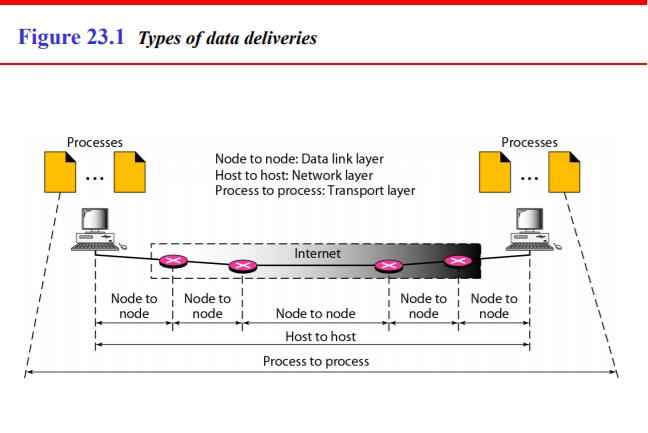
**Chapter 23: Process to Process delivery**

**Introduction**

The different types of data deliveries across a network are shown in the following figure

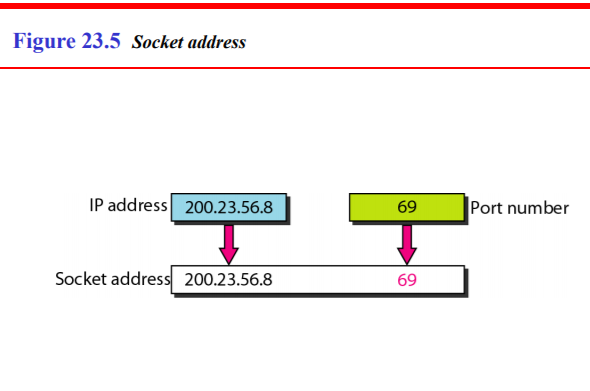


The transport layer is responsible for process process-to process delivery delivery—the delivery of a packet, packet, part of a message, message, from one process to another. Two processes communicate in a client/server relationship.

**The Client Server Paradigm**

* The client/server paradigm divides software into two categories - clients and servers.
* A client is software that initiates a connection and sends requests, whereas a server is software that listens for connections and processes requests.
* In the context of UDP programming, no actual connection is established, and UDP applications may both initiate and receive requests on the same socket.
* In the context of TCP, where connections are established between machines, the client/server paradigm is much more relevant.

**Multiplexing and Demultiplexing**

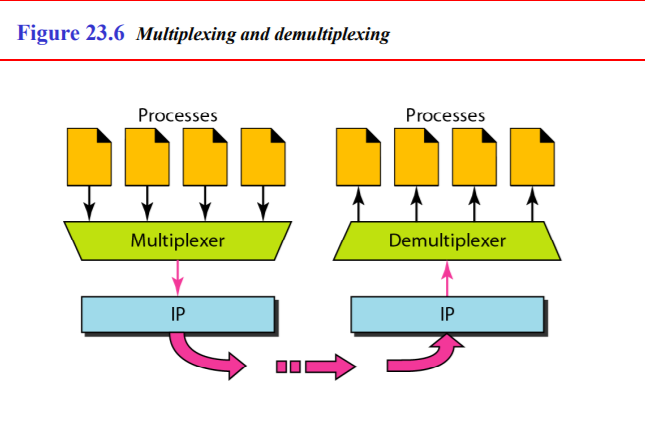
* Multiplexing and Demultiplexing services are provided in almost every protocol architecture ever designed.
* UDP and TCP perform the demultiplexing and multiplexing jobs by including two special fields in the segment headers: the source port number field and the destination port number field.
* **Multiplexing –** Gathering data from multiple application processes of sender, enveloping that data with header and sending them as a whole to the intended receiver is called as multiplexing.
* **Demultiplexing –** Delivering received segments at receiver side to the correct app layer processes is called as demultiplexing.
* **Socket Address – Combination of IP address and Port address of source or destination.**

* Port Numbering:

1. Well known ports: 0-1023
2. Unregistered Ports: 1024 – 49151
3. Dynamic/Private Ports: 49152 - 65535

* **Multiplexing and Demultiplexing Process**

1. sending data from the sender to the receiver, sender must know the socket address of destination to which he wants to transfer the data.
2. The data that is to be sent is wrapped up with the source IP address, destination IP address, source port no, destination port number and sent as a single message to the receiver. This process is called multiplexing.
3. At the destination, received data is unwrapped and the contents are sent to appropriate application by looking to the destination the port number. This process is called demultiplexing.



**TCP and UDP: Similarities and Differences**

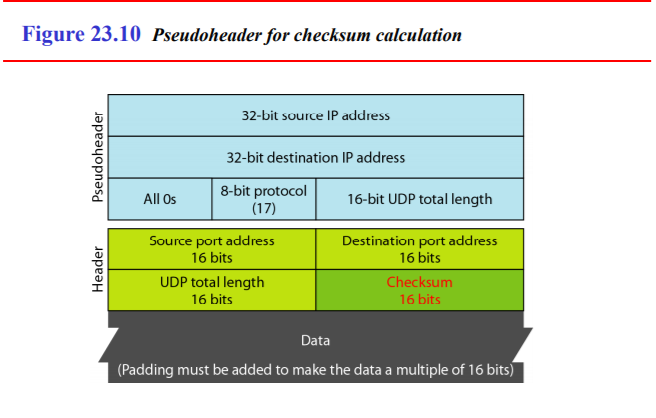
Similarities

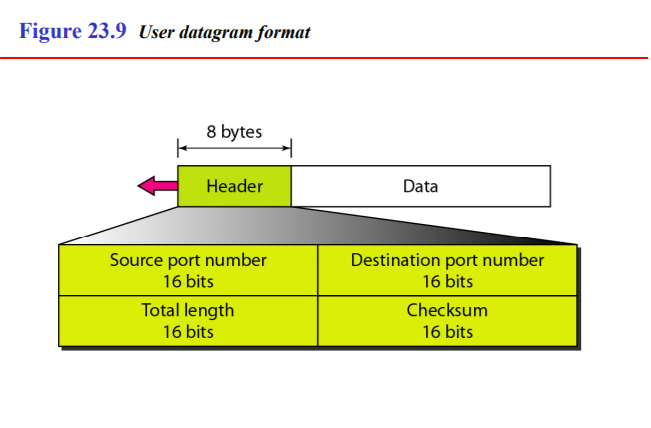
Differences

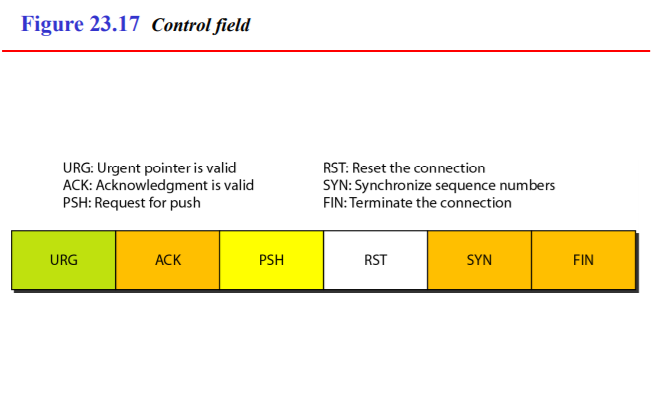
|  |  |  |
| --- | --- | --- |
| **Criteria** | **Transmission Control Protocol** | **User Datagram Protocol** |
| Connection | Connection – oriented: establishes a connection b/w sender and receiver before transmission | Connection- less: does not establish a connection b/w sender and receiver before transmission. |
| Reliability | Reliable – data is guaranteed to be delivered to destination. | Unreliable - data is not guaranteed to be delivered to destination. |
| Flow Control | Flow Control is mandatory - uses a flow control mechanism. Every time a packet is received, a message is sent to the sender with the value of the current receive window. | Flow Control is not mandatory - does not provide flow control. With UDP, packets arrive in a continuous stream or they are dropped. |
| Checksum | mandatory | not mandatory |
| Ordering | Does ordering and sequencing to guarantee that packets sent from a server will be delivered to the client in the same order they were sent | No ordering - UDP sends packets in any order. |
| Speed | Slower than UDP | Faster than TCP |
| Packet Format | Called a segment | Called a datagram |
| Acknowledgement | Acknowledgement Segments | No acknowledgement |
| Handshake | SYN, SYN+ACK, ACK | No Handshake |
| Encapsulation and Decapsulation | Required | Required |
| Multiplexing and Demultiplexing | Required | Required |
| Function | Required | Required |
| Functioning | Transmission Control in transport layer | Transmission Control in transport layer |

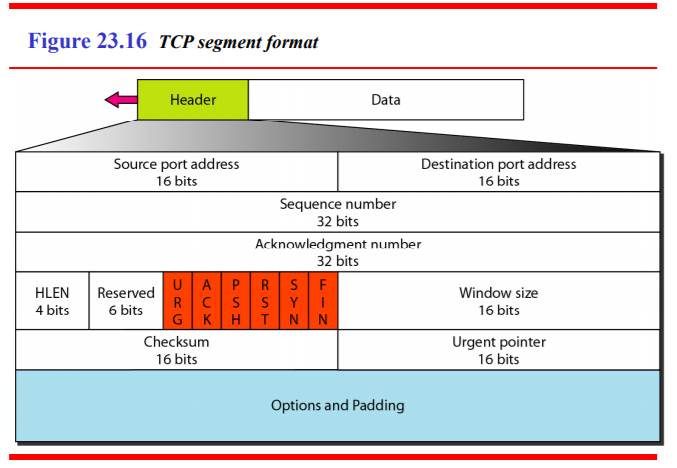
**TCP and UDP: Packet Formats and Important Diagrams**

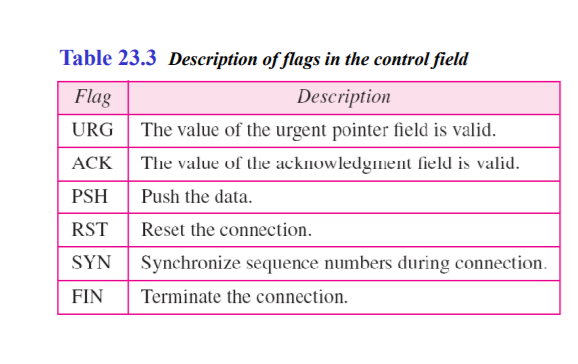
**Note** - To write theory about TCP and/or UDP just elaborate on the points given in the similarities and differences of either.

**UDP: Datagram**



**TCP: Segment**

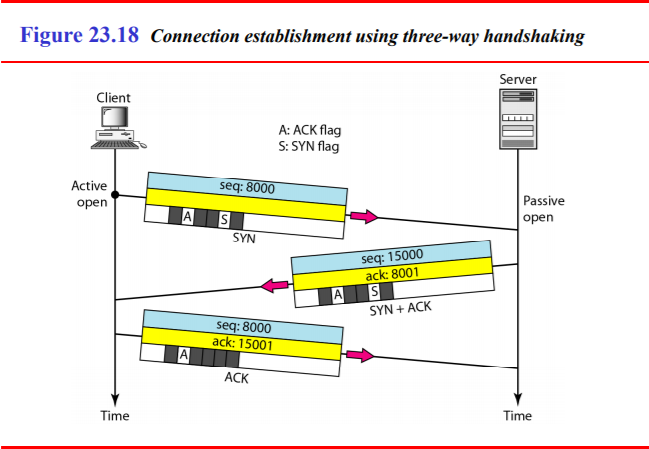
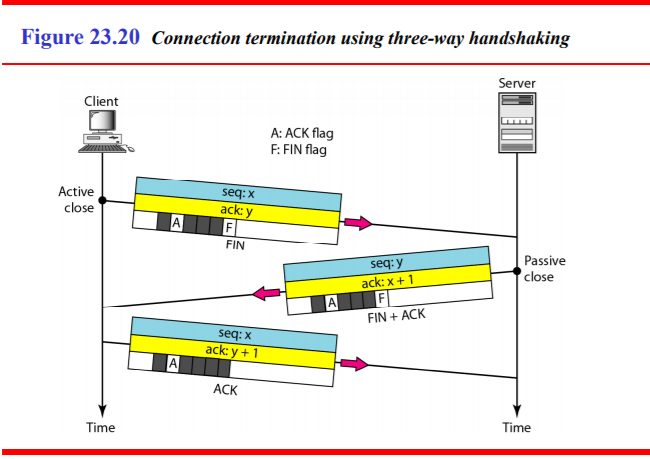
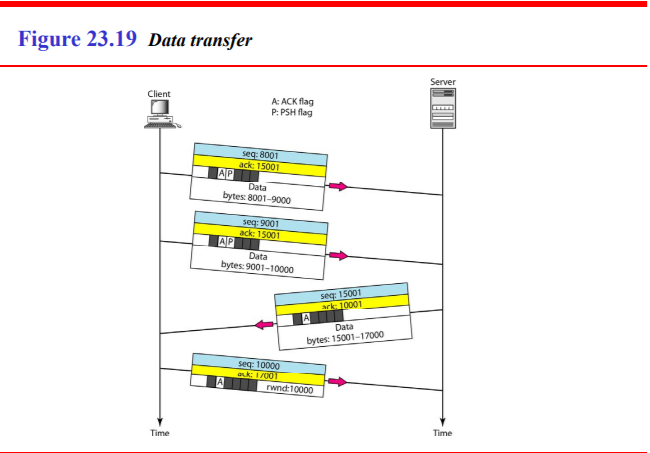




**TCP: Handshake**

Flags involved in TCP handshake are – SYN, SYN+ACK, PSH, ACK, FIN, FIN+ACK

* For connection establishment - SYN, SYN+ACK
* For data transfer – PSH + ACK
* For connection termination – FIN, FIN+ACK



|  |  |  |
| --- | --- | --- |
| **Flag** | **Carry Data** | **Sequence No’s Consumed** |
| **SYN** | Cannot | Consumes 1 |
| **SYN + ACK** | Cannot | Consumes 1 |
| **ACK** | Can carry data | When carrying data, consumes 0 sequence nos |
| **FIN** | Can carry data | Consumes 1 sequence no when carrying no data |
| **FIN + ACK** | Can carry data | Consumes 1 sequence no when not carrying data |

**TCP: Sliding Window**

* A sliding window is used to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed with data.
* TCP sliding y windows are byte-oriented.
* Some points about TCP sliding windows:

1. The size of the window is the lesser of rwnd (receiver window size) and cwnd (client window size).
2. The source does not have to send a full window’s worth of data.
3. The window can be opened or closed by the receiver, but should not be shrunk.
4. The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.
5. The receiver can temporarily shut down the window; the sender, however, can always send a segment of 1 byte after the window is shut 