

2 Basics of data packet transmission

Most basic prerequisite of digital transmission is detection and correction of errors. For this purpose bits of data are transmitted in fixed or variable size packets, which are commonly referred to as data frames. Each frame has a small part which is used for error detection, commonly by means of CRC codes. When the frame is variable size, it is also necessary to put flags at the beginning and end for demarcation of frames (See HDLC example in Stallings, Chapter 7).

In order to ensure error-free transmission of data frames, the receiver must send acknowledgement of data frames correctly (or not!). The techniques for error correction are usually of two types: stop-and-wait flow control, and sliding-window flow control with piggybacking.

Example: Problem 7.5 from Stallings.

2.1 Exercises

- E-1 Examine a music file of yours. How is it digitized, and what is the resolution of its digitization?
- E-2 Consider a transmission media in which probability of error in each digit's transmission is $\frac{1}{1000}$. What is the probability that a data packet of 1 Kbits is transferred without errors?
- E-3 (Adopted from Stallings, problem 7.12) Two stations communicate via a 1-Mbps satellite link with a propagation delay of 270 ms. The satellite serves merely to retransmit data received from one station

to another, with negligible switching delay. Using data frames of 1024 bits and 3-bit sequence numbers (i.e. sliding-window size of 8 frames), what is the maximum possible data throughput?

E-4 Read Licklider and Taylor's article [1]. Do you agree with the advantages of computer mediated communication in terms of access to distributed intellectual resources? What do you think has changed since the writing of the article?

E-5 Solve problem 7.3 in Stallings.

E-6 Solve problem 7.4 in Stallings.

3 The TCP/IP protocol architecture

Implementing a reliable transmission of data globally, as in the case of today's Internet, requires several features to be implemented starting with crude transmission of bits as we have seen above. For practical purposes, these features build on top of one another and divided into layers in a protocol stack. Most common protocol stack used in practice is TCP/IP (see also Stallings Chapter 2):

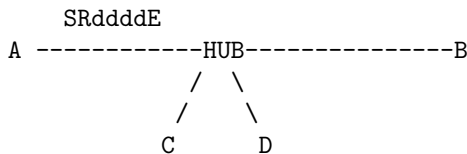
Higher layers

- | TRANSPORT : Reliable transport of data sequences+security
- | INTERNET : Global addressing of data frames+security
- | NETWORK ACCESS: Local addressing of data frames
- V PHYSICAL : Error free transmission of data frames

Lower layers

3.1 Network access layer

The key issue in network access layer is referencing/addressing between network devices that are directly accessible via the communication media. Although network topologies based on one-to-one connections are possible, most common topology in use today is one-to-many broadcast, be it wireless or wired using hub devices. In this scenario it is necessary to use unique identification for sender and intended recipient of a data frame:



In the above example, the content of the data frame includes parts that indicate sender (S) and recipient (R) of data, in addition to data itself (d) and error detection/correction code (E). Although the data frame is received by nodes C and D above, it is discarded by them since they are not the intended recipient. In practice some other information is used such as to indicate data length if data frames are of variable length, and clock synchronization patterns.

The most common form of network access layer in use today is IEEE 802 reference model (See Stallings [2] sections 15.3 and 16.2, its variants (e.g. 802.11) is used for recent wireless office networks) and defined as MAC (medium access control) protocol. The device identifiers used by this standard can be seen at the output of 'ifconfig' command on a Linux system, and actually these identifiers are globally unique as ranges of identifier numbers are shared by device producers around the globe. In wireless, bluetooth, and GSM technologies, similar unique identifiers are used.

The so called Ethernet protocol stack defines an additional protocol layer on top of MAC, which is called LLC(Logical Link Control). The key virtue of LLC is to provide an additional addressing so that different higher level protocols can be used on top of MAC/LLC. Two important such protocols are IP (the Internet Protocol) and ARP (Address Resolution Protocol) which we will see later.

3.2 Exercises

E-1 In problem Stallings 7.3, assume that 72 bits of the 1000 bit frame is used for protocol overhead. What is the net data rate?

3.3 Medium access strategies

The description of the network layer above is almost implying an uncoordinated access to communication medium. When multiple sources transmit signals, all of them would become incomprehensible. In reality, both coordinated and uncoordinated access strategies have been tried. For example the token ring protocol imposes that each party in a circular arrangement waits for their turn to use the medium. However this strategy results in waste of time when most of parties do not need the medium.

On the other hand uncoordinated strategies based on collision detection contention are widely used. In this strategy, stations attempt to transmit data if the media is free. But if there's a collision, they'll wait a -random- while, and retry. The technique works well for low-medium load of the channel (See Stallings Section 16.2).

3.4 Exercises

E-1 Solve problem Stallings 16.3

3.5 The Internet Protocol (IP)

The network layer enables us to clarify source and destination of data packets when there are multiple sources/recipients sharing a communication media and are directly accessible. The aim of the Internet (inter-network) layer is to provide global addressing which is used in delivery of data to recipients which are not directly accessible through the communication media, but rather should be delivered by help of intermediaries. In other words, Internet is a network of networks, each of which have only partial information about it.

The general strategy is called packet switching, in contrast to line switching which is used in cable phone networks. Global addressing requires intermediaries which has the necessary knowledge so that the data packet can be delivered through communication lines in a way that it incrementally gets closer to its destination. Therefore it requires so called 'routers', which play the role of postman in regular mail system.

The Internet protocol version 4 is described in detail in Stallings Section 18.4. Most details are irrelevant for our purposes, except a few:

- Internet addresses in version is 32 bit (hence there's a shortage these days, which is being addressed in version 6).
- There's no guarantee of packet delivery (this is left to transport control protocols in the above layer). Time-to-live (TTL) field in IPv4 frames is used for packet timeout to avoid packets travelling forever and occupying the network due to various unpredictable reasons.
- IPv4 was later extended to allow security (encryption) measures to be used to create VPNs. These will be covered later.

The IPv4, other versions of IP, along with many Internet related protocols are managed by the IETF(Internet Engineering Task Force).

IANA is the authority which coordinates allocation of IP addresses (along other Inter-network related allocations). Some of the address range is special such as those used for multi-casting.

3.6 IP and Local Area Networks (LANs)

IP is also used in corporate local area networks (LANs) as the layer above the network layer, even if the packets are not intended to go out of, say, a campus. For example to connect several network segments together. Special ranges of IP address space are allocated for LAN usage such as 192.168.x.x or 10.x.x.x. In addition use of a special broadcast address 0.0.0.0 is allowed in LANs, for the purposes of address and configuration discovery.

Configuration discovery When computers are turned on they mostly use DHCP(Dynamic Host Configuration Protocol) to obtain which IP address to use, what is the gateway (local router), and DNS service provider. To do that they send a broadcast packet.

Address discovery To learn which MAC is associated with which IP, LAN computers use ARP (Address Resolution Protocol) protocol in which they send broadcast packets.

3.7 Routing and switching equipment and strategies

Each LAN needs a router (like a post office) to connect to the global network. These routers have an idea about which of the outgoing connections is the likely shortest path to destination of the packet. Also since Internet is a dynamic and ever changing environment they need to review their ideas by talking to each other, and learning about which neighboring routers are malfunctioning or busy. For this purpose they use a different protocol than IP, the ICMP (Internet Control and Messagin Protocol), hence the reason for existence of SSAP/DSAP in LLC protocol! The ICMP is what you use when you issue a ping command on Linux.

Simplest technique for routing is flooding, in which a packet is sent to all possible directions. It is inefficient since too many copies circulate in the network. However it is used in rare cases (e.g. malfunction or high traffic load announcements in ICMP) since at least one packet will reach the destination through the shortest path. For most regular deliveries, adaptive strategies is used, in which routers occasionally conduct delay measures of their connections to update their routing preferences.

In larger LANs, smart equipment such as Layer 2 and 3 switches are used. These devices, unlike hubs, examine data frames and deliver copies of packets to only the recipient's connection (cable) which they also remember from experience.

3.8 Exercises

- E-1 Use the ping command on Linux to discover the shortest path between your computer and, for example, google.com.
- E-2 If branching factor is b in a network, and TTL value is 10, how many of copies of a packet would be created when it is routed using flooding strategy?

3.9 The Domain Name Service (DNS)

DNS is necessary if you want to use domain names (such as `www.bilgi.edu.tr`) instead of IP addresses, which are hard to remember. In early days of Internet, DNS service consisted of merely getting a file now and then from a central authority which provides a mapping of domain names to IP addresses. But as Internet has grown, a distributed database become a necessity. In modern DNS service, users query their designated DNS provider, which in turn queries a hierarchy of authorities (regional, national, global, etc.) to find out the answer.

3.10 Backbone technologies

Different protocols, such as ATM, is used for high speed backbone networks, but are essentially beyond the scope of this course. In essence most such protocols are based on fixed length data packets (in contrast to variable length IP packets) so that very fast and high capacity routing electronics can be used.

3.11 Nature and control of Internet traffic flow

Internet is a chaotic network by nature, and IP is the basis of all user level traffic flow (in addition to backbone protocols and ICMP for the service provider level flow) as seen in Figure 1. IP protocol provides an important mechanism to prioritize data flow, much like priority regular mail. Tye type of service field in IP protocol header can be used to indicate highly important type of data and request priority for its delivery. This mechanism is used frequently for services such as VoIP or multimedia streams.

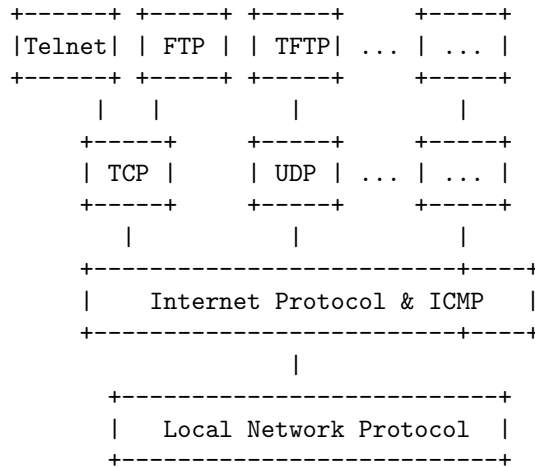


Figure 1: Protocol Relationships (Source RFC791)

4 User level data exchange: TCP and UDP

All kinds of Internet traffic goes in IP packets. This includes anything from e-mail to web flows. Between all these specialized application protocols and the IP are the two sister protocols: TCP(Transmission Control Protocol) and UDP(User Datagram Protocol), as seen in Figure 1. 1. The TCP and UDP provide three important services on top of IP:

- By providing a means of multiplexing data flow into different ‘ports’ on an Internet endpoint, they allow computers to differentiate flows related to different applications at the same IP address. So that, for example, an endpoint computer can differentiate flows for e-mail programs (SMTP, IMAP, POP, etc.) and web browser programs (HTTP, HTTPS, etc.).
- They differentiate the connection oriented (TCP) and connection-less(UDP) applications. Some types of applications require a long lasting connection to do their job (e.g. download of a large file), but for some others exchange of one packet is sufficient (e.g. DNS or DHCP services).
- For connection oriented services, the TCP provides sequencing and completion of data at the endpoint, since IP has no delivery guarantees. TCP makes sure that packets arriving at different speeds are put in the correct sequence before they are delivered to the user level application, and requests missing packets from the other endpoint. At the end a stream of data is received by the target user process in the order it is sent by the source user process.

The TCP frame structure is shown in Figure 2. Very much like the network layer protocols (e.g. MAC), the TCP protocol uses sliding-window flow control (by means of ‘sequence number’ and ‘acknowledgement number’ fields in the frame). The 16 bit source and destination port addresses allow a wide range of user level applications to be implemented. Many of the port addresses are reserved by IANA for common protocols. For example e-mail transfer protocol (SMTP) uses port 25, whereas web traffic (HTTP) goes to port 80.

UDP is a much simpler protocol. All it does is to add port addressing capability on top of IP.

4.1 Exercises

- E-1 On Unix/Linux computers, examine the output of ‘ifconfig’ command. What is ‘lo’(loopback) interface for?
- E-2 Do a research on firewalls, and specifically the ‘iptables’, default firewall infrastructure in Linux. What are the key services of a firewall in a corporate setting?

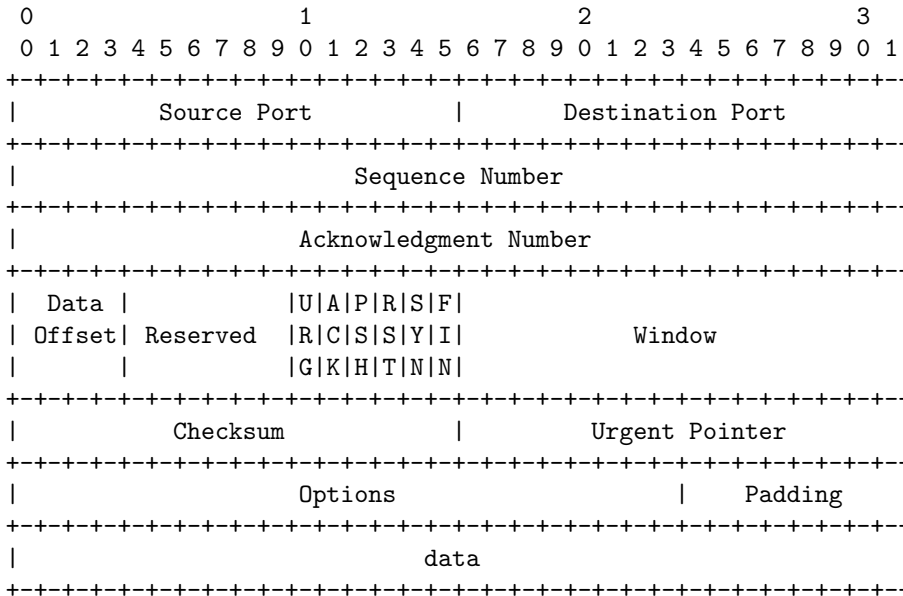


Figure 2: The TCP protocol frame structure (Source RFC 793)

5 Programming primitives: sockets and connections

Most network programming library implementations in use today has evolved from Berkeley IPC (Inter Process Communication) library which was the reference implementation of TCP/IP stack. The term ‘socket’ was also coined then to refer to any endpoint in a TCP or UDP communication. Most commonly, a socket is ‘bound’ to a network interface (one of many in your computer), but –following Berkeley IPC– a socket can also be used for communication between threads of a process, or independent processes on the same computer. For the latter case one can either use the so called UNIX sockets, or a special purpose virtual network interface called ‘localhost’ (with a designated IP address of 127.0.0.1). For the purposes of simplicity we will use ‘bound’ sockets, be them use localhost or actual Internet network interfaces. Most network programming material available also cosiders only these type of sockets (e.g. <http://java.sun.com/docs/books/tutorial/networking/index.html>).

5.1 UDP: Connectionless datagram sockets

The case of UDP is relatively simpler. A datagram socket (the DatagramSocket class in java.net library) can be bind to a port of a network interface if it is free (which is kept track of by the operating system). Once bound it can send/receive datagrams to/from any other socket. The length of datagram is system dependent but not recommended to be more than 1500 bytes.

The code example A.1 demonstrates how Java datagram sockets can be used to send broadcast datagrams. Since it uses broadcast adress, the packets can be received by all peers listening in the LAN.

5.2 TCP: connection oriented sockets

Since TCP is a connection oriented protocol, one must establish a connection before exchanging data. The notion of connection inherently imposes an asymmetry in TCP because one peer is the client (the one who initiates connection) and the other is server (the one who awaits for connections). A TCP client connection can easily be established using the ‘telnet’ program (e.g. try to connect to a web server).

Because of the asymmetry in connection oriented communication, TCP sockets are of two kind: server and client sockets. The corresponding classes in java.net library are ServerSocket and Socket. It usually does not matter where the client socket is bound, normally the operating system allocates a free local socket to

your process. However it is possible to bind the client socket to a specific port if necessary (for example if firewalls only allow certain ports).

The TCP client socket has a single connection which is full duplex, ie. it is a two way stream. However the server sockets are designed to communicate with multiple clients concurrently, therefore a connection is created for each accepted client. The connection created upon accepting a client is the instance of the Socket class used by the client. The code examples A.2 and A.3 demonstrate how java libraries are used for TCP connections. The example server program ignores the concurrency issue and can handle only a single client at once.

Homework assignment II (Work in teams of two)

1. Write a TCP server and client to play a music file from the server at the client.
2. Write a UDP broadcast server and client to play a music file at the server on all clients.

Since there is no delivery guarantee for UDP you may need to prefer different music encodings for the UDP case. Report any problems and solutions found.

6 TCP/UDP addressing and control

Interface information Information about network interfaces in the system can be retrieved using the appropriate classes in java.net package. The NetworkInterface class can be used for this purpose. See the code examples from Sun at <http://java.sun.com/docs/books/tutorial/networking/nifs/index.html>.

Unicast, Broadcast, Multicast Since TCP requires connection establishment, it is inherently unicast: data is sent to a single recipient. UDP can use broadcast and certain protocols rely on it heavily (e.g. DHCP, DHCP-PXEboot, ARP)

Nevertheless IP protocol and address space was designed to support multicast although it is not available throughout the Internet. However applications like IPTV is creating a synergy for wider implementation and applications are becoming interesting recently. Multicasting is extremely efficient compared to multiple unicast strategy but is demanding on the routers (see section 19.1 in Stallings). Specifically it requires clients to join multicast groups and routers must remember these registrations to send copies of data packets accordingly. A related protocol IGMP (Internet Group Management Protocol) supports the working of multicast at the LAN level. In general UDP multicast is becoming available in office LANs and metropolitan area networks.

The example code A.4 shows the multicast version of the UDP broadcast program. The difference lies in the usage of MulticastSocket on the client side instead of DatagramSocket, and an address from IP multicast range. The server side of the program is almost the same with broadcast version, but the client side requires joining and leaving a multicast group.

6.1 Exercises

- E-1 Consider a network structure with a branching factor of two (i.e. each router has three interfaces), and that a TV broadcast is being delivered across n routers to 2^n recipients. Compare the multicast and multiple-unicast traffic.

7 The content of networks: Application protocols

Each application has its peculiar requirements. Nevertheless we offer a broad taxonomy here. An application must choose to rely on either a connectionless(UDP) or connection oriented(TCP) transport layer. Exceptions are those which use the IP or lower layer protocols in the TCP/IP stack such as ARP, DHCP, ICMP, etc. For the purposes of current subject, we do not discuss those exceptions here.

Following are some dichotomies on the basis of which application protocols can be classified:

Orientation :The application itself may be episodic and as a consequence connectionless. For example HTTP interaction consists of a request and its response. However when the response is too long for the packet infrastructure, the application protocol chooses to use a connection oriented protocol, contrary to its nature, as in the case of HTTP. Conversely the application layer may choose to use UDP since it has little protocol overhead. For example TFTP(Trivial File Transfer Protocol) is similar to HTTP in the sense that it transfer rather long files. But to allow its use with simple systems (e.g. BIOS level PXE/BOOTP systems) TFTP uses UDP, and does a simple sequencing itself.

Architecture : Most applications are based on server/client architecture which is asymmetric. Server and client roles are quite different in such applications. Others are symmetric and do not rely on a single server. The latter is becoming popular as reliance on a server is also a weakness. For example peer-to-peer networks consist of a single type of entities (peers) where there is no server, and can continue to work when one or more peers malfunction. Server /client systems require a server to be alive at a certain IP address and port to function.

Content : Some protocol content is human readable (and hence writeable). HTTP, SMTP and POP are such examples. Others use a binary format which is not human readable. HTTPS, SMTPS, VoIP, FTP, DNS, RPC are such examples. Furthermore, some protocols may be programming language specific. Especially are those intended for distributed programming such as Java-RMI, which contain serialized Java objects, and hence cannot be decoded by a program written in another programming language. However, such specificity is generally limiting and disadvantageous from a software engineering standpoint.

Many of the application protocols for the Internet applications are maintained by IETF, while others come from organizations like W3C. A common practice in the professional community is the circulation of RFC (Request For Comment) documents to stimulate a stakeholder consensus in the computing industry. For example <http://tools.ietf.org/html/rfc2616> describes the current version of HTTP protocol, and as one may see it has matured through several steps and previous RFCs in years. Changes in an application protocol is not trivial as, in time, other application protocols rely on existing ones. This is demonstrated in research literature, see for example <http://cs.bilgi.edu.tr/~mgencer/pub/OSSpaper.pdf>.

Homework Assignment III We want to write a network application for tele-conferencing. In this application users will be able to draw on a shared virtual whiteboard, and send single line text messages to others. No authorization or secrecy criteria is considered for this simple application, i.e. everyone can join a teleconference anonymously and content can be send unencrypted.

- Choose orientation, architecture, and content type for this application protocol.
- Design the types and contents of packets to use.
- (BONUS) Redo the design for the case that also video and audio will be available in the tele-conference.

8 Security mechanisms

Security problems encountered in Internet communication are to a certain extent similar to those with immobile data. Essentially these problems fall into two generic categories:

Content authenticity : One needs to ensure that a content is authentic, i.e. it is not changed by an unauthorized party. This problem is somewhat equivalent to authenticity of handwriting and signature that ensures the source of a document in non-digital documents. Solution to this problem requires what is generally called digital signatures. Digital signatures do not prevent access to contents of a document but only allow us to determine whether the content is changed by an unauthorized party.

Content secrecy : In certain cases one wants to prevent access to contents of a document except by those who are authorized. Solution of the problem is always based on encryption of the content. The solution prevents access to contents of the document by unauthorized parties.

In addition to these problems, Internet communication have some of its own since Internet is an inherently insecure environment:

Traffic analysis : Other than revelation of contents of data packets, passive analysis of data flows can reveal communication patterns which can then be used to understand the nature of communication taking place.

Masquerading : An active threat in which a third party tries to pretend to be a different entity, for example by replaying a data packet which contains a valid authentication sequence.

Denial of Service (DoS) : A third party may prevent the normal and legitimate use of an Internet service. This is often made by sending an excess number of requests to a service so that legitimate users cannot receive timely and reliable service any more.

Here we will first cover three set of techniques for securing content, then continue to how these can be used to remedy the problems above. Two of these teachniques are for encryption of contents, and one for authentication of content.

8.1 Secrecy with symmetric encryption

Encryption techniques (Greek *kryptos*: hidden) have been used for several millenia, especially in military communication. Well known examples date back to Caesar's alphabet shift method, Spartan's cylinder wrap method, etc. All symmetric encryption techniques are based on a secret which is known to sender and recipient only. A simple example is encryption using XOR function and a shared key. For example:

```
SHARED KEY:          01010101
DATA      :          11110000
ENCRYPTION: XOR data with key
              10100101
DECRYPTION: XOR encrypted data with key again
              11110000
```

An obvious weakness of these techniques is that when the key is revealed, secrecy fails, and the key must be replaced and -re-shared with desired parties, which is a security problem in itself. This is called the key-exchange problem, which we will see in the next section.

Another important weakness of all the above encryption techniques lies in the fact that a symbol is always results in the same symbol in encryption. As a result of this, the ciphers can be easily broken using *cryptanalysis*. For example if one finds the frequency of symbols in the encrypted message, one can easily find corresponding data for most frequent letters ('e' in English) or common words (the, a, etc.). The cryptanalysis techniques date back to House of Wisdom scholars in 8th century Baghdad and specifically Al-Kindi and Al-Khawarizmi.

Second generation of symmetric encryption techniques aimed to prevent such cryptanalysis. A well known example is the Vigenere cipher (after Blaise de Vigenère). Vigenere cipher uses a poly-alphabetic method and a secret key such that each letter of the original message is translated using an alphabet chosen with respect to corresponding letter of the -repeated- key (see Wikipedia entry : http://en.wikipedia.org/wiki/Vigen\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{e\global\mathchardef\accent@spacefactor\spacefactor}\accent18e\egroup\spacefactor\accent@spacefactorre_cipher). Vigenere cipher is an example of block ciphers where encryption was cyclic not after a single letter but after the length of selected secret key.

Vigenere cipher is much more resistant to cryptanalysis compared to single alphabet ciphers. The Enigma cipher used by Germans during the World War II was quite similar. However cryptanalysis has developed in parallel to cryptographic techniques to break block ciphers. Alan Turing and his team was able to break the Enigma code using the slow electro-mechanical computers they have built during mid 1940s.

Modern symmetric encryption systems use binary data and secret keys with similar encryption methods. A standard called DES (Data Encryption Standard) was introduced in 1977 which used 56-bit keys. In 1998 Electronic Frontier Foundation was able to break the DES (using a machine which cost under \$250.000!). Starting with 1997 another symmetric block cipher, called AES (Advanced Encryption Standard) was designed which uses up to 256-bit long keys.

8.2 Secrecy with asymmetric encryption

When searching for a solution to the key-exchange problem, two researchers, Whitfield Diffie and Martin Hellman, invented a method in 1976 to establish a secret key using an insecure communication channel. The so called Diffie-Hellman (DH) key exchange algorithm not only solved the key exchange problem, but soon it would be realized that it allowed a whole new generation of asymmetric encryption techniques which does not require a key to be shared at all!

The DH method relies on the computational difficulty of computing reverse discrete logarithms. Let us first see how key exchange works, then consider its strength. If two parties, Alice and Bob wants to share a key, here are the steps in DH method:

Alice	Bob
Choose p, g (p is prime and g is a primitive root of p).	
Choose an a	
Compute $A = g^a \bmod p$	Send p, g , and $A \rightarrow$
	Choose b
	Compute $B = g^b \bmod p$
Compute $K = B^a \bmod p$	\leftarrow Send B
$K = g^{ab} \bmod p$	Compute $K = A^b \bmod p$
	$K = g^{ba} \bmod p$

Now let's turn to the question: how hard it is to find a given p, g , and A ?

Since g is chosen as a primitive root of p and p is prime, g can be any of the numbers $2, 3, \dots, p-1$. For any value A there are infinitely many values, a , such that $A = g^a \bmod p$. For example if $g^k \bmod p = 1$, then $g^{a+nk} \bmod p$ are all equal to A , for any value of n . As a result discrete logarithm is a computationally infeasible way of finding a given p, g , and A .

The same theoretical basis has lead to RSA asymmetric encryption algorithm. This (and others in a family of methods) is also called public-key encryption. Let's consider that Alice chooses two values k and K such that $x^{kK} \bmod p = 1$ for any x^1 . Then the public key K, p is revealed to anyone who requests it, whereas the private key k is kept secret. When Alice wants to send message, m , the encryption is done using $M = m^k \bmod p$. Any recipient can find m by reversing the operation as: $m = M^K \bmod p = m^{kK} \bmod p = m$.

Please note that the method is symmetric although the keys are not. For example one can send a message to Alice by encrypting first with the public key. As a result communication using Alice's public-private keys is secret inwards Alice, but not secret outwards her.

RSA is computationally costly to apply. For this reason it is commonly used to secure the process of exchanging a key for symmetric encryption. For example SSL standard uses this method.

8.3 Authenticity with hash functions

Hash functions are a family of functions which create a fixed length digest of some variable length data. An example, CRC, is used in all layers of TCP/IP protocol stack. Hash functions are developed so that they are sensitive to small changes in the data to ensure reliability.

9 Applications of security mechanisms

Encryption methods can be used to conceal data in place, i.e. when stored in a file. Network applications are not much different.

9.1 IP security

IP protocol was later enhanced to carry encrypted data, using symmetric encryption. This is done in one of two ways:

- Only the payload (data content) of IP packet is encrypted. In this way contents of packets is not revealed. But since IP headers (source and recipient addresses) are in clear, it is vulnerable to packet replay and traffic analysis.

¹Euler's theorem

- **Tunneling:** IP tunneling means encrypting a whole IP packet as a payload to an outer packet. This is commonly used for data flow between branch offices using VPN (Virtual Private Network). It has several advantages: the actual recipient and sender addresses are not revealed, and traffic flow can belong to a LAN which is distributed over Internet (since inner addresses can be LAN addresses). The method is much more resistant to both traffic analysis and replay attacks.

9.2 Digital signatures

Authenticity of data can be ensured in several ways without concealing its contents, hence reducing the computational overhead when this is acceptable. One way of doing this is to attach a hash of message+some-secret-key. Such a procedure cannot be repeated by others who does not have the secret key. Another method is to append a hash of message which is encrypted using public key encryption.

9.3 TCP security with SSL

SSL (Secure Sockets Layer) is a standard protocol on top of TCP which negotiates an encryption mechanism and, if necessary, a secret key before a TCP session starts. To prevent fraud, SSL protocol relies on security certificates. SSL certificates ensure that a public key used for asymmetric encryption with someone actually belongs to them. For SSL certificates it is necessary that a trusted authority digitally signs the identity and public key of someone, using public key encryption with the trusted authority's keys.

The `openssl` program suite on Linux can help manage SSL certificates, or establish SSL connections. For example:

```
openssl s_client -host cs.bilgi.edu.tr -port 443
openssl genrsa
```

10 SSL Sockets in Java

Creation and usage of SSL server and client sockets is not much different from unsecure sockets. The example programs A.5 and A.6 show some examples. However that SSL server does not work. You will receive an error message saying:

```
javax.net.ssl.SSLException: No available certificate or key
corresponds to the SSL cipher suites which are enabled.
```

The tedious issue with SSL server sockets is management of security certificates. To run an SSL server we need to create some SSL certificates and modify the server program above. TO create certificates, one can use the 'keytool' program that comes with Sun Java SDK:

```
/opt/jdk1.6.0_12/bin/keytool -genkey -keystore serverkeys -keyalg rsa -alias mehmet
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: Mehmet Gencer
What is the name of your organizational unit?
[Unknown]: Department of Computer Science
What is the name of your organization?
[Unknown]: Istanbul Bilgi University
What is the name of your City or Locality?
[Unknown]: Istanbul
What is the name of your State or Province?
[Unknown]: Istanbul
What is the two-letter country code for this unit?
[Unknown]: TR
```

```
Is CN=Mehmet Gencer, OU=Department of Computer Science, O=Istanbul Bilgi University, L=Istanbul, ST=Ista
```

```
[no]: yes
```

```
Enter key password for <mehmet>  
(RETURN if same as keystore password):
```

I have created the above certificate which is stored in the keystore file “serverkeys”, using a password “foobar” for both the –only– certificate and the keystore. You can later export the certificate using the command:

```
keytool -export -keystore serverkeys -alias mehmet -file server.cert
```

The improved SSL Echo server code in A.7 works with this created certificate. When this serve is running if you try to connect to it using open ssl client you can do so:

```
openssl s_client localhost 8000
```

However if you try to use the program in A.6 you will receive a security exception. The certificate we have created is not authorized by any trusted certificate authority! Java libraries deny connecting to such servers by default, unlike the simple openssl program. This is essentially what happens when your web browser encounters an untrusted certificate.

The only way to remedy this situation is to add the certificate as a trusted certificate to Java VM. This can be done using the exported certificate, and once again the keytool:

```
keytool -import -keystore trustedcerts -alias mehmet -file server.cert
```

Now if we run the SSLclient this time by the following modified VM:

```
java -Djavax.net.ssl.trustStore=trustedcerts SSLclient localhost 8000
```

The program will trust the certificate received, and continue to its work.

A final example of SSL server sockets is shown in A.8

11 Exercises

- E-2 Propose two different methods so that a user can login to a remote server using a login name and a preset password without sending the password in clear form or using any sort of encryption.
- E-3 Is your proposal(s) to the problem above resistant to packet replay attacks? Can you propose an improvement?
- E-4 Propose a scenario so that two parties, A and B, can communicate secretly without having a shared key beforehand. Assume that symmetric or public-key encryption/decryption is available.

12 Distributed programming

With the increased availability of low cost computers, there is a strong motivation for distributing computationally intensive tasks over multiple computers. Such tasks include industrial problems such as ray-trace computer animation rendering, to scientific problems such as protein folding or cryptanalysis.

While memory sharing is possible in multiple threads working on the same hardware, distributed computing requires different techniques and involves different risks. Some methods for distributed computing are as follows:

Distributed RAM : A recent family of technologies which provides access to RAM on other machines on a local area network using high-speed network connections and hardware supported mechanisms. In principle D-RAM allows threads on different computers to share raw memory. Technologies under different names such as RDMA(Remote Direct Memory Access), DSM (Distributed Shared Memory), has appeared in recent years which provide similar services.

Network Attached Memory : NAM is somewhat similar to D-RAM except that it is implemented at the user software level, rather than hardware. Because of this NAM can be implemented across great distances, but has lower performance. Despite the performance drawback, however, NAM can be embedded into the programming environment, hence allowing remote threads to transparently share objects, rather than raw memory. A well known example is Terracotta system for Java (See <http://www.terracotta.org/>).

Remote Method Invocation : A traditional and robust method for delegating tasks to a remote computer is calling/invoking a method on the remote computer. Various names is used for this general technique. In non-object oriented languages, the Remote Procedure Call (RPC) method has been used for years. Since actions on Java objects are called methods, it is named as Remote Method Invocation (RMI) in Java. Both methods rely on a communication language to indicate name of remote procedure/object-method and its parameters, and to receive the return value. A recent standard, XML-RPC, allows language independent communication hence allowing different programming languages to be mixed in a distributed application, with the cost of lacking the concept of objects.

Message Passing : Method invocation involves calling a method and waiting for it to return a value. Message passing, on the other hand, is a much simpler task since it is only one way. The industry standard, Message Passing Interface, allows not only one-to-one (unicast), but also one-to-many (broadcast) messaging.

Despite its promise, distributed involves certain risks different from multi-threaded programming:

- Performance distributed process is directly effected by the performance of underlying network. A well designed monitoring and adaptive network traffic management is necessary.
- Malfunctioning of remote processes are hard to detect and recover from. Task sharing and performance monitoring mechanisms are required for the distributed process to be fault tolerant.
- Data security risks must be accounted for and their solution may require encryption/decryption techniques which are computationally intensive themselves. Task sharing must be designed to minimize data transfer needs to remedy the problem. Furthermore, unauthorized or malicious procedure/method invocation is also a problem which needs to be addressed in distributed application security.

13 Remote Method Invocation (RMI)

Sun Java SDK provides a library (`java.rmi`) to enable distributed programming, for which extensive tutorial and reference material available at <http://java.sun.com/javase/6/docs/technotes/guides/rmi/index.html>.

Invoking a method of an object which exists in a remote process requires some conditions to be satisfied:

1. Caller and callee must agree on the name of method, list and data type of parameters, and return value type.
2. If the remote method creates an exception, there must be a way to pass the exception relation information back.
3. Both parameters and return value should be suitable for object serialization so that they can be send through a network connection.
4. A certain network socket must be arranged, if necessary with security precautions such as SSL (Secure Sockets Layer).

To satisfy the first condition above, a common method in distributed Java applications is to use an interface definition. Java RMI dictates that such interface definitions extend the Remote interface in `java.rmi` library. An example interface is given in code example A.9. The interface is similar to primality checking and factorization programs we have used previously. The method definitions in the interface uses a special exception class, `RemoteException`, from `java.rmi` library, as a solution to the second problem in our list.

Since all parameter and return value types are already serializable, we are not concerned with the object serialization problem for this example. But if they were instances of classes we have written, we must make sure the class implements Serializable interface.

The interface definition must be shared by the RMI server and client program in order for them to communicate using a common language. The server and client implementations in code examples A.10 and A.11 both use the defined interface. The class in the server program implements the RMIExampleInterface. An instance of the class is registered and published using the UnicastRemoteObject class from java.rmi.server library. This class provides the infrastructure with which the created class instance can be attached to a suitable network socket and called using RMI protocol. This class also provides constructors suitable for using a non-standard network port or security enabled (SSL) sockets for communication. We skip these features for the moment.

Before we can run any RMI server or client program, an instance of ‘rmiregistry’ program must be running on the system. This program extends capabilities of Java VMs to find and use RMI servers. Simply running the ‘rmiregistry’ program satisfies this condition. Once the program is running the server program can be started. The server program finds the running RMI registry process on the system using LocateRegistry class.

The client program, similar to the server, first finds the RMI registry process on the host computer. After that the named object on the RMI service is located using registry.lookup(). Once the remote object is obtained, as can be seen in the client program, the object can be used much like a local object.

13.1 RMI Concurrency

Java RMI subsystem has a concurrent implementation of its network services. It creates threads automatically as client requests arrive. Therefore it is necessary to implement mutual exclusion in objects shared via RMI.

13.2 RMI over SSL Sockets

It is possible to replace sockets used by RMI. One possible use is using SSL for these sockets, while other implementations for compression, etc., are possible. The code example A.12 demonstrates an example implementation using SSL. Please note that no change is necessary for clients. The RMI-SSL example implements the shared interface by extending the UnicastRemoteObject and changing the underlying sockets by calling its constructor. Because of this change it is no longer necessary to export the shared object via UnicastRemoteObject.exportObject().

14 Architectures for distributed programming

The fact that one side of remote execution is passive leads to asymmetric server/client architecture in distributed programs. The term suggests a single server and multiple clients. However, realization of distributed execution commonly requires multiple servers and a single client. For this reason it would be more sensible to call this architecture as coordinator/servant. In this architecture it is imperative that the coordinator process be a multi-threaded one, since it must control multiple servants concurrently.

However, other architectures are possible. For example if servants are pro-actively requesting tasks instead of passively waiting for requests, the coordinator/servant architecture can be reversed. But one must keep in mind that delivery of results poses a problem in this situation.

Both types of architectures are sensitive to failures of the central host. More recently, cloud computing systems using un-centralized (or peer-to-peer) communication is being investigated as they are resistant to failures.

Regardless of the architecture, content secrecy and authorization issues must be addressed in data transport and access control mechanisms, respectively.

Homework Assignment IV Consider the problem of establishing primality of an integer. The problem can be broken into independent problems of finding factors of the integer in distinct ranges. Implement a Java program using RMI with SSL sockets which speeds up the process using several servants. Your

implementation of servants must require users to authenticate using a login and password before they can proceed to submit tasks.

NOTES:

- Use BigInteger class to represent integers.
- Consider using thread pools for coordinator program, with Callable interface.

15 XML-RPC

XML-RPC (Remote Procedure Call) is a language-independent standard protocol for executing remote procedures. It lacks object orientation of Java-specific RMI technology, but nevertheless provides similar capabilities.

The design priority in XMLRPC has always been language-independence. The standard is described in <http://www.xmlrpc.com/spec>. XML-RPC calls and responses are converted into XML coded language independent form as in the following example:

```
##### REQUEST #####
POST /RPC2 HTTP/1.0
User-Agent: Mozilla/1.0 (Ubuntu)
Host: cs.bilgi.edu.tr
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>example.factorize</methodName>
  <params>
    <param>
      <value><i4>12</i4></value>
    </param>
  </params>
</methodCall>
##### RESPONSE (HTTP Headers omitted)#####
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <array>
        <data>
          <value><i4>3</i4></value>
          <value><i4>2</i4></value>
          <value><i4>2</i4></value>
        </data>
      </array>
    </param>
  </params>
</methodResponse>
##### ERROR RESPONSE (HTTP Headers omitted)#####
<?xml version="1.0"?>
<methodResponse>
  <fault>
    <value>
      <struct>
        <member>
```

```

        <name>faultCode</name>
        <value><int>4</int></value>
    </member>
</member>
    <name>faultString</name>
    <value><string>Too many parameters.</string>
    </value>
</member>
</struct>
</value>
</fault>
</methodResponse>

```

XMLRPC standard defines a limited, but nevertheless sufficient set of data types. Any *value* can be one of:

- `int`: four-byte signed integer.
- `boolean`: true/false value.
- `double`: double precision, signed floating point number.
- `String`: a text string.
- `dateTime.iso8601`: date/time, e.g. "19980717T14:08:55"
- `base64`: base64 encoded binary data. This is useful if the data is a binary content such as a picture, etc.

These values can appear alone, or as part of *array* or *struct* constructs, as shown in the examples above. If different languages needs to be supported in strings, the character set encoding can be identified in XML tag. However since default for the Apache Java XMLRPC library is UTF8, this is rarely necessary.

Since XML-RPC data is in textual form, it is transported easily over HTTP protocol used for web pages. The term 'web services' is commonly used for web servers which provide XML-RPC based programmatic interface in addition to regular web pages service.

The Java support for XMLRPC comes from Apache project rather than standard Sun JDK (see <http://ws.apache.org/xmlrpc/>). The examples A.13 through A.16 exemplify server and client side XMLRPC with Java, with an additional client in Python. The class of object shared via XMLRPC must be declared public and for this reason put on a separate file. Unlike RMI, each client request causes an instance of this class to be created. For this reason the job counter in the example is declared as a static class variable rather than as an instance variable.

XMLRPC standard was later 'relaxed' to allow transfer of language-specific data types, by enabling the so called vendor extensions. These facilities are omitted in our examples since it seriously undermines the portability of the mechanism. Furthermore if one is to use Java specific facilities, RMI is a much better choice.

15.1 XMLRPC Introspection

XMLRPC allows a client to ask method signature, and even get method help from server. These extensions are called introspection extensions, and their use is exemplified in the code referred to above.

15.2 Asynchronous XMLRPC calls

The Apache library has a mechanism to make asynchronous calls, which can be utilized for algorithm parallelization purposes. See example A.17. By using asynchronous callback objects, one can replace or augment thread pool-like techniques for parallelization.

Homework Assignment V Repeat assignment IV using XMLRPC instead of RMI.

Homework Assignment VI (Work in teams of two) Choose a distributed computing problem and present problem description and requirements for the software you will implement to solve the problem.

16 Agent based distributed systems

All system architectures we have exemplified has the common theme that the parts are designed to work together to form a grand design. On the other hand most real world phenomena demonstrates a different behavior in which complex systems emerge from behavior and interaction of uncoordinated parts. Agent based design is a recent approach to simulate, replace or support such processes. The approach has proved useful in simulating and experimenting with systems that are otherwise not easily accessible (e.g. biological systems, economic systems) or support decision making (e.g. emergency decision making simulations).

The key element in agent based design and multi agent simulations is that agents have only local information and there is no central coordination which they rely upon.

Examples: HeatBugs, distributed auction.

References

- [1] J.C.R. Licklider and Robert W. Taylor. The computer as a communication device. *Science and Technology*, April 1968.
- [2] William Stallings. *Data and Computer Communications (7th edition)*. Pearson Education, 2004.

A Code examples

A.1 UDP Broadcast

```
/*
 * Java UDP broadcasting peer
 * Author : Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 *
 * This program uses a single class which broadcasts single line messages
 * entered by user, hence can be used as a primitive LAN chat system.
 *
 * Usage:
 *     UDPBroadcast port
 */

import java.net.*;
import java.io.*;
import java.util.*;

class Listener implements Runnable {
    int port;
    DatagramSocket socket;
    Thread thread;
    public Listener (DatagramSocket socket) {
this.socket=socket;
        thread =new Thread(this);
        thread.setDaemon(true);
        thread.start();
    }

    public void run() {
        System.out.println("Listener has started");
try {
        byte[] receiveBuffer=new byte[1500];
        while(true) {
            DatagramPacket receivePacket=new DatagramPacket(receiveBuffer, receiveBuffer.length);
            socket.receive(receivePacket); //THIS IS A BLOCKING CALL!
            String what=new String(receiveBuffer,receivePacket.getOffset(),receivePacket.getLength());
            System.out.println(String.format("Received from %s:%d --> %s", receivePacket.getAddress(),
                receivePacket.getPort(), what));
        }
    } catch(IOException e) {
        System.out.println(String.format("Listener error : %s",e));
        return;
    }
}

public class UDPBroadcast {
    public static void main(String[] argv){
if (argv.length<1) {
    System.out.println("Usage:");
    System.out.println("    UDPBroadcast port");
    System.exit(0);
}
}
```

```

int port=Integer.parseInt(argv[0]);
try{
    DatagramSocket socket=new DatagramSocket(port);
    //socket.bind(new InetSocketAddress(0));
    System.out.println(socket.getLocalSocketAddress());
    socket.setBroadcast(true);
    new Listener(socket);
    InetAddress broadcastAddress=InetAddress.getByName("255.255.255.255");
    while(true) {
        try {
            String msg=System.console().readLine("Enter your message:");
            byte[] buffer=msg.getBytes();
            DatagramPacket packet=new DatagramPacket(buffer,buffer.length,broadcastAddress,port);
            socket.send(packet);

        }catch(Exception e) {
            System.out.println(String.format("Announcer error : %s",e));
            e.printStackTrace(System.out);
            System.exit(0);
        }
    }
}catch(Exception e) {System.out.println(e);}
}
}

```

A.2 TCP Echo Server

```
/*
 * Echo Server
 * Author: Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 */
import java.net.*;
import java.io.*;

public class TCPEchoServer {
    public static void main(String[] args){
        if (args.length<1) {
            System.out.println("Usage:");
            System.out.println(" TCPEchoServer portno");
            System.exit(0);
        }
        try {
            System.out.println("Opening server socket at port : "+args[0]);
            ServerSocket serverSocket = new ServerSocket(Integer.parseInt(args[0]));
            System.out.println("Accepting clients");
            while (true) {
                Socket clientSocket = serverSocket.accept();
                clientSocket.setReceiveBufferSize(1500);
                System.out.println("Client accepted. Remote address:"+clientSocket.getInetAddress());
                PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
                BufferedReader in = new BufferedReader(
                    new InputStreamReader(
                        clientSocket.getInputStream()));
                String inputLine, outputLine;
                while ((inputLine = in.readLine()) != null) {
                    System.out.println("Got msg from client : "+inputLine);
                    outputLine = inputLine;
                    out.println(outputLine);
                }
                out.flush();
                System.out.println("Client disconnected");
                out.close();
                in.close();
                clientSocket.close();
            }
            //serverSocket.close();
        } catch (Exception e) {
            System.out.println("Something went wrong!");
            System.out.println(e);
        }
    }
}
```

A.3 TCP Echo Client

```
/*
 * EchoClient.java
 * Author : Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 *
 * Run the program without any parameters to see its usage.
 */
import java.net.*;
import java.io.*;
import java.util.*;

public class TCPEchoClient{
public static void main(String argv[]){
    if (argv.length<2) {
        System.out.println("Usage:");
        System.out.println(" TCPEchoClient servername portno");
        System.exit(0);
    }
    try{
Socket socket = new Socket(argv[0], Integer.parseInt(argv[1]));
socket.setSoTimeout(1000); //socket timeout 1000 ms
BufferedReader fromServer = new BufferedReader(new InputStreamReader(socket.getInputStream()));
PrintWriter toServer = new PrintWriter(socket.getOutputStream());
        BufferedReader console = new BufferedReader(new InputStreamReader(System.in));
while (true) {
    System.out.println("Enter something to send to server:");
String toSend = console.readLine();
System.out.println("Sending to server...");
toServer.println(toSend);
toServer.flush();
Thread.sleep(1);
try {
    System.out.println("Reading from server...");
    String line=fromServer.readLine();
    System.out.println("Response:"+line);
    } catch (Exception e){
        System.out.println("Something went wrong when reading from socket!");
        System.out.println(e);
    }
}
}
catch(Exception e){
    System.out.println("Something went wrong!");
    System.out.println(e);
}
}
}
```

A.4 UDP Multicast

```
/*
 * Java UDP multicast peer
 * Author : Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 *
 * This program uses a single class which multicasts single line messages
 * entered by user, hence can be used as a primitive LAN chat system.
 *
 * Usage:
 *     UDPBroadcast port
 */

import java.net.*;
import java.io.*;
import java.util.*;

class Listener implements Runnable {
    int port;
    MulticastSocket socket;
    InetAddress address;
    Thread thread;
    public Listener (int port) throws IOException{
        socket = new MulticastSocket(port);
        address = InetAddress.getByName("230.0.0.1");
socket.joinGroup(address);
        thread =new Thread(this);
        thread.setDaemon(true);
        thread.start();
    }

    public void run() {
        System.out.println("Listener has started");
try {
        byte[] receiveBuffer=new byte[1500];
        while(true) {
            DatagramPacket receivePacket=new DatagramPacket(receiveBuffer, receiveBuffer.length);
            socket.receive(receivePacket);
            String what=new String(receiveBuffer,receivePacket.getOffset(),receivePacket.getLength());
            System.out.println(String.format("Received from %s:%d --> %s", receivePacket.getAddress(),
                receivePacket.getPort(), what));
        }
    } catch(IOException e) {
        System.out.println(String.format("Listener error : %s",e));
    }

        try{
            socket.leaveGroup(address);
            socket.close();
        } catch (IOException e) {}

    }
}

public class UDPMulticast {
    public static void main(String[] argv){
```

```

if (argv.length<1) {
    System.out.println("Usage:");
    System.out.println("  UDPBroadcast port");
    System.exit(0);
}

    int port=Integer.parseInt(argv[0]);
    try{
        DatagramSocket socket=new DatagramSocket(port);
        System.out.println(socket.getLocalSocketAddress());
        new Listener(port+1);
        InetAddress multicastAddress=InetAddress.getByName("230.0.0.1");
        while(true) {
            try {
                String msg=System.console().readLine("Enter your message:");
                byte[] buffer=msg.getBytes();
                DatagramPacket packet=new DatagramPacket(buffer,buffer.length,multicastAddress,port);
                socket.send(packet);

            }catch(Exception e) {
                System.out.println(String.format("Announcer error : %s",e));
                e.printStackTrace(System.out);
                System.exit(0);
            }
        }
    }catch(Exception e) {System.out.println(e);}
}

```

A.5 SSL TCP Echo Server (which does not work!)

A.6 SSL TCP Echo Client

```
/**
 * An Echo client using SSL
 * Author:Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 */
import java.io.*;
import java.net.*;
import javax.net.ssl.*;
import java.security.*;

public class SSLclient {
    public static void main(String[] args) {
        String serverName="localhost";
        int port=8000;
        try {
            serverName=args[0];
            port=Integer.parseInt(args[1]);
        }catch(Exception e) {
            System.out.println("Usage:\n SSLclient host port");
            System.exit(0);
        }
        try {
            //Security.addProvider(new com.sun.net.ssl.internal.ssl.Provider());
            SSLSocketFactory factory = (SSLSocketFactory)
                SSLSocketFactory.getDefault();
            SSLSocket socket = (SSLSocket)
                factory.createSocket(serverName, port);
            socket.setSoTimeout(500);
            BufferedReader fromServer = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintWriter toServer = new PrintWriter(socket.getOutputStream());
            while (true) {
                String message = System.console().readLine("Enter something to send to server:");
                toServer.println(message);
                toServer.flush();
                try {
                    while(true) {
                        String line=fromServer.readLine();
                        System.out.println("Response:"+line);
                    }
                }catch(SocketTimeoutException ignore) {
                }catch (Exception e){
                    System.out.println("Something went wrong when reading from socket!");
                    System.out.println(e);
                }
            }
        }catch(Exception e){
            System.out.println("Something went wrong!");
            System.out.println(e);
        }
    }
}
```

A.7 SSL TCP Echo Server which works

```
/**
 * An Echo server using SSL
 * Author:Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 */
import java.io.*;
import javax.net.*;
import javax.net.ssl.*;
import java.security.*;

class ClientHandler implements Runnable{
    SSLSocket socket;
    ClientHandler (SSLSocket socket) {
        this.socket=socket;
        Thread thread=new Thread(this);
        thread.setDaemon(true);
        thread.start();
    }
    public void run() {
        System.out.println("Serving client");
        try{
            BufferedReader in =
                new BufferedReader(
                    new InputStreamReader( socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream());
            String inputLine, outputLine;
            while ((inputLine = in.readLine()) != null) {
                outputLine = inputLine;
                out.println(outputLine);
                out.flush();
            }
            System.out.println("Client disconnected");
            out.close();
            in.close();
            socket.close();
        } catch(Exception e) {
            System.out.println("Something went wrong!");
            System.out.println(e);
        }
    }
}

public class SSLserver {
    public static void main(String[] args) {
        int port=8000;
        String certFile="";
        try {
            port=Integer.parseInt(args[0]);
            certFile=args[1];
        } catch(Exception e) {
            System.out.println("Usage:\n SSLserver <port> <servercertificatefile>");
            System.exit(0);
        }
        SSLServerSocket server;
```

```

try {
    //THE FOLLOWING LINES ARE NEEDED FOR SSL CERTIFICATE LOADING
    KeyStore ks = KeyStore.getInstance("JKS");
    char storePassword[] = System.console().readPassword("Enter store password:");
    char keyPassword[] = System.console().readPassword("Enter certificate key password:");
    ks.load(new FileInputStream(certFile), storePassword);
    KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
    kmf.init(ks, keyPassword);
    SSLContext sslcontext = SSLContext.getInstance("SSLv3");
    sslcontext.init(kmf.getKeyManagers(), null, null);
    ServerSocketFactory ssf = sslcontext.getServerSocketFactory();
    server = (SSLServerSocket) ssf.createServerSocket(port);
    for(;;) {
        SSLSocket client = (SSLSocket) server.accept();
        new ClientHandler(client);
    }
    } catch(Exception e) {
System.out.println("Something went wrong!");
    System.out.println(e);
    }
}
}

```

A.8 Simple HTTP-S Server

```
/**
 * A simplistic secure HTTP server using SSL
 * Author:Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 */
import java.io.*;
import java.net.*;
import javax.net.*;
import javax.net.ssl.*;
import java.security.*;
import java.util.*;

class ClientHandler implements Runnable{
    Socket socket;
    ClientHandler (Socket socket) {
        this.socket=socket;
        Thread thread=new Thread(this);
        thread.setDaemon(true);
        thread.start();
    }
    public void run() {
        Socket clientSocket;
        System.out.println("Serving client");
        try{
            BufferedReader in =
                new BufferedReader(
                    new InputStreamReader( socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream());
            String request, response;
            request=in.readLine();
            StringTokenizer st = new StringTokenizer( request );
            try {
                st.nextToken();
                String path=st.nextToken();
                File f=new File("."+path);
                FileInputStream file=new FileInputStream(f);
                out.write("HTTP/1.0 200 OK\r\n");
                out.write("Content-Length: " + f.length() +"\r\n");
                out.write("Content-Type: text/plain \r\n\r\n");
                try {
                    while(true)
                        out.write(file.read());
                }catch(Exception ignore) {}
            } catch(Exception e) {
                System.out.println(e);
            }
            out.close();
            in.close();
            socket.close();
        } catch(Exception e) {
            System.out.println("Something went wrong!");
            System.out.println(e);
        }
    }
}
```

```

}

class Server implements Runnable {
    ServerSocket socket;
    Server(ServerSocket socket) {
        this.socket=socket;
        Thread thread=new Thread(this);
        thread.setDaemon(true);
        thread.start();
    }
    public void run() {
        Socket clientSocket;
        try {
            while(true) {
                clientSocket=socket.accept();
                new ClientHandler(clientSocket);
            }
        }catch(Exception e) {
            System.out.println(e);
        }
    }
}

public class HTTPSserver {
    public static void main(String[] args) {
        String certFile="";
        try {
            certFile=args[0];
        } catch(Exception e) {
            System.out.println("Usage:\n HTTPSserver <servercertificatefile>");
            System.exit(0);
        }
        ServerSocket socket;
        SSLServerSocket secureSocket;
        try {
            //THE FOLLOWING LINES ARE NEEDED FOR SSL CERTIFICATE LOADING
            KeyStore ks = KeyStore.getInstance("JKS");
            char storePassword[] = System.console().readPassword("Enter store password:");
            char keyPassword[] = System.console().readPassword("Enter certificate key password:");
            ks.load(new FileInputStream(certFile), storePassword);
            KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
            kmf.init(ks, keyPassword);
            SSLContext sslcontext = SSLContext.getInstance("SSLv3");
            sslcontext.init(kmf.getKeyManagers(), null, null);
            ServerSocketFactory ssf = sslcontext.getServerSocketFactory();

            socket =new ServerSocket(80);
            secureSocket = (SSLServerSocket) ssf.createServerSocket(443);

            new Server(socket);
            new Server(secureSocket);
            System.console().readLine("Press ENTER to stop server.");
        } catch(Exception e) {
            System.out.println("Something went wrong!");
            System.out.println(e);
        }
    }
}

```

}
 }
 }

A.9 RMI Example - Interface

```
/**
 * An RMI Interface definition
 * Author: Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 */
import java.rmi.*;
import java.util.*;

public interface RMIExampleInterface extends Remote {
    /** Check an return whether the given integer is prime*/
    boolean isPrime(int n) throws RemoteException;

    /** Return a list of prime factors of given integer */
    ArrayList factorize(int n) throws RemoteException;

    /** Return the number of factorizations or primality checks
     * carried out by the remote object*/
    int jobCount() throws RemoteException;
}
```

A.10 RMI Example - Server

```
/**
 * An example RMI Server
 * Author: Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 */
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.util.*;

class RMIExampleServerImplementation implements RMIExampleInterface{
    private int jobcount;
    public RMIExampleServerImplementation() throws RemoteException {
        jobcount=0;
    }
    public boolean isPrime(int n) throws RemoteException {
        if (n<=0)
            throw new RemoteException("Cannot check non-positive numbers");
        synchronized (this) {
            jobcount+=1;
        }
        for(int j=2;j<n;j=j+1)
            if (n%j==0)
                return false;
        return true;
    }
    public ArrayList factorize(int n) throws RemoteException {
        if (n<=0)
            throw new RemoteException("Cannot factorize non-positive numbers");
        synchronized (this) {
            jobcount+=1;
        }
        System.console().format("Factorizing %d\n",n);
    }
}
```

```

        ArrayList factors=new ArrayList();
        int m=n;
        for(int j=2;j<n;j=j+1)
            if (isPrime(j))
                while (m%j==0) {
                    factors.add(j);
                    m=m/j;
                }
        return factors;
    }
    public int jobCount() throws RemoteException{
        return jobcount;
    }
}

public class RMIExampleServer {
    public static void main(String[] args) {
        try {
            //Create the shared object
            RMIExampleServerImplementation object = new RMIExampleServerImplementation();
            //Export the object to RMI subsystem to be served on a random suitable network socket
            RMIExampleInterface shared = (RMIExampleInterface) UnicastRemoteObject.exportObject(object,

// Bind the remote object's stub in the registry so that
            // socket can be found by callers and named object can be located
            Registry registry = LocateRegistry.getRegistry();
            registry.bind("Factorizer", shared);

            System.out.println("Server is ready");
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

A.11 RMI Example - Client

```

import java.io.*;
import java.util.*;
import java.rmi.*;
import java.rmi.registry.*;

public class RMIExampleClient {
    public static void main(String[] args) {
        String host="";
        String objectname="";
        try {
            host=args[0];
            objectname=args[1];
        }catch(Exception e) {
            System.out.println("Usage:\n RMIExampleClient <server> <object>\n for example: RMIExampleClient
            System.exit(0);
        }
    }
}

```

```

//Locate the serve and the object on it
Registry registry = LocateRegistry.getRegistry(host);
RMIEExampleInterface server = (RMIEExampleInterface)registry.lookup(objectname);
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

for(;;) {
    try{
        String input=in.readLine();
        int n=Integer.parseInt(input);
        ArrayList factors=server.factorize(n); //call remote method
        boolean isPrime=server.isPrime(n);    //call remote method
        System.console().format("%d is prime? :%s \nPrime factors of %d: %s\n", n, isPrime,
    } catch (Exception e){
        System.out.println(e);
        break;
    }
}
} catch (Exception e) {
    System.out.println(e);
}
}
}

```

A.12 RMI Example - SSL Server

```
/**
 * An example RMI Server
 * Author: Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 */

import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.util.*;
import java.net.*;
import javax.net.ssl.*;
import java.security.KeyStore;
import javax.net.*;
import javax.security.cert.X509Certificate;

//This implementation extends UnicastRemoteObject
class RMIExampleServerImplementation extends UnicastRemoteObject implements RMIExampleInterface{
    private int jobcount;
    public RMIExampleServerImplementation(String storename, char[] storepass, char[] keypass) throws RemoteException {
        //Following changes the underlying socket mechanism of UnicastRemoteObject
        super(0, new RMISSLClientSocketFactory(),
            new RMISSSLServerSocketFactory(storename,storepass,keypass));
        jobcount=0;
    }
    public boolean isPrime(int n) throws RemoteException {
        if (n<=0)
            throw new RemoteException("Cannot check non-positive numbers");
        synchronized (this) {
            jobcount+=1;
        }
        for(int j=2;j<n;j=j+1)
            if (n%j==0)
                return false;
        return true;
    }
    public ArrayList factorize(int n) throws RemoteException {
        if (n<=0)
            throw new RemoteException("Cannot factorize non-positive numbers");
        synchronized (this) {
            jobcount+=1;
        }
        System.console().format("Factorizing %d\n",n);
        ArrayList factors=new ArrayList();
        int m=n;
        for(int j=2;j<n;j=j+1)
            if (isPrime(j))
                while (m%j==0) {
                    factors.add(j);
                    m=m/j;
                }
        return factors;
    }
}
```

```

    public int jobCount() throws RemoteException{
        return jobcount;
    }
}

class RMISSLServerSocketFactory
    implements RMIServerSocketFactory, Serializable {
    SSLContext ctx;
    RMISSLServerSocketFactory(String storename, char[] storepass, char[] keypass) {
        try {
            // set up key manager to do server authentication
            KeyManagerFactory kmf;
            KeyStore ks;

            ctx = SSLContext.getInstance("TLS");
            kmf = KeyManagerFactory.getInstance("SunX509");
            ks = KeyStore.getInstance("JKS");

            ks.load(new FileInputStream(storename), storepass);
            kmf.init(ks, keypass);
            ctx.init(kmf.getKeyManagers(), null, null);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public ServerSocket createServerSocket(int port) throws IOException {
        SSLServerSocketFactory ssf = null;
        try {
            ssf = ctx.getServerSocketFactory();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ssf.createServerSocket(port);
    }
}

class RMISSLClientSocketFactory
    implements RMIClientSocketFactory, Serializable {

    public Socket createSocket(String host, int port)
    throws IOException
    {
        SSLSocketFactory factory =
        (SSLSocketFactory)SSLSocketFactory.getDefault();
        SSLSocket socket = (SSLSocket)factory.createSocket(host, port);
        return socket;
    }
}

public class RMIExampleServerSSL {
    public static void main(String[] args) {
        try {
            int port=8000;
            String certFile="";

```

```

try {
    port=Integer.parseInt(args[0]);
    certFile=args[1];
} catch(Exception e) {
    System.out.println("Usage:\n  SSLserver <port> <servercertificatefile>");
    System.exit(0);
}
//THE FOLLOWING LINES ARE NEEDED FOR SSL CERTIFICATE LOADING
char storePassword[] = System.console().readPassword("Enter store password:");
char keyPassword[] = System.console().readPassword("Enter certificate key password:");

//Create the shared object
RMIExampleServerImplementation object = new RMIExampleServerImplementation(certFile,storePas
//Since the object extends UnicastRemoteObject, there is no need to export the object to RMI
//RMIExampleInterface shared = (RMIExampleInterface) UnicastRemoteObject.exportObject(object

// Bind the remote object's stub in the registry so that
// socket can be found by callers and named object can be located
Registry registry = LocateRegistry.getRegistry();
registry.bind("Factorizer", object);

    System.out.println("Server is ready");
} catch (Exception e) {
    System.out.println(e);
}
}
}

```

A.13 XMLRPC Example - The shared object

```
/**
 * A class to be shared over XMLRPC
 * Author: Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 *
 * Since the class has to be public, it is put on a separate file
 */
import java.util.*;

public class XMLRPCShared{
    private static int jobCount;

    /** Return whether given number is prime*/
    public boolean isPrime(int i){
        for(int j=2;j<i;j=j+1)
            if (i%j==0)
                return false;
        return true;
    }

    /** Return prime factors of given number */
    //XMLRPC limits return value to some basic types in the standard
    // or objects or arrays of objects
    public Object[] primeFactors(int n) {
        System.console().format("Factorizing %d\n",n);
        ArrayList<Integer> factors=new ArrayList<Integer>();
        int m=n;
        for(int j=2;j<n;j=j+1)
            if (isPrime(j))
                while (m%j==0) {
                    factors.add(new Integer(j));
                    m=m/j;
                }
        synchronized (XMLRPCShared.class) {
            jobCount+=1;
        }
        System.console().format("Factors of %d: ",n);
        System.out.println(factors);
        return factors.toArray();
    }

    public int getJobCount() {
        return jobCount;
    }
}
```

A.14 XMLRPC Example - Server

```
/*
 * Simple XMLRPC server using Apache's XML-RPC library
 *
 * Author: Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 *
 * This example uses builtin web server of Apache library.
```

```

* Download and extract the library from http://ws.apache.org/xmlrpc/
* This program is tested with version 3.1.1 of the Apache xmlrpc library.
* Once you have the library extracted somewhere you must compile and run this code similar to following:
*
* CLASSPATH=../xmlrpc-3.1.1/lib/commons-logging-1.1.jar:../xmlrpc-3.1.1/lib/ws-commons-util-1.0.2.jar:..
* export CLASSPATH
* javac XMLRPCServerExample.java
* java XMLRPCServerExample 8000
*
*/
import org.apache.xmlrpc.webserver.*;
import org.apache.xmlrpc.server.*;
import org.apache.xmlrpc.common.*;
import org.apache.xmlrpc.metadata.*;
import java.util.*;

public class XMLRPCServerExample{
    public static void main(String args[]){
        //Read the port number to use for XML-RPC enabled Web server
        int port=0;
        try {
            port=Integer.parseInt(args[0]);
        }catch(Exception e) {
            System.out.println("Usage:\n XMLRPCServerExample <port-no>");
            System.exit(0);
        }
        try{
            WebServer webserver = new WebServer(port);
            XmlRpcStreamServer rpcserver=webserver.getXmlRpcServer();
            PropertyHandlerMapping phm = new PropertyHandlerMapping();
            phm.addHandler("Factorizer", XMLRPCShared.class);
            rpcserver.setHandlerMapping(phm);
            //Following line enables introspection:
            XmlRpcSystemImpl.addSystemHandler((PropertyHandlerMapping)rpcserver.getHandlerMapping());
            //Following could be set if Java specific objects (eg. primitive types) needs to be returned
            // the so called Vendor specific extensions
            //XmlRpcServerConfigImpl serverConfig =(XmlRpcServerConfigImpl) rpcserver.getConfig();
            //serverConfig.setEnabledForExtensions(true);
            //serverConfig.setContentLengthOptional(true);
            webserver.start();
            System.out.println("XML-RPC server is awaiting for calls at port:"+port);
        } catch (Exception e){
            System.out.println(e);
        }
    }
}

```

A.15 XMLRPC Example - Client

```

/*
* Simple XMLRPC Client using Apache's XML-RPC library
*
* Author: Mehmet Gencer, mgencer@cs.bilgi.edu.tr

```

```

*
* You will need both Apache xml-rpc library (http://ws.apache.org/xmlrpc/)
* and httpclient library (http://hc.apache.org/downloads.cgi)
* This program is tested with version 3.1.1 of the Apache xmlrpc library and 4.0beta2 of httpclient library
* Once you have the library extracted somewhere you must compile and run this code similar to following:
*
* CLASSPATH=../xmlrpc-3.1.1/lib/commons-logging-1.1.jar:../xmlrpc-3.1.1/lib/ws-commons-util-1.0.2.jar:..
* export CLASSPATH
* javac XMLRPCClientExample.java
* java XMLRPCClientExample http://localhost:8000/xmlrpc
*/

import java.util.*;
import java.net.*;
import org.apache.xmlrpc.*;
import org.apache.xmlrpc.client.*;

public class XMLRPCClientExample {
    public static void main(String[] args) throws Exception {
        String serverUrl="";
        try{
            serverUrl=args[0];
        }catch(Exception e) {
            System.out.println("Usage:\n XMLRPCClientExample <server-url>\n Where server url is something like http://localhost:8000/xmlrpc");
            System.exit(0);
        }
        XmlRpcClientConfigImpl config=new XmlRpcClientConfigImpl();
        config.setServerURL(new URL(serverUrl));
        //Following could be set if Java specific objects (eg. primitive types) needs to be returned
        // the so called Vendor specific extensions
        //config.setEnabledForExtensions(true);
        //config.setContentLengthOptional(true);
        XmlRpcClient client= new XmlRpcClient();
        client.setConfig(config);
        try {
            // The parameters needed for call
            Object[] params = new Object[1];
            int n;
            for(;;) {
                try{
                    n=Integer.parseInt(System.console().readLine("Enter a number to factorize:"));
                } catch(NumberFormatException e) {break;}
                //parameters are always an array and always contain Object types (not primitive types)
                //so the following 'packaging' is necessary
                params[0]=new Integer(n);
                Object[] factors=(Object[]) client.execute("Factorizer.primeFactors",params);
                System.console().format("Factors of %d are: ",n);
                for (int i=0;i<factors.length;i++)
                    System.console().format("%d ",((Integer)factors[i]).intValue());
                System.out.println();
            }
            //The following call has no parameters, hence an array of length 0
            System.console().format("Server job count: %d \n",client.execute("Factorizer.getJobCount",new Object[0]));
        }catch(Exception e) {
    
```

```

        System.out.println(e);
    }
}

```

A.16 XMLRPC Example - A Python client

```

import sys,xmlrpclib

try:
    serverurl=sys.argv[1]
except:
    print "Usage:\n %s <server-url>"%sys.argv[0]
    sys.exit()
server = xmlrpclib.ServerProxy(serverurl) #create server connection
#Display method help
for method in server.system.listMethods():
    print "%s: %s\n %s"%(method, server.system.methodSignature(method), server.system.methodHelp(m
while 1:#Repeatetly input from user
    try:n=input("Enter an integer:")
    except:
        print "Server job count:",server.Factorizer.getJobCount()
        break
    print "N is prime? ", server.Factorizer.isPrime(n)
    print "N factors: ", server.Factorizer.primeFactors(n)

```

A.17 XMLRPC Example - An Asynchronous Client

```
/*
 * Asynchronous XMLRPC Client using Apache's XML-RPC library
 *
 * Author: Mehmet Gencer, mgencer@cs.bilgi.edu.tr
 *
 */
import java.util.*;
import java.net.*;
import org.apache.xmlrpc.*;
import org.apache.xmlrpc.client.*;

class MyCallback implements AsyncCallback {
    int n;
    boolean done;
    public MyCallback(int n) {
        this.n=n;
        done=false;
    }
    public void handleResult (XmlRpcRequest request, Object result) {
        System.console().format("Factors of %d are: ", n);
        Object[] results=(Object[]) result;
        for (int i=0;i<results.length;i++)
            System.console().format("%d ",((Integer)results[i]).intValue());
        System.out.println();
        done=true;
    }
    public void handleError (XmlRpcRequest request, Throwable error) {
        System.console().format("An exception is thrown when factorizing %d:\n %s ", n, error.toString());
        done=true;
    }
}

public class XMLRPCAsyncClient{
    // this method returns a string
    public static void main(String args[]){
        String serverUrl="";
        try{
            serverUrl=args[0];
        }catch(Exception e) {
            System.out.println("Usage:\n XMLRPCAsyncClient <server-url>\n Where server url is something like http://www.xmlrpc.com");
            System.exit(0);
        }
        try{
            XmlRpcClientConfigImpl config=new XmlRpcClientConfigImpl();
            config.setServerURL(new URL(serverUrl));
            XmlRpcClient client= new XmlRpcClient();
            client.setConfig(config);
            Object[] params = new Object[]{new Integer(1234)};
            MyCallback c1=new MyCallback(1234);
            client.executeAsync("Factorizer.primeFactors", params, c1);
            params=new Object[]{new Integer(12345)};
            MyCallback c2=new MyCallback(12345);
            client.executeAsync("Factorizer.primeFactors", params, c2);
        }
    }
}
```

```
        while (!(c1.done&&c2.done));
    }catch(Exception e){
        System.out.println(e);
    }
}
}
```