Transport Layer (part 1)

1 Transport-layer services

1.1 Transport services and protocols

- Provide logical communication between app processes running on different hosts, in contrast the network layer just communicates between hosts, not processes on hosts
- Transport protocols run in end systems
  - Send side: breaks up app messages into segments, passes to network layer
  - Receive side: reassembles segments into messages, passes to app layer
- More than one transport protocol available to apps:
  - Internet: TCP and UDP

1.2 Transport vs network layer

Definition: Network layer

Logical communication between **hosts**

Definition: Transport layer

Logical communication between **processes**. Relies on and enhances network layer services

1.3 Internet transport-layer protocols

TCP (Transmission Control Protocol):

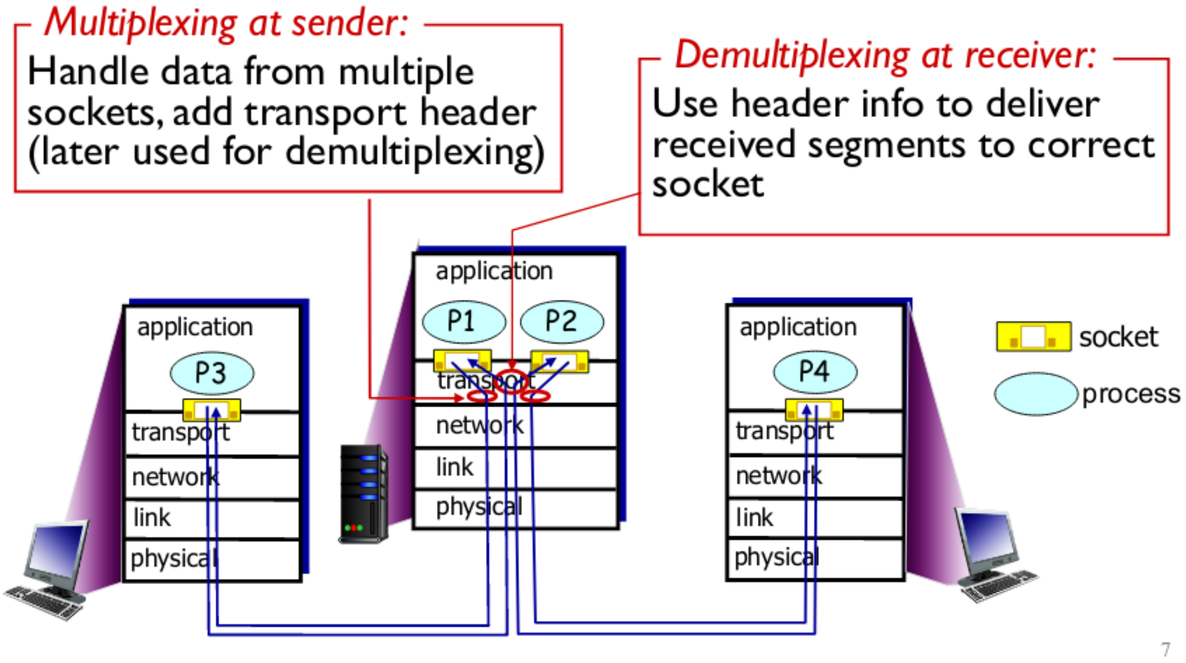
- Reliable, in-order delivery
- Congestion control
- Flow control, ack., timer
- Connection setup

UDP(User Datagram Protocol):

- Unreliable, unordered delivery
- No-frills extension of ”best-effort” IP
- Services not available
  - Delay guarantees
  - Bandwidth guarantees

TCP and UDP extend IP delivery service between hosts to delivery service between processes → transport layer multiplexing and demultiplexing

2 Multiplexing and demultiplexing



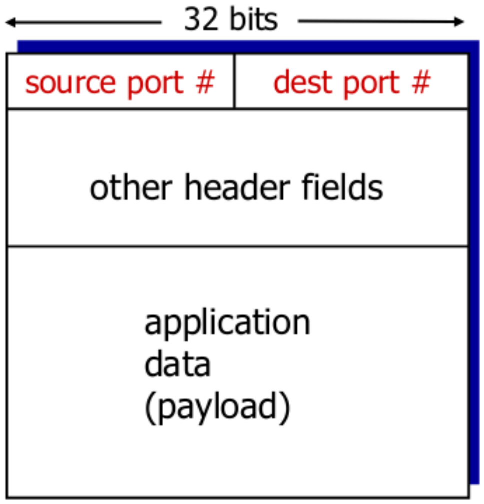
Multiplexing at sender:
Handle data from multiple sockets, add transport header (later used for demultiplexing)

Demultiplexing at receiver:
Use header info to deliver received segments to correct socket

Important: Groups of data

In the network layer it is called a packet
In the transport layer it is called a segment

- Host receives IP datagrams
- Each datagram has source IP address, destination IP address
- Each datagram carries one transport-layer segment
- Each segment has source, destination port number
- Host uses IP addresses and port numbers to direct segment to appropriate socket
- Each socket has a unique identifier

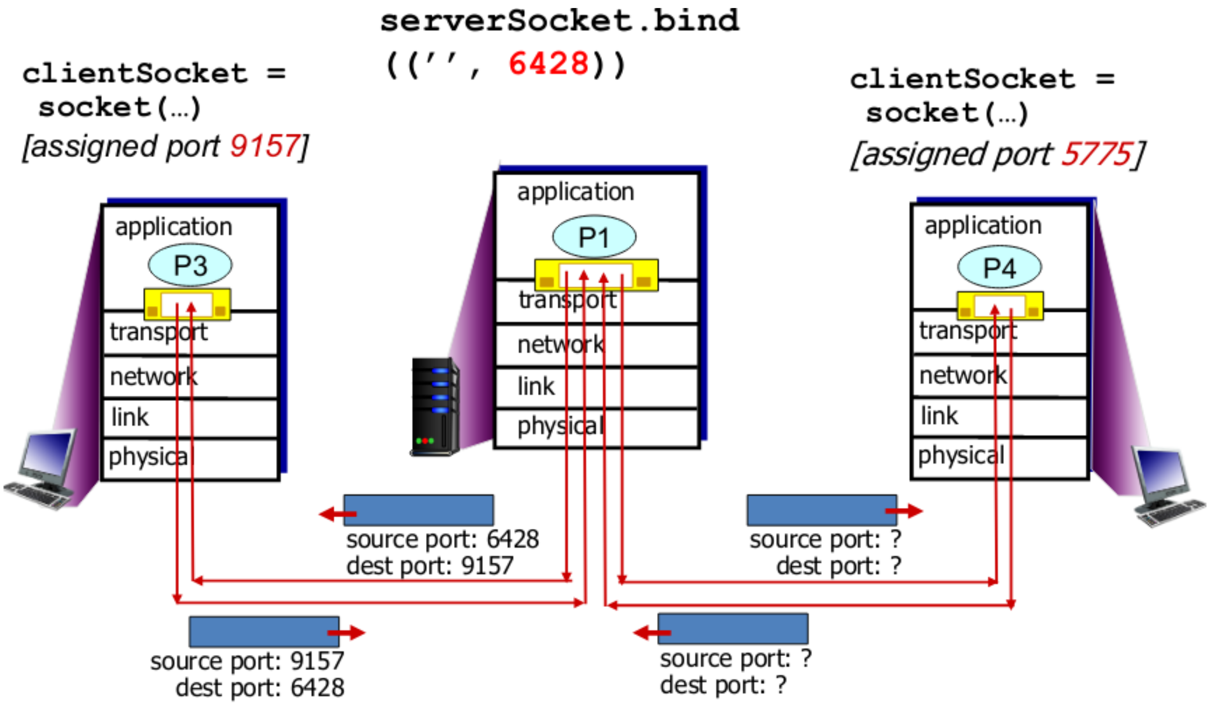


TCP/UDP segment format

2.1 Connectionless multiplexing and demultiplexing

- All sockets have host-local port #
- Assigned automatically, or via `bind()`
- `serverSocket,bind((ip, port))`
- When host receives UDP segment:
  - Checks destination port # in segment
  - Directs UDP segment to socket with that port #

If two UDP segments have different source IP addresses and/or source port numbers but same dest IP and port #, they will be directed to same process via same process via same socket as dest



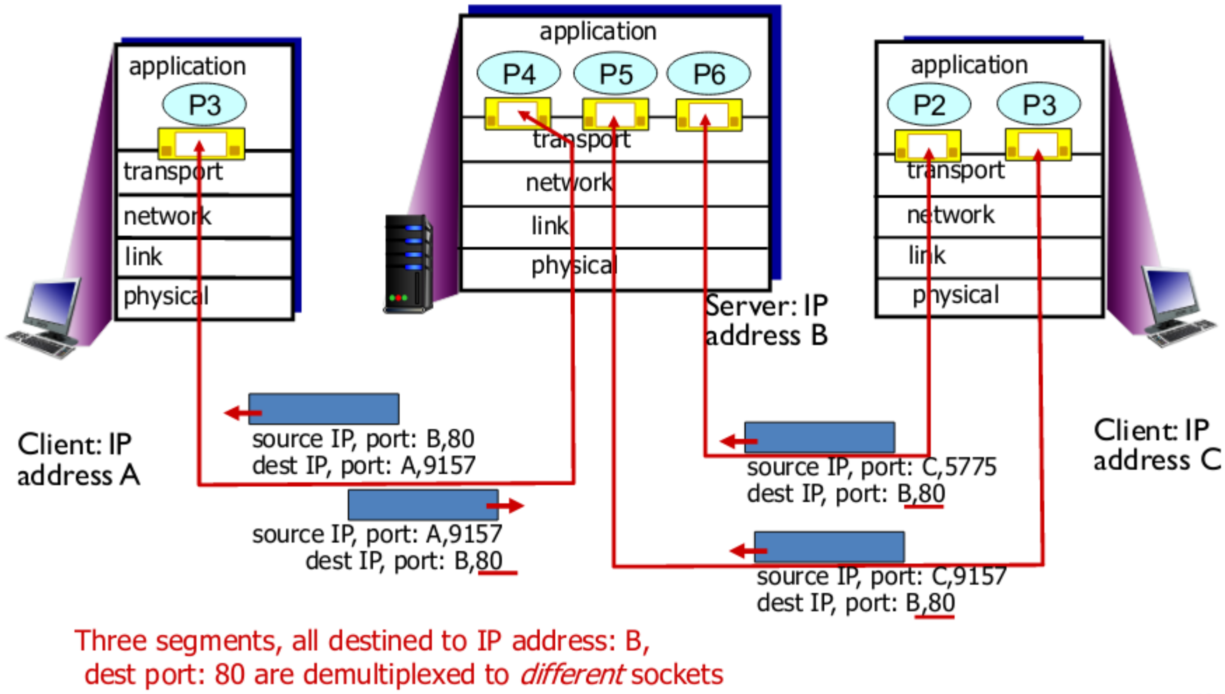
2.2 Connection-oriented multiplexing and demultiplexing

TCP socket identified by 4-tuple

- Source IP address
- Source port number
- Destination IP address
- Destination port number

Demux: receiver used all four values to direct segment to appropriate socket
Server host may support many simultaneous TCP sockets:

- Each socket identified by its own 4-tuple
- Two arriving TCP segments with different source IP/ #Port will be directed to two different sockets



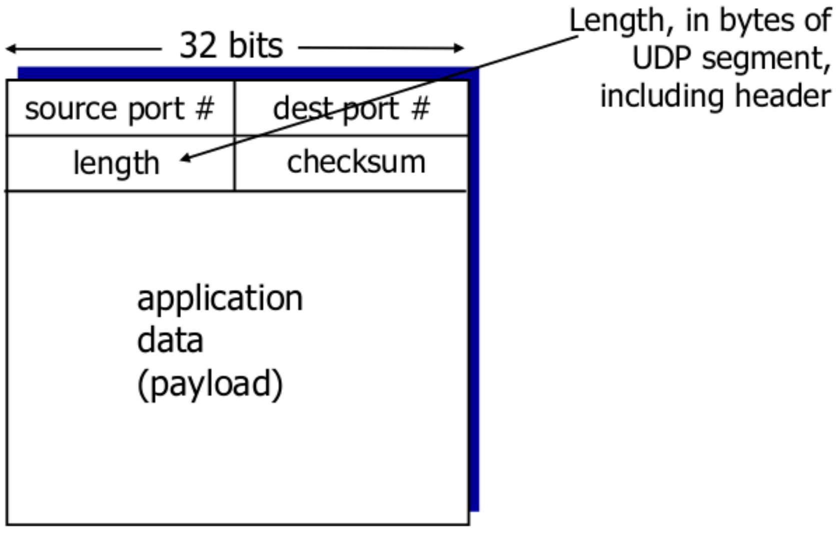
Important: TCP Sockets

With TCP we have a different socket for each connection

3 Connectionless transport: UDP

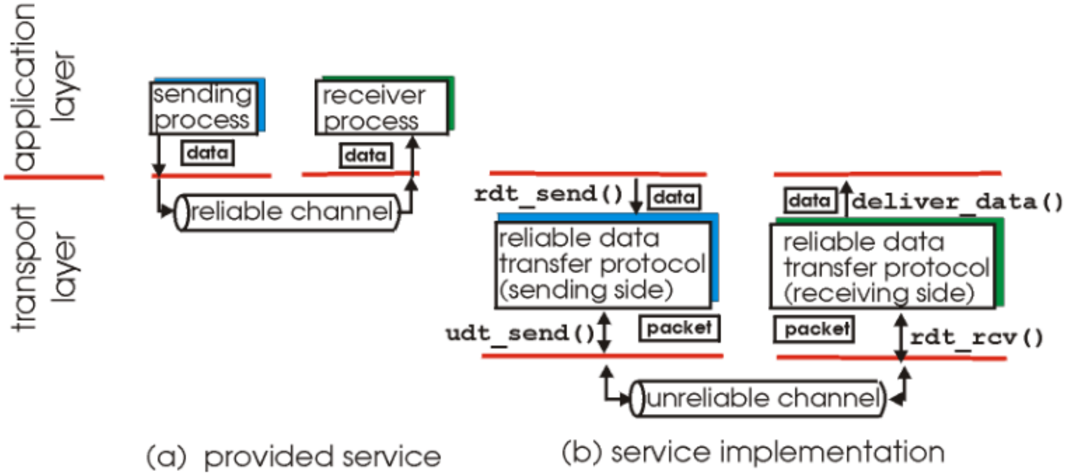
- ”No frills”, ”bare bones” internet transport protocol
- ”Best effort” service, UDP segments may be
  - Lost
  - Delivered out-of-order to app
- Connectionless:
  - No handshaking between sender/receiver
  - Each UDP segment handled independently of others
- UDP use:
  - Streaming multimedia apps (loss tolerant, rate sensitive)
- Reliable transfer over UDP
  - Add reliability at application layer
  - Application-specific error recovery

3.1 Segment Header

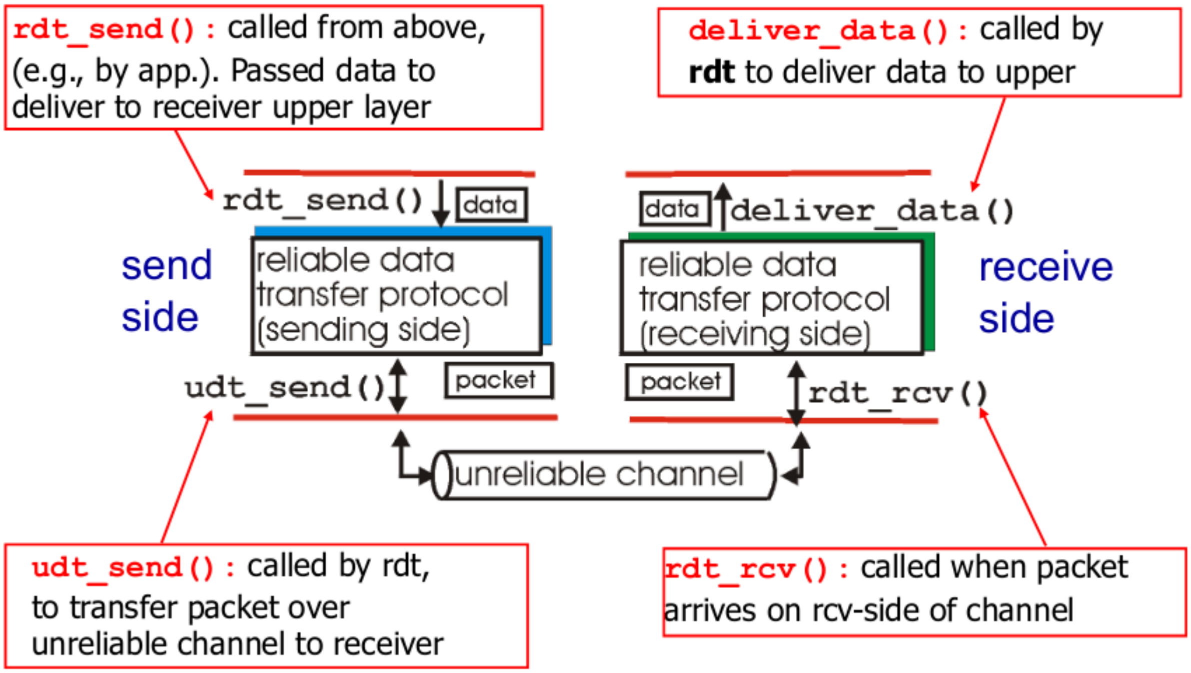

32 bits
Length, in bytes of UDP segment, including header
source port #
dest port #
length
checksum
application data (payload)

UDP segment format

4 Principles of reliable data transfer



Note that reliable data transfer protocol is not a standard, it is just for us to look at academically



rdt_send() : called from above, (e.g., by app.). Passed data to deliver to receiver upper layer

deliver_data() : called by **rdt** to deliver data to upper

udt_send() : called by rdt, to transfer packet over unreliable channel to receiver

rdt_rcv() : called when packet arrives on rcv-side of channel