**Name: Nirmallya Kundu**

**Net Id: nxk161830**

**UTD Id: 2021321095**

**Topic S3: Chat Program**

**Objective: Design and program (socket programming) a simple chat session application**

**Synopsis:**

1. Designing a session between two individuals on separate computers to exchange messages using UDP Datagram.
2. Connection Management and Reliable Communication using ACK on both sides.
3. Multi-party chat, allowing multiple users to chat at the same time.
4. Multi-media support, sending of Image files to other.
5. Broadcasting message, sending a message to all other users.
6. Graphical interface using Swing.

**Project Description:**

**Basic Concept:**

The simplest possible transport protocol is one that extends the host-to-host delivery service of the underlying network into a process-to-process communication service. There are likely to be many processes running on any given host, so the protocol needs to add a level of de-multiplexing, thereby allowing multiple application processes on each host to share the network. Aside from this requirement, the transport protocol adds no other functionality to the best-effort service provided by the underlying network. The Internet’s User Datagram Protocol is an example of such a transport protocol.

The approach used by UDP, is for processes to indirectly identify each other using an abstract locater, usually called a port. The basic idea is for a source process to send a message to a port and for the destination process to receive the message from a port. The header for an end-to-end protocol that implements this de-multiplexing

function typically contains an identifier (port) for both the sender (source) and the receiver (destination) of the message. The UDP port field is only 16 bits long. This means that there are up to 64K possible ports. A process is really identified by a port on particular host—a <port, host> pair. In fact, this pair constitutes the de-multiplexing key for the UDP protocol.

Typically, a client process initiates a message exchange with a server process. Once a client has contacted a server, the server knows the client’s port (from the SrcPrt field contained in the message header) and can reply to it. The real problem, therefore, is how the client learns the server’s port in the first place. A common approach is for the server to accept messages at a well-known port. That is, each

server receives its messages at some fixed port that is widely published. The client and server use the wellknown port to agree on some other port that they will use for subsequent communication, leaving the well-known port free for other clients. A client would send a message to the port mapper’s well-known port asking for the port it should use to talk to the “whatever” service, and the port mapper returns the appropriate port. Typically, a port is implemented by a message queue. When a message arrives, the protocol (e.g., UDP) appends the message to the end of the queue. The client then process the queue at its speed to empty the queue and process the DataPackets received from the queue.

(Ref: Textbook Computer Networks A Systems Approach By Larry L. Peterson and Bruce S. Davie)