USER DATAGRAM PROTOCOL (UDP)

Original TCP/IP protocol suite specifies 2 protocols for transport layer: UDP & TCP.

UDP lies between application & IP layer.

UDP does process to process communication using port numbers. Provides no flow control, no ack for recd packets. If UDP detects error in recd packet, UDP drops it.

UDP is connectionless, unreliable transport protocol.

Process to Process Communication

Host to Host "

Process to Process "

To define processes we need identifiers called port numbers ranging from 0 to 65,535.

Client program defines itself with ephemeral port no.

TCP/IP uses universal port nos for servers called well known port nos.

Well-known ports 0–1023 are assigned & controlled by ICANN.

Registered Ports 1024–49151 are not assigned by or controlled by ICANN. They can only be registered with ICANN to prevent duplication.

Dynamic Ports 49152–65535 are neither controlled nor registered. They can be used as temporary or private port numbers. Original recommendation was that ephemeral port nos for clients be chosen from this range. Most systems don't follow this recommendation.

SOCKET ADDRESSES

Socket Address = IP address + Port NO.

Client & Server socket addresses are necessary for use of UDP Services.

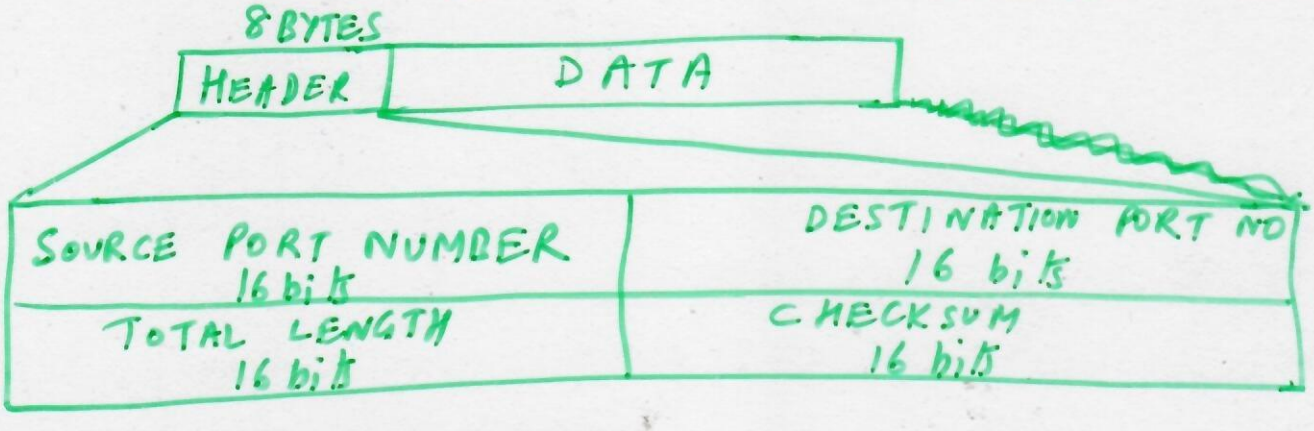
IP header contains IP addresses

UDP " " Port Numbers.

USER DATAGRAM

UDP packets called user datagrams has fixed sized header of 8 bytes.

USER DATAGRAM PACKET FORMAT



Source Port NO Range 0-65535.

Source Host	Port NO	Comments
Client	Ephemeral	(requested by process & chosen by UDP SW running on source)
Server	well Known	(server sending a response)

Destination Port NO

Dest Host	Port NO	Comments
SERVER	WELL KNOWN	(client sending a request)
CLIENT	EPHEMERAL	(Server sending a response. Server copies the ephemeral port no it has received in the request packet

Total length could be upto 65535 bytes. Since UDP user datagram is stored in IP datagram (whose total length is 65535), Header for UDP packet is smaller than 65535.

(2) The above field is not necessary, because

UDP length = IP length − IP header's length

CHECKSUM detect errors over entire user datagram (header + data).

UDP OPERATION

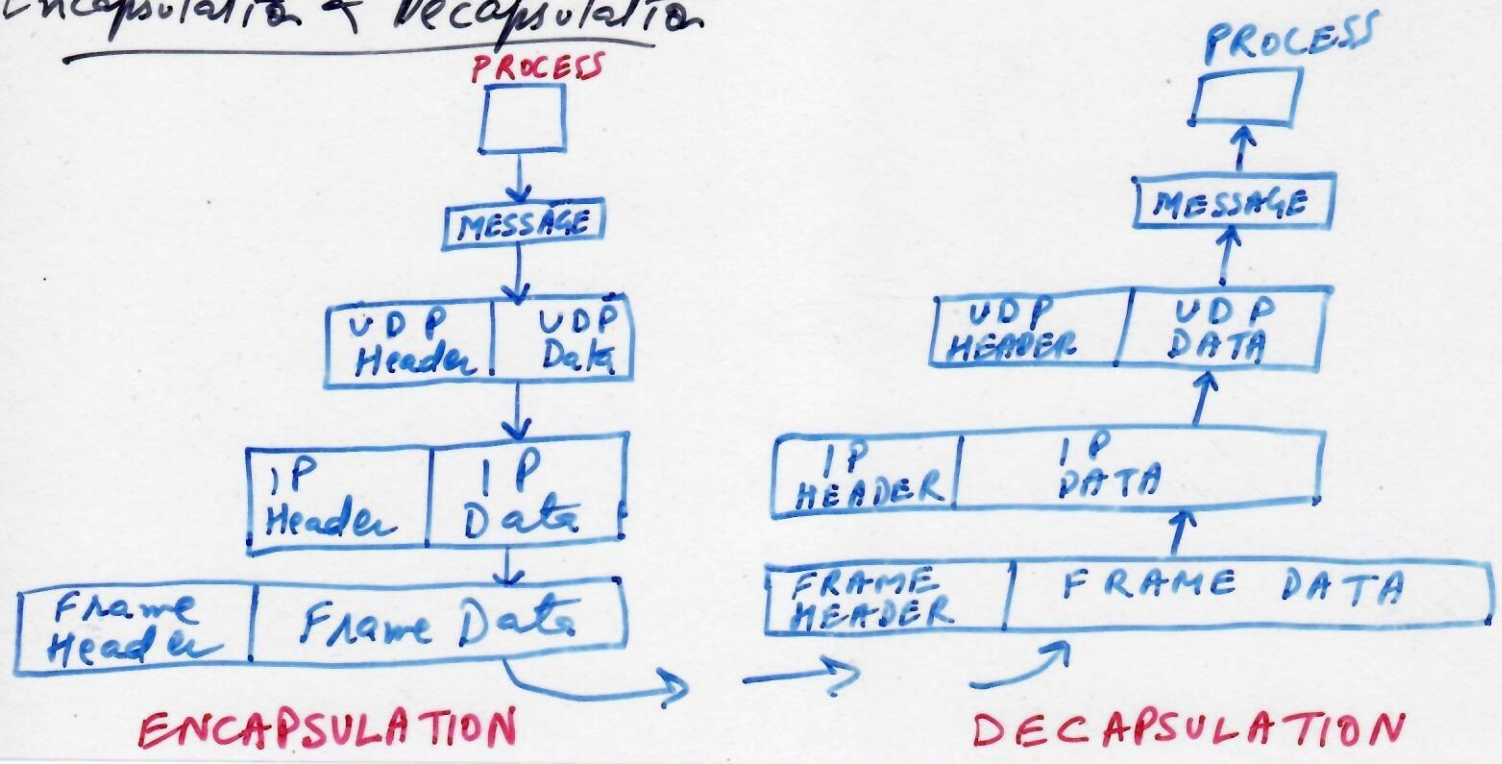
UDP provide connectionless service which implies UDP is an independent datagrams are not numbered. UDP datagrams can travel on a different path.

Processes sending short messages should use UDP (or large messages won't be chopped off by UDP into fragments)

FLOW & ERROR CONTROL

NO FLOW CONTROL. ERROR control is in the form of checksum. Sender doesn't know if message has been lost or duplicated.

Encapsulation & Decapsulation



ENCAPSULATION

1. Message passed to UDP with pair of socket addresses & length of data.
2. UDP receives data adds UDP header
3. UDP passes datagram to IP with socket addresses.
4. IP adds own header (putting value 17 in protocol field) implies data has come from UDP protocol.
5. IP datagram passed to data link layer.
6. Data link layer adds own header & passes to physical layer.
7. Physical layer encodes bits into electrical or optical signal & sends to remote machine.

DECAPSULATION

1. Physical layer decodes the signal into bits & passes to data link layer
2. Data Link Layer uses header to check the data. If it's OK header is dropped & datagram passed to IP.
3. IP software performs its own checking. If no error, header dropped, user datagram passed to UDP with sender & receiver IP addresses.
4. UDP uses checksum to check entire user datagram
5. If no error, header dropped, application data + sender socket address passed to process.

QUEUING

In UDP, queues are associated with ports.

On client side, when process starts, it requests port no from operating system. Implementations are:-

(a) Both outgoing & incoming queue with each process.
(b) Only incoming queue with a process.

Process, if communicates with multiple processes obtains ephemeral port no & one outgoing & one incoming

queue. Process ~~Function~~ Running → queues function

Process terminates → queues destroyed.

Client sends messages to outgoing queue using source port no written in request. Outgoing queue can overflow. In that case OS asks client process to wait before sending fresh messages.

Message arrives for client. UDP checks if incoming queue is there or not. Queue present → user datagram send to end of queue.

Queue absent → UDP discards user datagram & ICMP "port unreachable" message sent to server.

For Server queues mechanism is different.

Simplest form is queues with well known port no, when server starts running.

Message for server

UDP checks, incoming queue for port no specified in dest port no field.

Queue present → datagram send to end of queue.

Queue absent → datagram discarded, port unreachable message sent.

Incoming queue overflows → datagram dropped port unreachable message sent.

Message from Server

Messages to Outgoing queue using source port no specified in request.

Outgoing queue can overflow, then Operating system asks the server to wait before sending fresh messages.

Multiplexing & De multiplexing UDP perform the functionality of multiplexing & demultiplexing.

USE OF UDP

1. Useful for process requiring simple request-response communication with little concern for flow & error control. Usually not used for FTP, as it needs to send bulk data.
2. UDP suitable for process with internal flow & error control mechanisms. Ex e.g. Trivial File Transfer Protocol (TFTP) process includes flow & error control.
3. UDP suitable for multicasting. Multicasting is not there in TCP.
4. UDP used for mgmt purposes such as SNMP.
5. UDP used for some route updating protocols such as Routing Information Protocol (RIP).

TRANSMISSION CONTROL PROTOCOL (TCP)

TCP → Intermediary between application programs & N/w operations.
TCP is process to process protocol, connection oriented, uses flow & error control mechanisms at transport level.

TCP Services

Process to Process Communication

Stream Delivery Service

TCP is stream oriented protocol, unlike UDP (where each message from a process is called user datagram).

TCP delivers data → stream of bytes.
TCP receives data → " " "

TCP creates environment where 2 processes seem to be connected by an imaginary "tube" that carries data across the Internet.