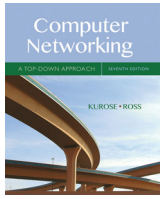
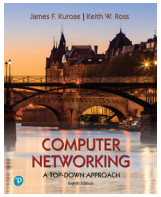
Chapter 3
Transport Layer

Courtesy to the textbooks' authors and Pearson Addison-Wesley because many slides are adapted from the following textbooks and their associated slides.



Jim Kurose, Keith Ross, "Computer Networking: A Top Down Approach", 7th Edition, Pearson, 2016.



Jim Kurose, Keith Ross, "Computer Networking: A Top Down Approach", 8th Edition, Pearson, 2020.

Transport layer: overview

Our goal:

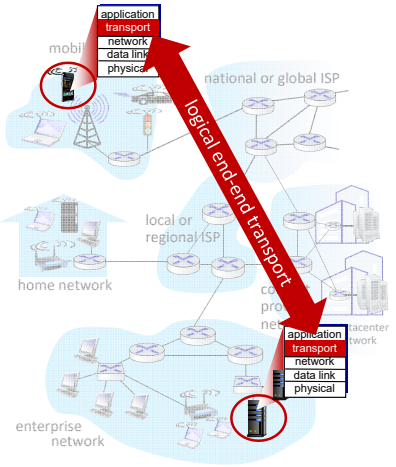
- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - TCP congestion control

Transport layer: roadmap

- Review of transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

Transport services and protocols

- provide *logical communication* between application processes running on different hosts
- transport protocols actions in end systems:
  - sender: breaks application messages into *segments*, passes to network layer
  - receiver: reassembles segments into messages, passes to application layer
- ≥2 transport protocols available to Internet applications
  - TCP, UDP



Transport vs. network layer services and protocols

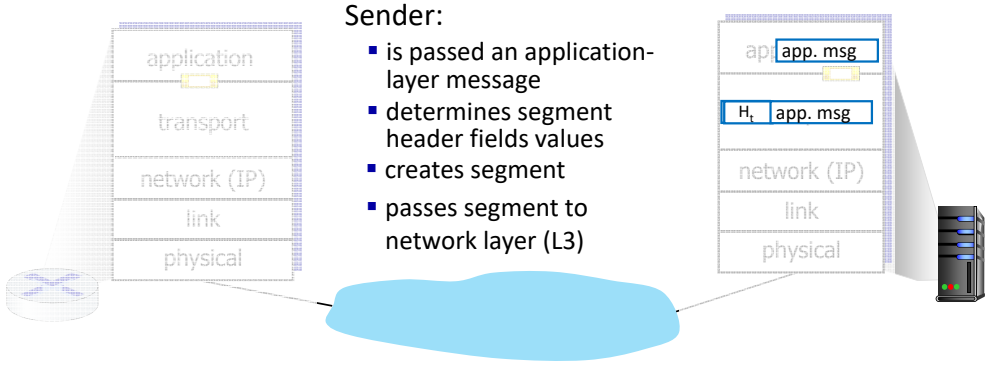
- network layer: logical communication between *hosts*
- transport layer*:* logical communication between *processes*
  - relies on network layer services
  - enhances network layer services

household analogy:

There are multiple persons (Alice, Bob, …) in a house:

- hosts = houses
- processes = persons
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house persons
- network-layer protocol = postal service

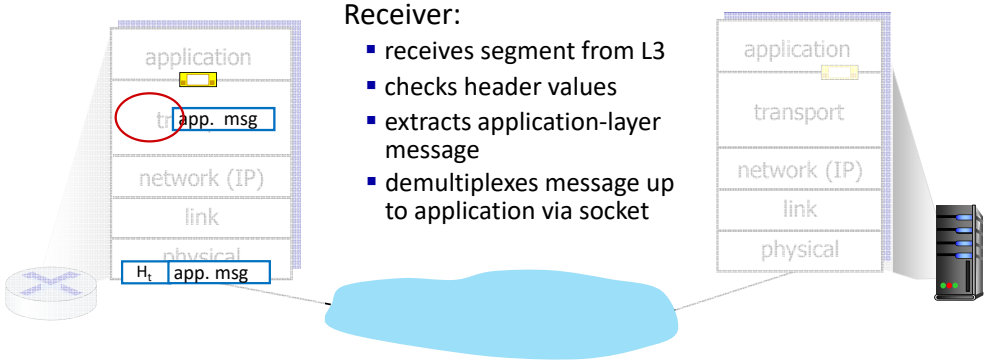
Transport Layer Actions



Sender:

- is passed an application-layer message
- determines segment header fields values
- creates segment
- passes segment to network layer (L3)

Transport Layer Actions

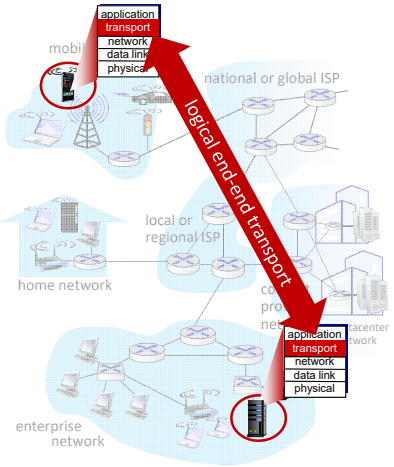


Receiver:

- receives segment from L3
- checks header values
- extracts application-layer message
- demultiplexes message up to application via socket

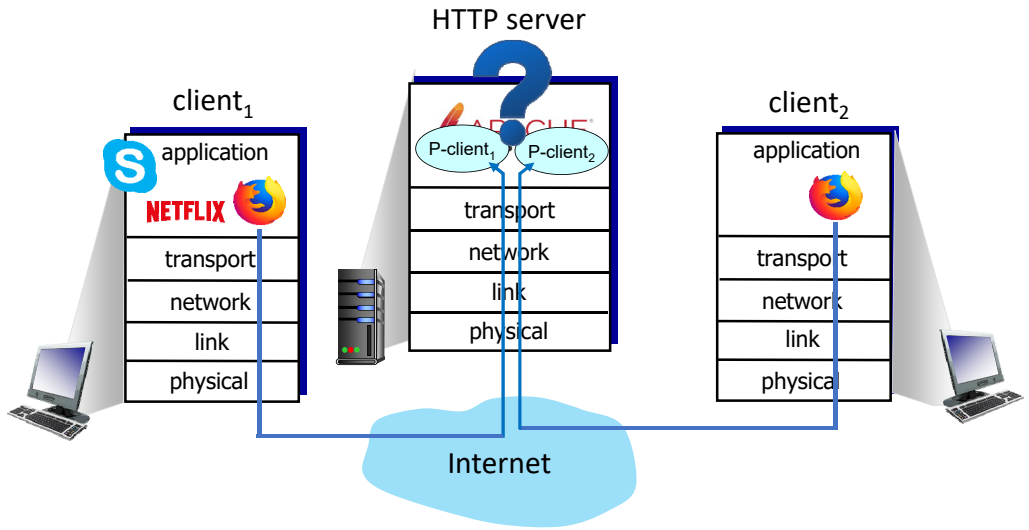
Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



Chapter 3: roadmap

- Review of transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality



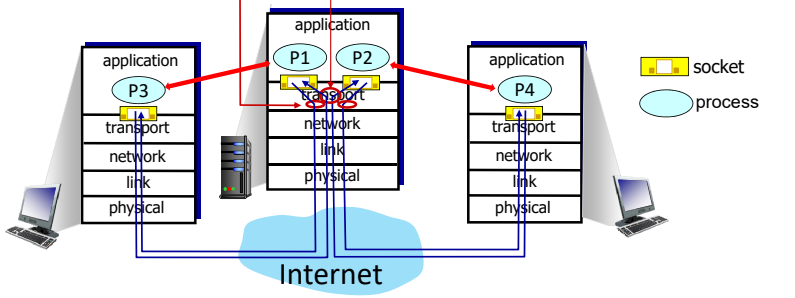
Multiplexing/demultiplexing

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

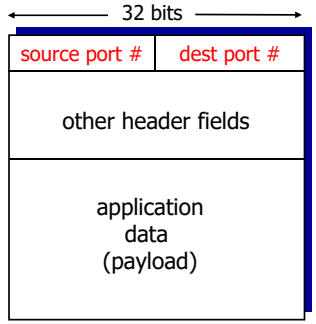
demultiplexing at receiver:

use header info to deliver received segments to correct socket



How demultiplexing works

- 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* & *port numbers* to direct segment to appropriate socket



TCP/UDP segment format

Connectionless (UDP) demultiplexing

Recall:

- when creating socket, must specify *host-local* port #:

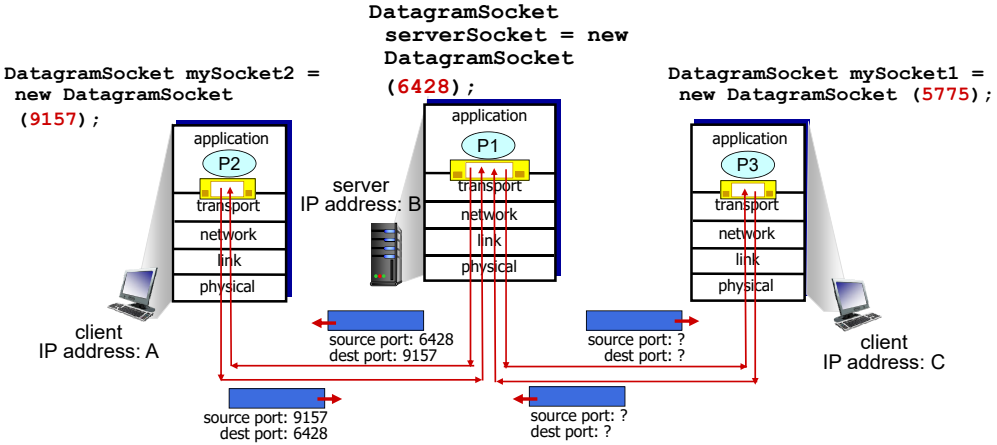
```
DatagramSocket mySocket1
 = new DatagramSocket(12534);
```

- when app at the source side creates datagram to send into UDP socket, it must specify
  - destination IP address
  - destination port #

- when receiving host receives UDP segment:
  - checks destination port # in segment
  - directs UDP segment to socket with that port #

UDP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

Connectionless demultiplexing: an example

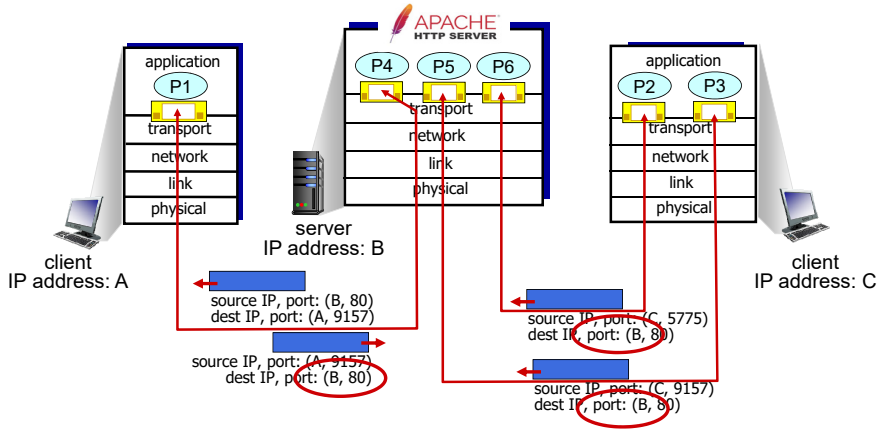


Connection-oriented (TCP) demultiplexing

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values (4-tuple)* to direct segment to appropriate socket

- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

Connection-oriented demultiplexing: example



Three segments, all destined to (B,80) are demultiplexed to *different* sockets

Summary

- Multiplexing and demultiplexing are based on header field values
- **UDP:** destination host demultiplexes segments using destination port number (only)
  - a UDP socket is uniquely identified by (dest IP, dest port #)
- **TCP:** demultiplexing using 4-tuple
  - IP addresses and port numbers of source and destination
- Multiplexing/demultiplexing happen at *all* layers