



Lebanese University

Faculty of Engineering III

# **Local Area Network Messaging Application using Network Programming**

By

Oussama H. Kondakji

Supervised By

Dr. Mahmoud Doughan

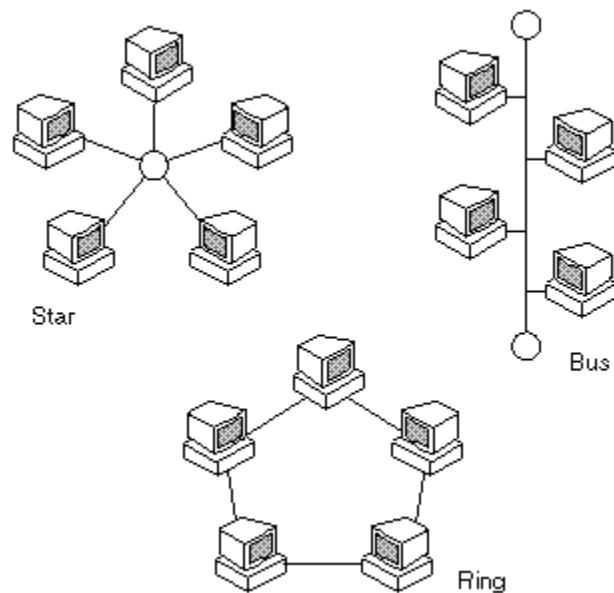
## Table of Content

Chapter I: Introduction to Computer Networks.....	3
Chapter II: TCP Communication on LANs.....	9
II.1 Introduction.....	9
II.2 Mechanism of TCP Communication.....	10
II.3 TCP Session Establishment .....	10
II.4 TCP's Ordered Delivery.....	11
II.5 Reliable Transmission and Retransmission.....	12
II.6 TCP Session Termination.....	13
II.7 Conclusion.....	14
Chapter III: Client-Server-Based Messaging.....	15
III.1 Introduction.....	15
III.2 The System's Architecture.....	16
III.3 Connecting to Server.....	17
III.4 Message Forwarding.....	18
III.5 Code.....	19
III.6 Conclusion.....	22
Conclusion.....	23
References.....	24

## Chapter I: Introduction to Computer Networks

**N**etworks in telecommunication are defined as a group of two or more terminal nodes which are linked together to enable telecommunication between them. The transmission links connect the nodes together. The nodes use circuit switching, message switching or packet switching to pass the signal through the correct links and nodes to reach the correct destination terminal. Each terminal in the network usually has a unique address so messages or connections can be routed to the correct recipients. The collection of addresses in the network is called the address space. Our main concern in this project is the networks involved in packet switching, which are basically computer networks, or data networks that involve digital telecommunications which allows nodes to share resources. In computer networks, computing devices exchange data with each other using connections between nodes (data links.) These data links are established over cable media such as wires (Serial or Twister Pair) or optic cables, or wireless media such as WiFi.

This type of networks are considered as digital communication systems, which means that the original information send and received is digital (binary). In general, computer networks follow very specific protocols and standards in the representation of data, and the nature of the transmission of data between two devices or among several of them.. Computer networks support an enormous number of applications and services such as access to the World Wide Web, digital video, digital audio, shared use of application and storage servers, printers, and fax machines, and use of email and instant messaging applications as well as many others. In order to understand how packets are represented and how they are structured, we must get familiar with the hierarchy and the model at which it is modified and formed. The data unit in computer networks is called a



*Figure 1.1 Computer Network Types and Forms*

Protocol Data Unit (PDU). The PDU's structure differs as it follows the protocols that consist the network it is being transmitted across.

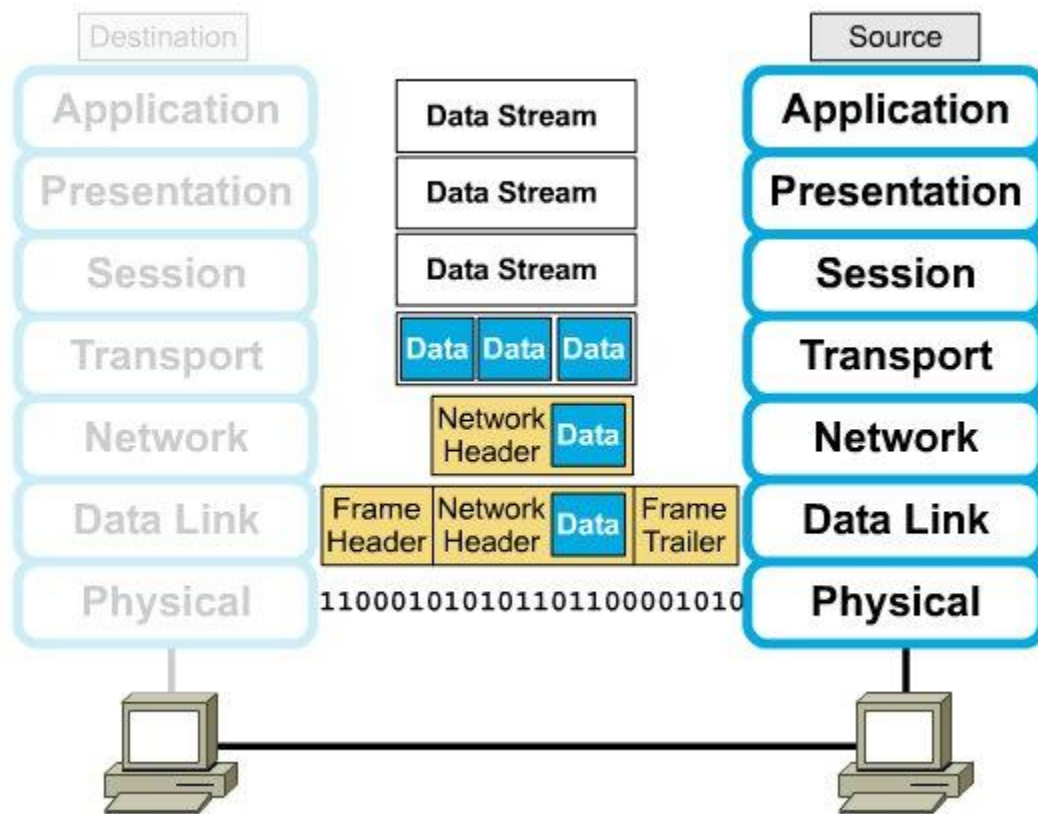


Figure 1.2 Data Encapsulation in Computer Networks

Consider two communication entities (computers) that are connected directly via an Ethernet link. The source computer has a running application, a messaging application, for instance, and is trying to communicate with the destination computer on the opposite end. The architecture that governs the representation and the representation of data going from a source entity to a destination entity in computer networks is called the OSI model. The Open Systems Interconnection model (OSI model) is a conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to its underlying internal structure and technology. Its goal is the interoperability of diverse communication systems with standard protocols. The model partitions a communication system into abstraction layers. The original version of the model defined seven layers. The data going from the source to the destination undergoes a U-shaped path across the layers of the model going from Application down to Physical and then through the channel and up from Physical to Application at the destination computer (Figure 1.2). The data

passes through different shapes when going from one layer to another. Before diving into the different shapes of the PDU, let's explain the importance and the role of each layer in the OSI model[2].

The Application layer is the OSI layer closest to the end user, which means both the OSI Application layer and the user interact directly with the software application. This layer interacts with software applications that implement a communicating component. Some protocols inside the Application layer are File Transfer Protocol (FTP), Trivial File Transfer Protocol (TFTP), electronic mail transport like Simple Mail Transfer Protocol (SMTP), and networking support like Domain Name System (DNS). The presentation layer establishes context between application-layer entities, in which the application-layer entities may use different syntax and semantics. This layer provides independence from data representation by translating between application and network formats. The presentation layer transforms data into the form that the application accepts. The session layer controls the dialogues (connections) between computers. It establishes, manages and terminates the connections between the local and remote application. The transport layer provides the functional and procedural means of transferring variable-length data sequences from a source to a destination host, while maintaining the quality of service functions. The transport layer controls the reliability of a given link through flow control, segmentation/desegmentation, and error control. The best-known transport protocol is the Transmission Control Protocol (TCP), and lent its name to the title of the entire suite. It is used for connection-oriented transmissions, whereas the connectionless User Datagram Protocol (UDP) is used for simpler messaging transmissions. TCP is the more complex protocol, due to its stateful design incorporating reliable transmission and data stream services.

The network layer provides the functional and procedural means of transferring variable length data sequences (called packets) from one node to another connected in "different networks". A network is a medium to which many nodes can be connected, on which every node has an address and which permits nodes connected to it to transfer messages to other nodes connected to it by merely providing the content of a message and the address of the destination node and letting the network find the way to deliver the message to the destination node, possibly routing it through intermediate nodes. Network layer protocols are many. Dynamic routing protocols like RIP, OSPF, EIGRP, and BGP are protocols working in the network layer where IP Addressing is introduced in order to segment different networks and hosts and identify end devices and Layer 3 (Network Layer) interfaces (Figure 1.3). The data link layer provides node-to-node data transfer—a link between two directly connected nodes. It detects and possibly corrects errors that may occur in the physical layer. It defines the protocol to establish and

terminate a connection between two physically connected devices. It also defines the protocol for flow control between them. Examples of data link protocols are Ethernet for local area networks (multi-node), the Point-to-Point Protocol (PPP), HDLC and ADCCP for point-to-point (dual-node) connections. The data link layer introduces a media access control address (MAC address) of a device is a unique identifier assigned to a network interface controller (NIC) for communications at the data link layer of a network segment through a Switch (Layer 2 device).

Finally, the physical layer defines the electrical, optical, and physical specifications of the data connection. It defines the relationship between a device and a physical transmission medium (for example, an electrical cable, an optical fiber cable, or a radio frequency link). This includes the layout of pins, voltages, line impedance, cable specifications, signal timing and similar characteristics for connected devices and frequency (5 GHz or 2.4 GHz etc.) for wireless devices. It is responsible for transmission and reception of unstructured raw data in a physical medium (Figure 1.4). Bit rate control is done at the physical layer.

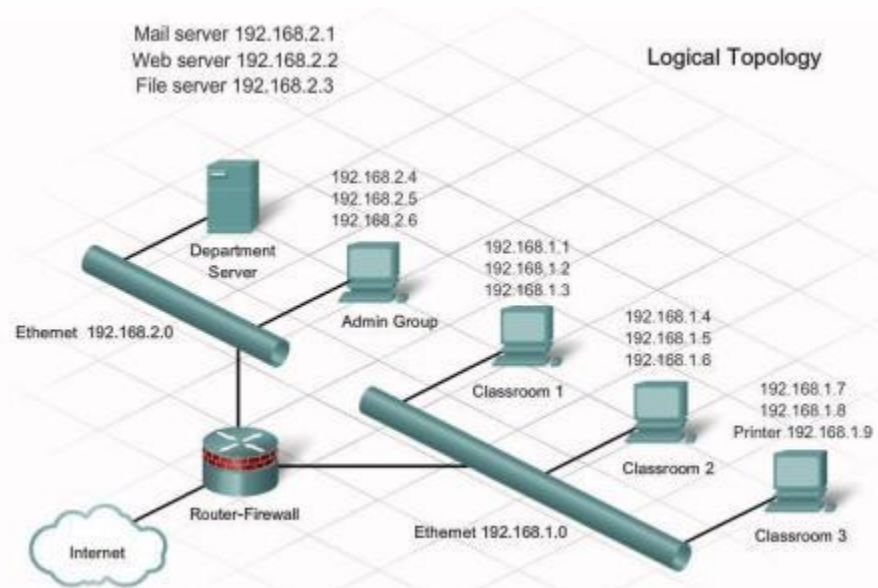


Figure 1.3 Network topology with network addressing

Now that we have understood the nature and the structure of data and data transmission in computer networks, it is inevitable that the computer networks are growing rapidly with internet and their capabilities are now reaching all the services companies and users might need. Those include banking information exchange, collaboration in the workplace, file sharing, video conferences, project management,

ecommerce, mailing services, virtualization and cloud computing, SoHo work environments and fast, reliable, and secure communication among concerned entities.

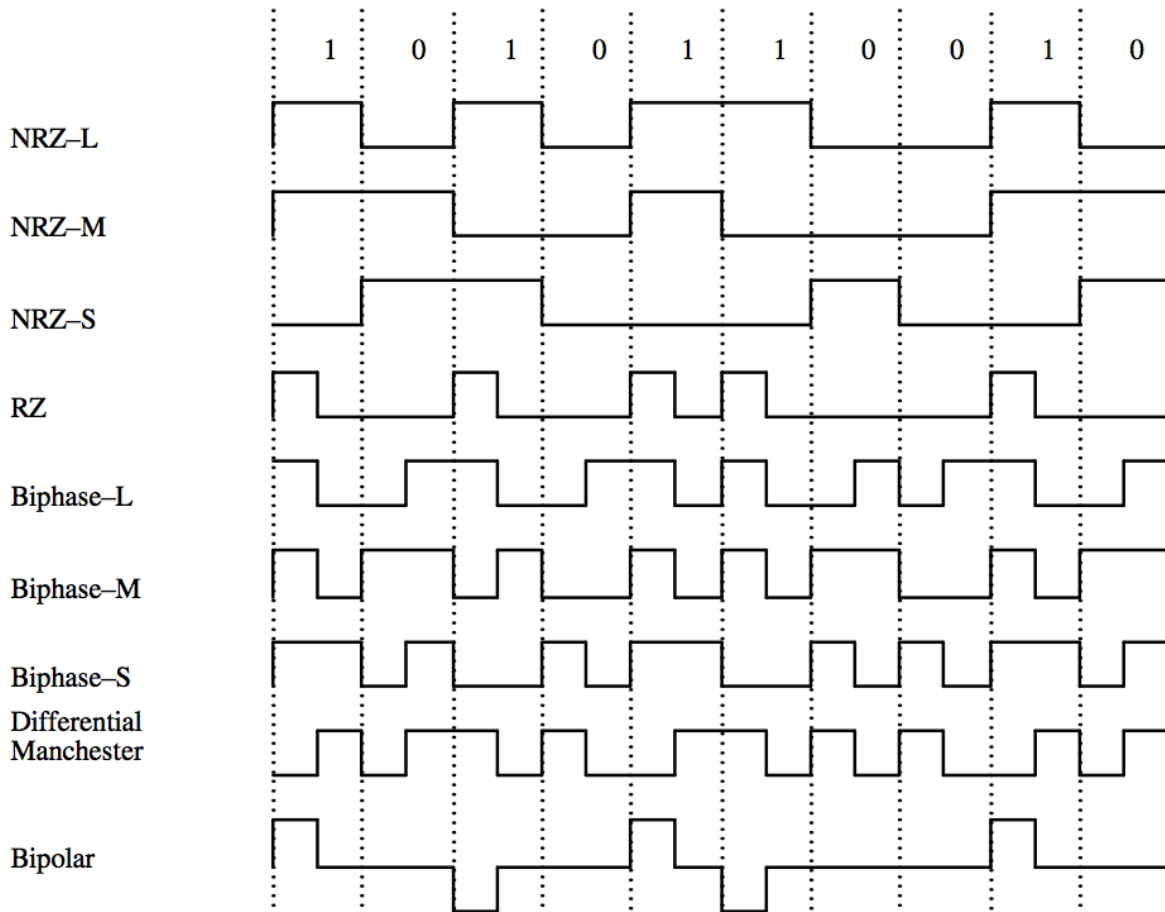


Figure 1.4 Binary coding schemas in physical layer

Therefore, the project under study is a way of utilizing computer networks in order to establish a messaging service for users existing on the same network segment (Local Area Network). Socket programming is used to implement such a service and TCP protocol to establish communication between end-users. Since it is a communication over a Local Area Network, packets exchanged among users do not go outside the network these users are in, i.e. the Internet. Hence, the communication will be on the 2<sup>nd</sup> Layer in the OSI model and the intermediary device that will be responsible for the connection among users will be a Layer 2 device, which is a Switch. A switch is a device that routes frames sent from users to other users using the source and destination MAC addresses which means that when the frame goes into the switch, the switch goes up to the data link layer only to examine the destination MAC address in order to identify the port it must send the frame through and it does not examine the IP addresses of the

two communicating users since it does not go to Layer 3 (Network Layer) because that is the function of a router.

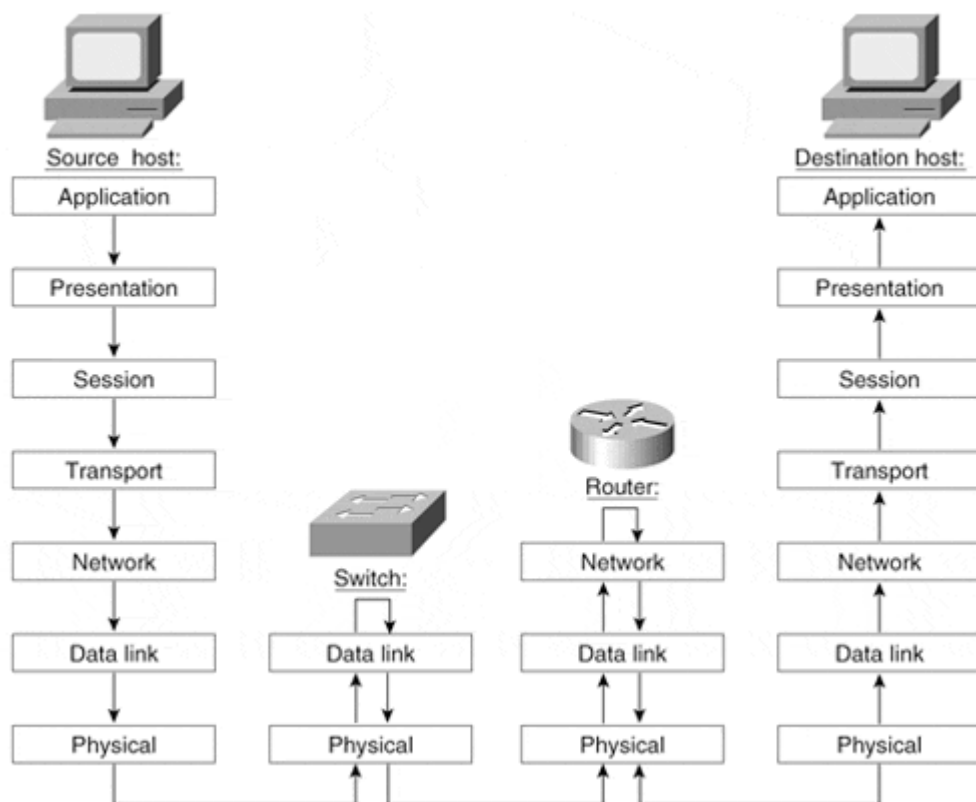


Figure 1.5 Routing and switching functionalities

So from now on, the users communicating in the scope of this project are connected and communicating via a layer 2 switch. In the next chapter, TCP protocol will be discussed in order to identify how sessions are established between the clients and the server found inside the LAN and how they communicate over a reliable connection-oriented manner.



## Chapter II: TCP Communication over LANs

### Introduction

Connection-oriented communication is very crucial to the types of services that cannot handle any loss of information during communication. On the other hand, some services can handle slight loss of information in exchange for low latency in transmission. Hence, connection-oriented communication have a higher transmission latency than connectionless transmission due to the existence of session establishment and acknowledgment mechanisms. The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability. Examples of TCP services are: web browsing that uses TCP to load pages, FTP (File Transfer) is also a great example of TCP communication, sending and receiving emails, E-banking, and remote access to network devices (SSH or Telnet). On the contrary, online gaming utilizes UDP, video conferences, IP telephony, and resolving the IP

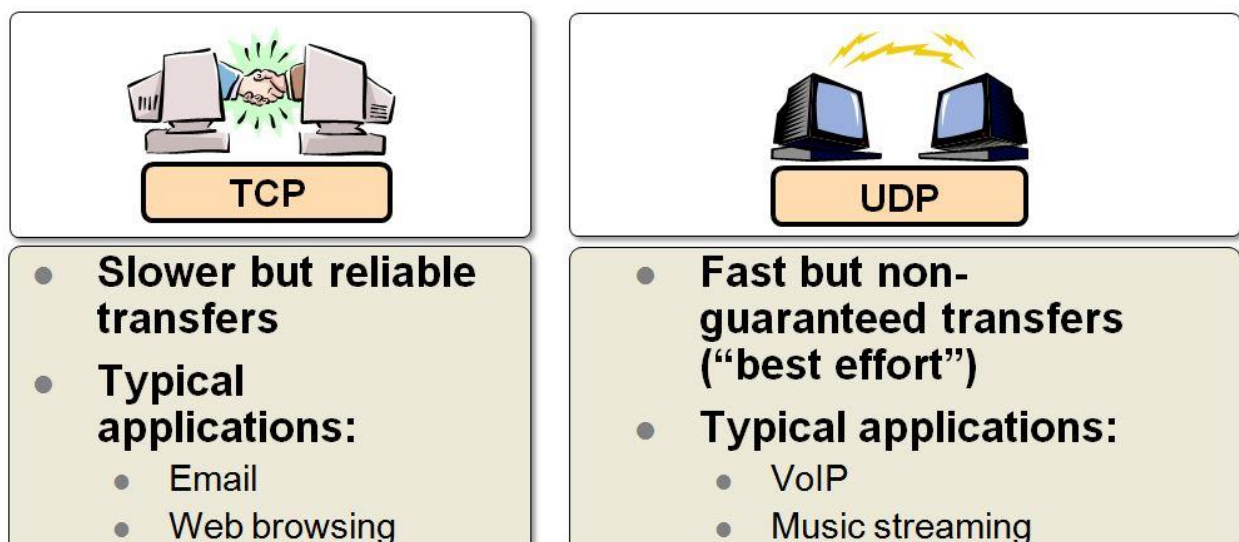


Figure 2.1 Comparison between TCP and UDP transport layer protocols

address of a website also use UDP. After understanding the concepts of TCP and UDP, how can TCP achieve such reliability?

## **Mechanism of TCP Communication**

The Transmission Control Protocol provides a communication service at an intermediate level between an application program and the Internet Protocol. It provides host-to-host connectivity at the Transport Layer of the Internet model. An application does not need to know the particular mechanisms for sending data via a link to another host, such as the required packet fragmentation on the transmission medium. At the transport layer, the protocol handles all handshaking and transmission details and presents an abstraction of the network connection to the application.

At the lower levels of the protocol stack, due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets may be lost, duplicated, or delivered out of order. TCP detects these problems, requests re-transmission of lost data, rearranges out-of-order data and even helps minimize network congestion to reduce the occurrence of the other problems. If the data still remains undelivered, the source is notified of this failure. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the receiving application. Thus, TCP abstracts the application's communication from the underlying networking details.

TCP is a reliable stream delivery service which guarantees that all bytes received will be identical with bytes sent and in the correct order. Since packet transfer by many networks is not reliable, a technique known as positive acknowledgement with re-transmission is used to guarantee reliability. This fundamental technique requires the receiver to respond with an acknowledgement message as it receives the data. The sender keeps a record of each packet it sends and maintains a timer from when the packet was sent. The sender re-transmits a packet if the timer expires before receiving the message acknowledgement. The timer is needed in case a packet gets lost or corrupted.

## **TCP Session Establishment**

In TCP connections, the host client establishes the connection with the server. A TCP connection is established in three steps. In the first step, the initiating client requests a client-to-server communication session with the server by sending a SYN request. Then in the second step, the server acknowledges the client-to-server communication session and requests a server-to-client communication session (SYN Request and ACK Response). Finally, the initiating client acknowledges the server-to-client communication session (ACK Response).

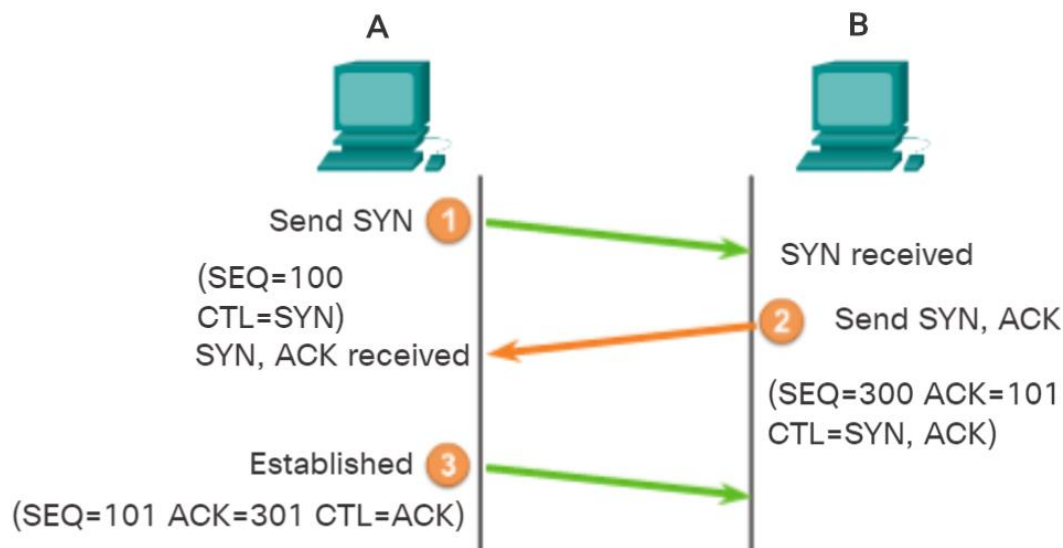


Figure 2.2 Session Establishment in TCP

## TCP's Ordered Delivery

TCP segments may arrive at their destination out of order. For the original message to be understood by the recipient, the data in these segments is reassembled into the original order. Sequence numbers are assigned in the header of each packet to achieve this goal. The sequence number represents the first data byte of the TCP segment.

During session setup, an initial sequence number (ISN) is set. This ISN represents the starting value of the bytes for this session that is transmitted to the receiving application. As data is transmitted during the session, the sequence number is incremented by the number of bytes that have been transmitted. This data byte tracking enables each segment to be uniquely identified and acknowledged. Missing segments can then be

identified. Segment sequence numbers indicate how to reassemble and reorder received segments.

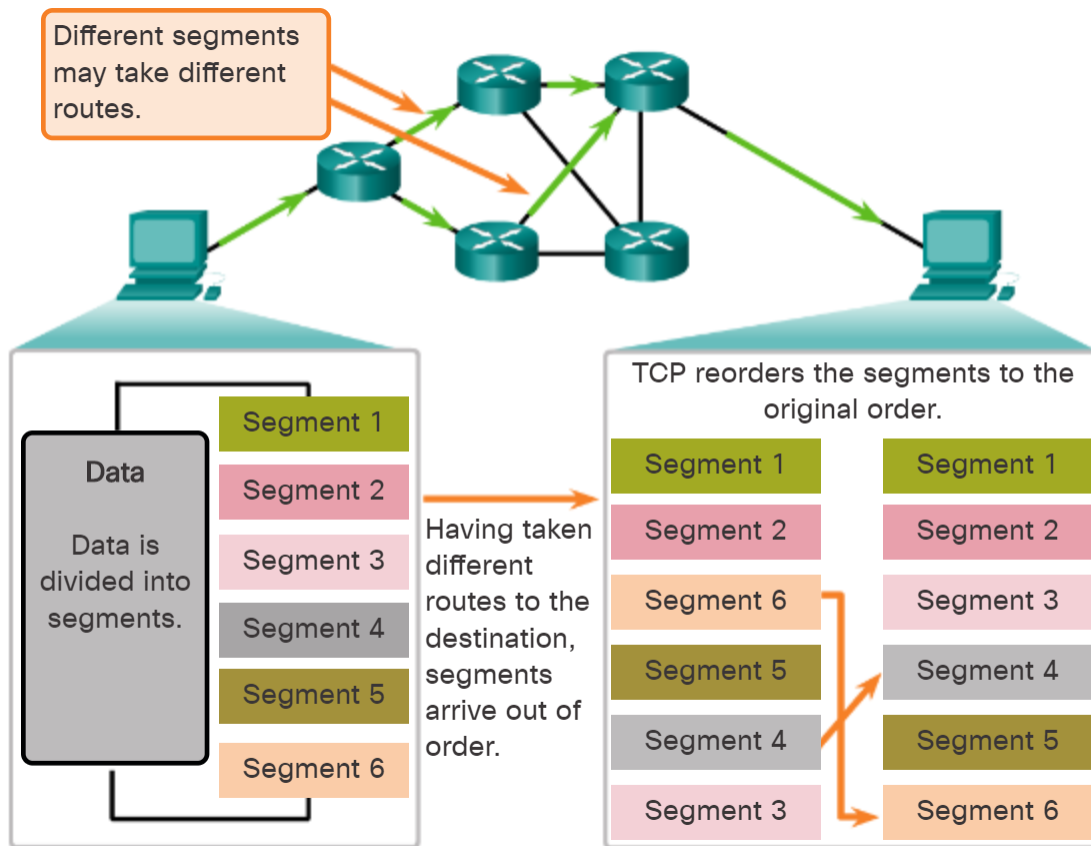


Figure 2.3 Reordering received information using Segment Sequence Numbers

The receiving TCP process places the data from a segment into a receiving buffer. Segments are placed in the proper sequence order and passed to the application layer when reassembled. Any segments that arrive with sequence numbers that are out of order are held for later processing. Then, when the segments with the missing bytes arrive, these segments are processed in order.

## Reliable Transmission and Retransmission

TCP uses a sequence number to identify each byte of data. The sequence number identifies the order of the bytes sent from each computer so that the data can be reconstructed in order, regardless of any packet reordering, or packet loss that may occur during transmission. Acknowledgements (Acks) are sent with a sequence number

by the receiver of data to tell the sender that data has been received to the specified byte. Acks do not imply that the data has been delivered to the application. They merely signify that it is now the receiver's responsibility to deliver the data.

However, in the case of packet loss, there are two techniques that identify that has occurred in order to let the sender realize that the lost packet was lost. The first technique is Duplicate Acknowledgement based retransmission. Every acknowledgment packet holds a sequence number identifying the sequence number of the next packet expected, which means that the ones before it have been delivered. If a single packet (say packet 100) in a stream is lost, then the receiver cannot acknowledge packets above 100 because it uses cumulative acks. Hence the receiver acknowledges packet 99 again on the receipt of another data packet. This duplicate acknowledgement is used as a signal for packet loss. The other technique is timeout-based retransmission. Whenever a packet is sent, the sender sets a timer that is a conservative estimate of when that packet will be acked. If the sender does not receive an ack by then, it transmits that packet again. The timer is reset every time the sender receives an acknowledgement. This means that the retransmit timer fires only when the sender has received no acknowledgement for a long time. Moreover, in case a retransmit timer has fired and still no acknowledgement is received, the next timer is set to twice the previous value (up to a certain threshold)

Furthermore, TCP provides error detection capabilities, like using sequence numbers to discard duplicate packets, insuring integrity of data by checksum computation thus adding to the reliability of TCP.

## **TCP Session Termination**

To close a connection, the Finish (FIN) control flag must be set in the segment header. To end each one-way TCP session, a two-way handshake, consisting of a FIN segment and an Acknowledgment (ACK) segment, is used. Therefore, to terminate a single conversation supported by TCP, four exchanges are needed to end both sessions.

First, when the client has no more data to send in the stream, it sends a segment with the FIN flag set. Then, the server sends an ACK to acknowledge the receipt of the FIN to terminate the session from client to server. After that, server sends a FIN to the client to terminate the server-to-client session. The client responds with an ACK to

acknowledge the FIN from the server. Finally, when all segments have been acknowledged, the session is closed.

## **Conclusion**

To end with, the messaging service application will utilize TCP and UDP since a session between the messaging server and the clients is needed and minimal packet loss is required. TCP is a highly reliable protocol that is used in the transmission of the traffic of many essential loss-intolerant services. Although it has a higher latency, but it is worth the reliability of the transmission.

## Chapter III: Client-Server-Based Messaging

### Introduction

As mentioned before, computer networks have provided great services and have facilitated many life and business aspects. Furthermore, network programming has enabled developers to configure servers and end-users to harness such services, like messaging, file sharing, mailing services, audio and video calling and more. To be more precise, socket programming [1] is mostly used to configure the behavior of the socket. A Socket is the combination of an IP address and a Port. The IP Address was already defined in the first chapter. The port is a number that is involved in the transport layer in the OSI model. Its role is to identify the type of service that the traffic is involved with, for example port 80 is involved with web browsing, and ports 20 and 21 are responsible in File Transfer Protocol. Now that the socket is defined we can get into the technical and the main ideas of the messaging service LAN application.

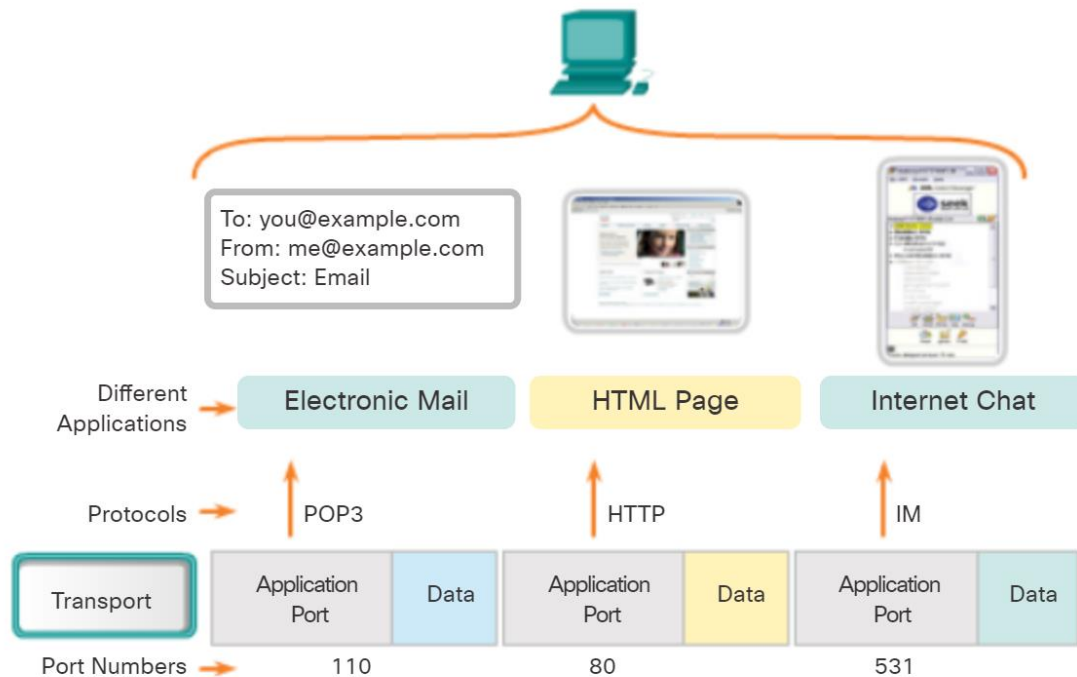
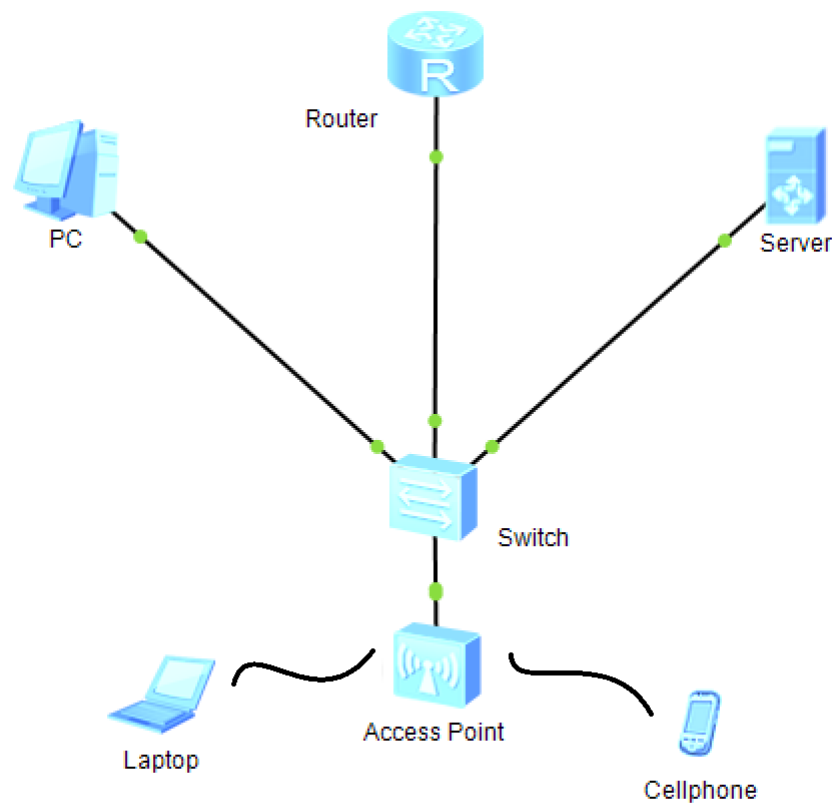


Figure 3.1 Port Addressing

## The System's Architecture

The system consists of a Client-Server architecture both found inside the same Local Area Network. The Server will handle Client connections and delivery of messages among the connected users. The main purposes of the system being found inside the same network is because of speed, since the latency is significantly lower when the traffic does not go outside the network and use the internet. Also, it is much more secure since the data being transferred over the internet is vulnerable to loss of integrity and interception.



*Figure 3.2 the system's topology*

The messaging server inside the LAN is running a Server software that accepts Clients' connections, authenticates users, and delivering messages send from a connected users to the others connected users. The clients have a client software in order to enter their credentials and send messages over the network. The Server must have a static IP address so that the clients don't lose connection to it since using a dynamic IP address



for the server is not suitable for servers as they are frequently accessed. However, the IP address of the server is not sufficient for clients to connect to it and start messaging after being authenticated. As mentioned in the chapter's introduction, Socket programming is used in this system. A Socket in computer networks is the combination of the Layer 3 IP Address and the Layer 4 Port number. Clients must use both in order to utilize the service and establish a connection with the Server on the network. The port number chosen must be one which is not a reserved one for a specific standard service like FTP or HTTP. A port number 10000 is suitable for the service that is being implemented. After knowing the socket of the server, the users can now establish a TCP session with the active server.

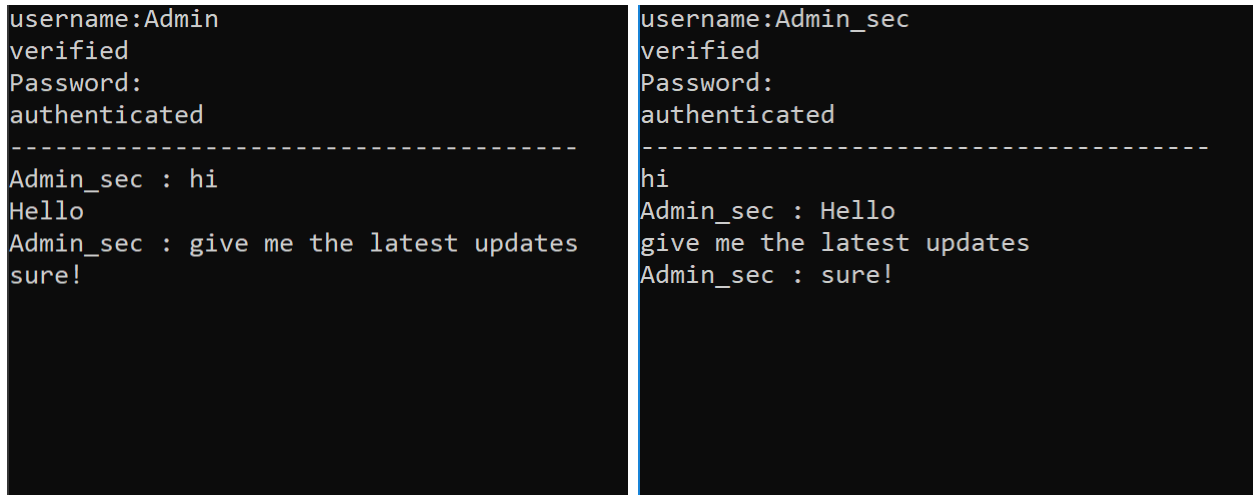
## **Connection to Server**

When the client runs the .exe application on his machine, the program automatically tries to connect to the server using the preconfigured IP address and port number. It is important that the server is up and running on the network so that clients would be able to connect. When the server runs, it binds its local IP address to the port number 10000 in order to create a server from that socket and starts the listening phase where it starts listening to connection requests and accepts them subsequently. As soon as the client tries to connect to the server, the server starts an authentication process where the client inserts the credentials and the server establishes a connection with an SQLite database and searches for entries in the users table. The passwords in the database use SHA512, therefore the password received at the server is hashed using SHA512 and compared with that of the inserted username, if found in the table.

In case the user could not be authenticated, the authentication process is restarted as a result. However, if the client has been authenticated by the server, the client is now prompted to be able to send and receive messages from and to other clients of the LAN and is added to the list of connections found on the server. A server also prints a message with the newly connected user. The client and server code are both written in python using the Socket and Threading Libraries. At the server, a Thread is launched for each connecting client. A Thread is a piece of code that runs in the background in parallel with the main code of the program. Therefore, to manage all connected users simultaneously, the server must run a thread for every user and deal with that client independently.

## Message Forwarding

After the connection and authentication process, the users can send and receive messages in the chat room. What the server does after authenticating the client, is to launch a thread for the connecting client to keep receiving and sending messages from that client. When data is received, the server sends the message to all the other connected clients except the sending one (Figure 3.3).



```
username:Admin
verified
Password:
authenticated
-----
Admin_sec : hi
Hello
Admin_sec : give me the latest updates
sure!
```

```
username:Admin_sec
verified
Password:
authenticated
-----
hi
Admin_sec : Hello
give me the latest updates
Admin_sec : sure!
```

Figure 3.3 Authentication Process and Messaging between two users

The server on the other hand, shows which users have connected and which have disconnected. The client and server codes both have a lot of exception handling techniques in order to prevent any of both from crashing. It is important to also mention that the data exchanged among clients is being switched using the MAC addresses of the source and destination device

```
Admin is connected
Admin_sec is connected
Admin_sec is disconnected
Admin_sec is disconnected
```

Figure 3.4 Server's Terminal

```

#Server Code
import socket
import threading
import sqlite3
import hashlib

class Server:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    connections = []

    def __init__(self):

        self.sock.bind(('0.0.0.0',10000))
        self.sock.listen(2)

    def authenticate(self, c):
        while True:
            try:
                data = c.recv(1024)
            except:
                c.close()
                return 0

            if "username:_" in str(data, 'utf-8'):
                global username
                username = str(data, 'utf-8').replace('username:', '')

                try:
                    dbconn = sqlite3.connect("chat_users.db")
                    dbcurs = dbconn.cursor()
                    dbcurs.execute("select Username, Password from users
where Username = '" + username + "'")
                    result = dbcurs.fetchone()
                    dbconn.commit()
                    dbconn.close()
                except:
                    try:
                        dbcurs.execute("select * from users")
                        dbconn.rollback()
                        dbconn.close()
                    except:
                        return 0

                if str(result) == "None":
                    c.send(bytes("notverified", 'utf-8'))
                    continue
                else:
                    c.send(bytes("verifieduser", 'utf-8'))

                try:
                    password = c.recv(1024)
                except:
                    c.close()
                    return 0

                password = str(password, 'utf-8').replace('password:', '')

```

```

''')
        password = hashlib.sha512(bytes(password, 'utf-8')).hexdigest()

        if password == result[1]:
            c.send(bytes("authenticated", 'utf-8'))
            self.connections.append(c)
            print(username + " is connected")
            return 1
        else:
            c.send(bytes("notauthenticated", 'utf-8'))
            continue

    def handler(self, c, a):
        auth = self.authenticate(c)
        if auth == 1:
            while True:
                try:
                    data = c.recv(1024)

                    data = username + " : " + str(data, 'utf-8')

                    for connection in self.connections:
                        if connection != c:
                            connection.send(bytes(data, 'utf-8'))
                except:
                    print(username + " is disconnected")
                    self.connections.remove(c)
                    c.close()
                    break

            else:
                c.close()
                return

    def run(self):
        while True:

            c, a = self.sock.accept()

            cThread = threading.Thread(target=self.handler, args=(c, a))
            cThread.daemon = True
            cThread.start()

server = Server()
server.run()

```

```

#Client Code
import socket
import threading
import getpass

class Client:
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    users = []

    def sendMsg(self):
        while True:
            self.sock.send(bytes(input(""), 'utf-8'))

    def __init__(self):
        try:
            self.sock.connect(("192.168.11.103", 10000))
            while True:
                username = input("username:")

                self.sock.send(bytes("username:_" + username, 'utf-8'))

                data = self.sock.recv(1024)

                data = str(data, 'utf-8')

                if data == "verifieduser":
                    print("verified")

                    password = getpass.getpass(prompt='Password: ',
stream=None)

                    self.sock.send(bytes("password:_" + password, 'utf-8'))

                    data = self.sock.recv(1024)

                    data = str(data, 'utf-8')

                    if data == "authenticated":
                        print("authenticated")
                        print("-----")
                        break

                    elif data == "notauthenticated":
                        print("Username and Password are not valid.")
                        continue

                    elif data == "notverified":
                        print("not verified")
                        continue

            iThread = threading.Thread(target=self.sendMsg)
            iThread.daemon = True
            iThread.start()

```

```
        while True:
            data = self.sock.recv(1024)
            if not data:
                break
            print(str(data, 'utf-8'))

        except:
            print("Server is currently unavailable.")

client=Client()
```

## Conclusion

Finally, this Client-Server-based Local Area Network messaging service can be used in many enterprises for exchanging updates and having a centralized server is of great benefit for fast and efficient automation of business tasks.

## Chapter IV: Conclusion

Finally, many services are being implemented over computer networks nowadays. These networks are becoming very significant in businesses and enterprises. A Client-Server-based Local Area Network Messaging service is advantageous for getting and providing updates to co-workers. It has low latency since it is free of layer 3 routing and multiple hops as well as a higher level of security since the traffic does not go over the internet and a centralized server insures a lower load on clients' machines. This application can be further extended or improved for file sharing and much more features.

## References

- [1] David M Beazley, “Python Network Programming”, Edition “Thu Jun 17 2010”
- [2] Peter L Dordal, “An Introduction to Computer Networks”, Edition Release 1.9.10
- [3] Chris Hoffman, “What’s the Difference Between TCP and UDP?”, URL” [howtogeek.com/190014/htg-explains-what-is-the-difference-between-tcp-and-udp](http://howtogeek.com/190014/htg-explains-what-is-the-difference-between-tcp-and-udp)”