Computer Networks

Rohini Naik

Recap

- Network Architecture
- Network Models

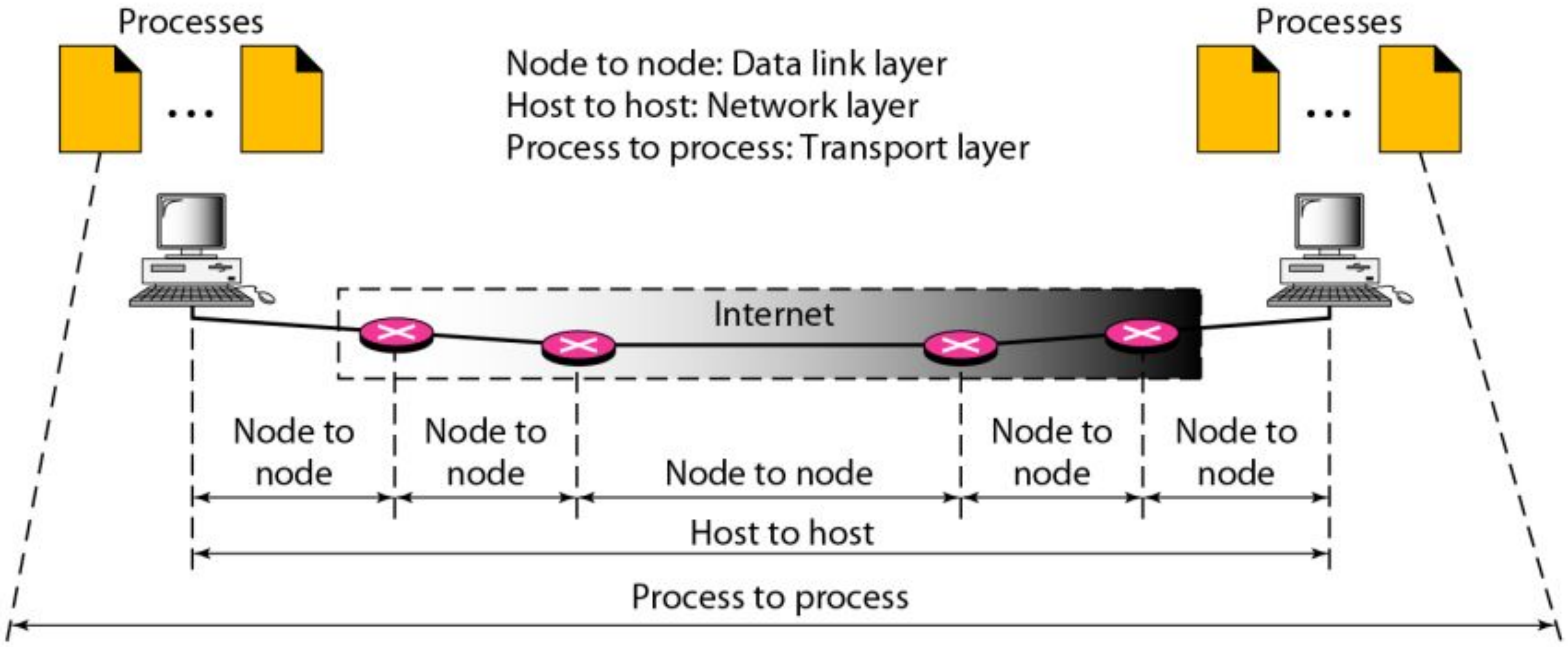
Unit 4

Transport Layer

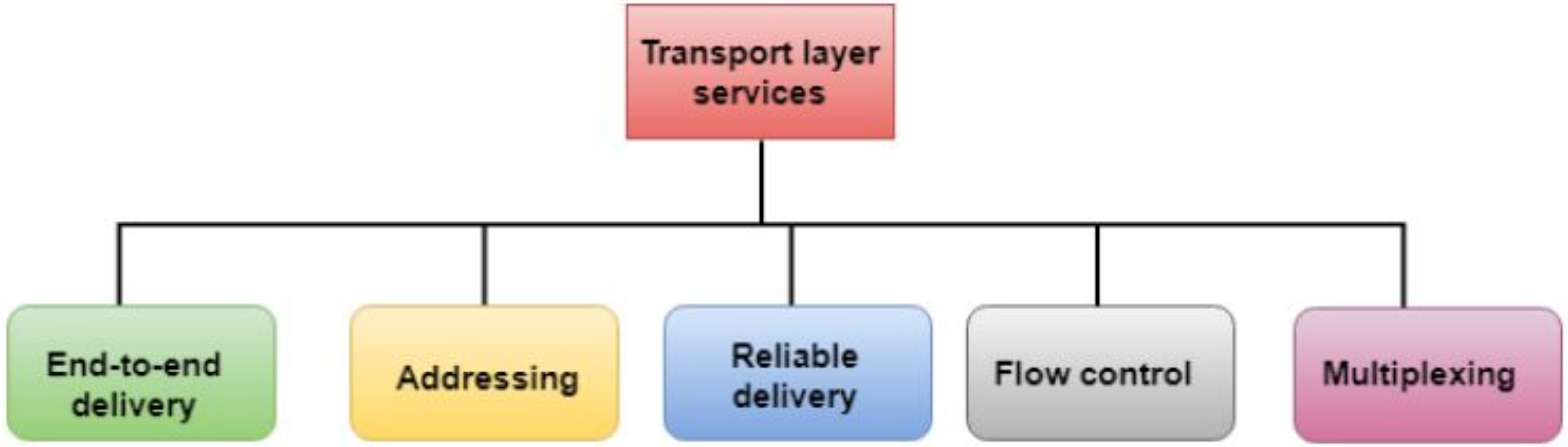
Contents

Process to Process

- The data link layer is responsible for delivery of frames between two neighboring nodes over a link. This is called node-to-node delivery
- The network layer is responsible for delivery of datagrams between two hosts. This is called host-to-host delivery
- Real communication takes place between two processes (application programs). We need process-to-process delivery
- process-to-process delivery—the delivery of a packet, part of a
- message, from one process to another. Two processes communicate in a client/server relationship

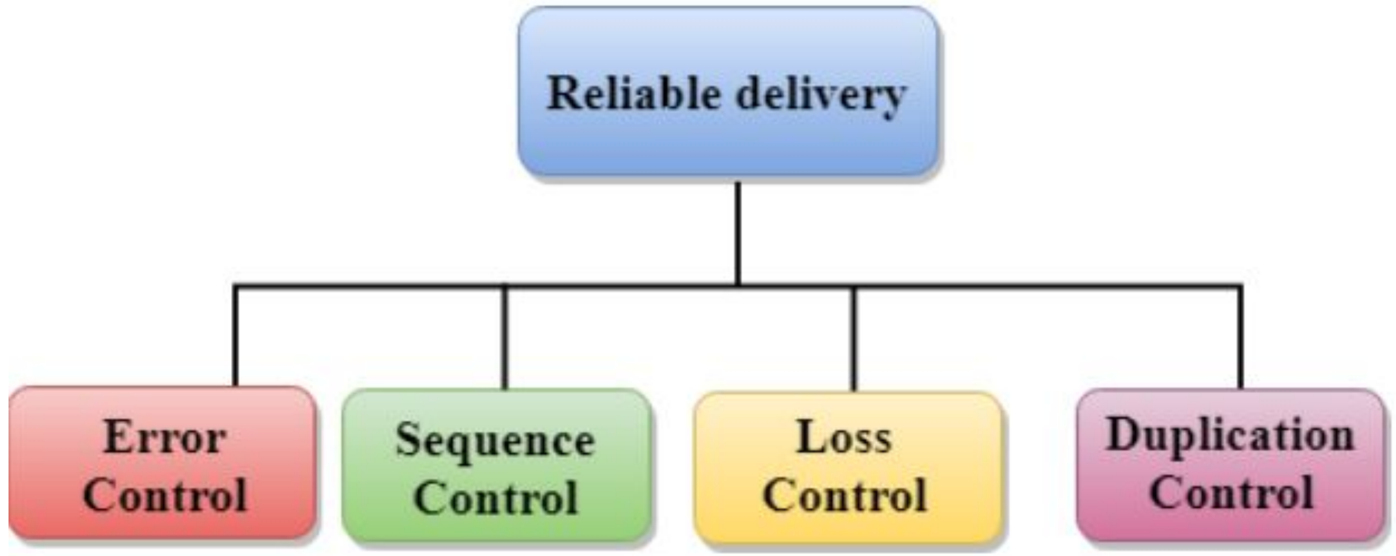

Processes
Processes
Node to node: Data link layer
Host to host: Network layer
Process to process: Transport layer
Internet
Node to node
Node to node
Node to node
Node to node
Node to node
Host to host
Process to process

Transport Layer Services



End-to-end delivery:
The transport layer transmits the entire message to the destination. Therefore, it ensures the end-to-end delivery of an entire message from a source to the destination.

Reliable delivery:
The transport layer provides reliability services by retransmitting the lost and damaged packets.



Flow Control

Flow control is used to prevent the sender from overwhelming the receiver. If the receiver is overloaded with too much data, then the receiver discards the packets and asking for the retransmission of packets. This increases network congestion and thus, reducing the system performance. The transport layer is responsible for flow control. It uses the sliding window protocol that makes the data transmission more efficient as well as it controls the flow of data so that the receiver does not become overwhelmed

Multiplexing

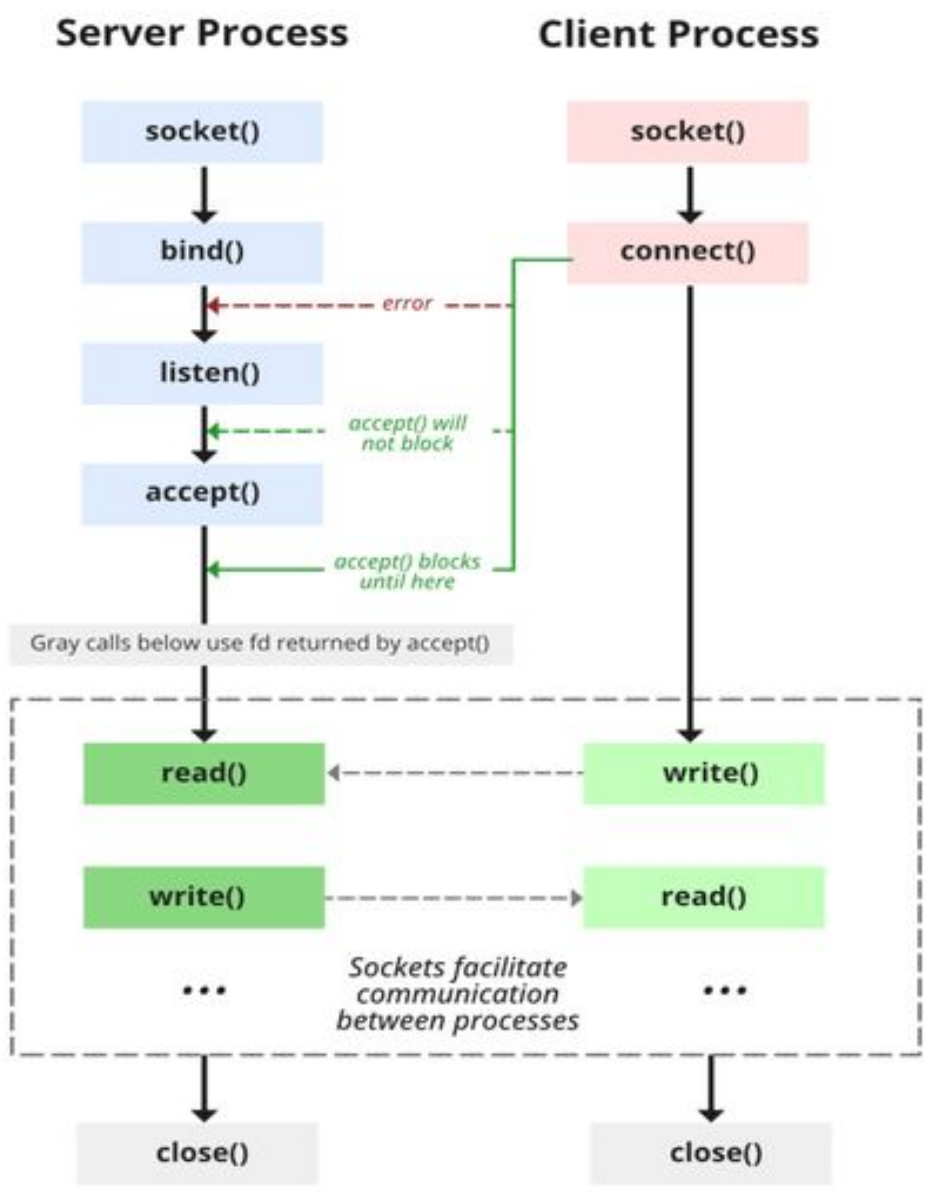
The transport layer uses the multiplexing to improve transmission efficiency. Multiplexing can occur in two ways: Upward multiplexing, Downward multiplexing.

Addressing:

- According to the layered model, the transport layer interacts with the functions of the session layer. Many protocols combine session, presentation, and application layer protocols into a single layer known as the application layer. In these cases, delivery to the session layer means the delivery to the application layer. Data generated by an application on one machine must be transmitted to the correct application on another machine. In this case, addressing is provided by the transport layer.
- The transport layer provides the user address which is specified as a station or port. The port variable represents a particular TS user of a specified station known as a Transport Service access point (TSAP). Each station has only one transport entity.
- The transport layer protocols need to know which upper-layer protocols are communicating.

Socket Programming

- Socket programming is a way of connecting two nodes on a network to communicate with each other.
- One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection.
- The server forms the listener socket while the client reaches out to the server.



State diagram for server and client model of Socket

Primitive	Meaning
SOCKET	Create a new communication endpoint
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Stages for Server:

1. **Socket creation:**
int sockfd = socket(domain, type, protocol)

sockfd: socket descriptor, an integer (like a file-handle)
domain: integer, specifies communication domain. We use AF_ LOCAL as defined in the POSIX standard for communication between processes on the same host. For communicating between processes on different hosts connected by IPV4, we use AF_INET and AF_I NET 6 for processes connected by IPV6.
type: communication type
SOCK_STREAM: TCP(reliable, connection oriented)
SOCK_DGRAM: UDP(unreliable, connectionless)
protocol: Protocol value for Internet Protocol(IP), which is 0. This is the same number which appears on protocol field in the IP header of a packet.(man protocols for more details)

2. **Setsockopt:**
This helps in manipulating options for the socket referred by the file descriptor sockfd. This is completely optional, but it helps in reuse of address and port. Prevents error such as: “address already in use”.

```
int setsockopt(int sockfd, int level, int optname,  const void *optval, socklen_t
optlen);
```

3. Bind:

```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);
```

After the creation of the socket, the bind function binds the socket to the address and port number specified in addr(custom data structure). In the example code, we bind the server to the localhost, hence we use INADDR_ANY to specify the IP address.

4. **Listen:**
int listen(int sockfd, int backlog);
It puts the server socket in a passive mode, where it waits for the client to approach the server to make a connection. The backlog, defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.

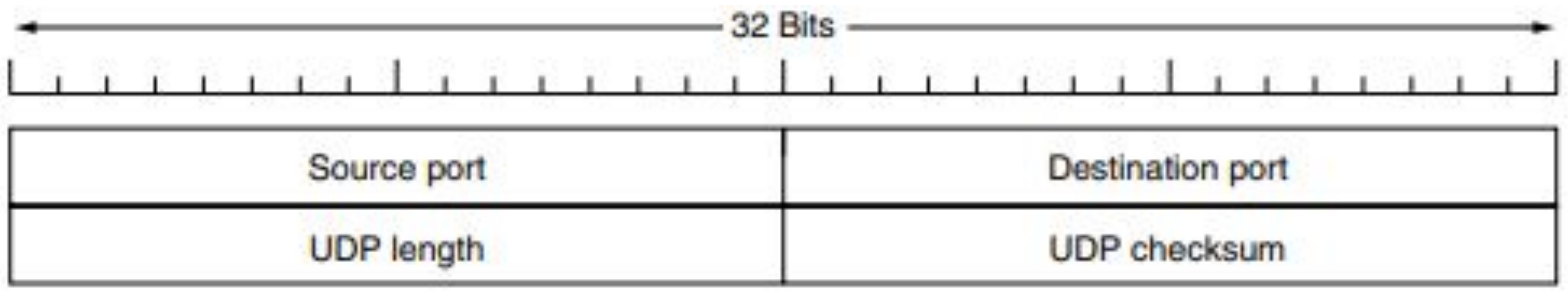
5. Accept:
int new_socket= accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to that socket. At this point, the connection is established between client and server, and they are ready to transfer data.

Stages for Client:

Socket connection: Exactly same as that of server's socket creation
Connect: The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server's address and port is specified in addr.
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

Transport layer protocols

- UDP : Doesn’t provide acknowledgement of the sent packets. Therefore, it isn’t reliable and depends on the higher layer protocols for the same. But on the other hand it is simple, scalable and comes with lesser overhead as compared to TCP. It is used in video and voice streaming.
- TCP : Provides acknowledgement of the received packets and is also reliable as it resends the lost packets. It is better than UDP but due to these features it has an additional overhead. It is used by application protocols like HTTP and FTP.



The UDP header.

Source port: The source port is primarily needed when a reply must be sent back to the source. By copying the Source port field from the incoming segment into the Destination port field of the outgoing segment

UDP length: The UDP length field includes the 8-byte header and the data. The minimum length is 8 bytes, to cover the header. The maximum length is 65,515 bytes, which is lower than the largest number that will fit in 16 bits because of the size limit on IP packets

Checksum: for extra reliability

Thank you