Transport Layer

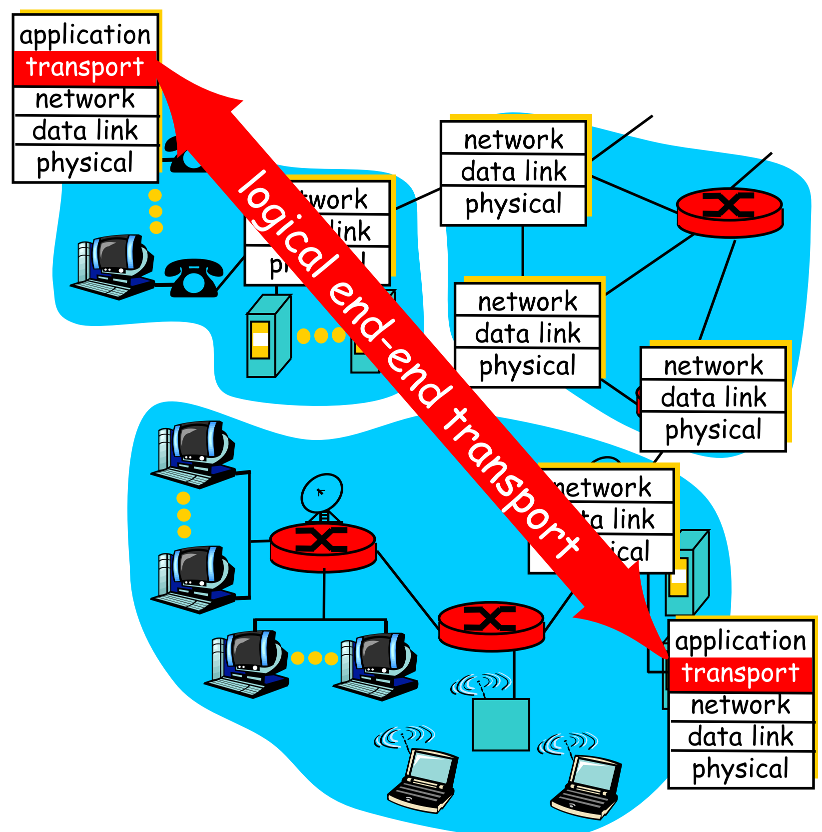
Instructor: C. Pu (Ph.D., Assistant Professor)

Lecture 08

puc@marshall.edu

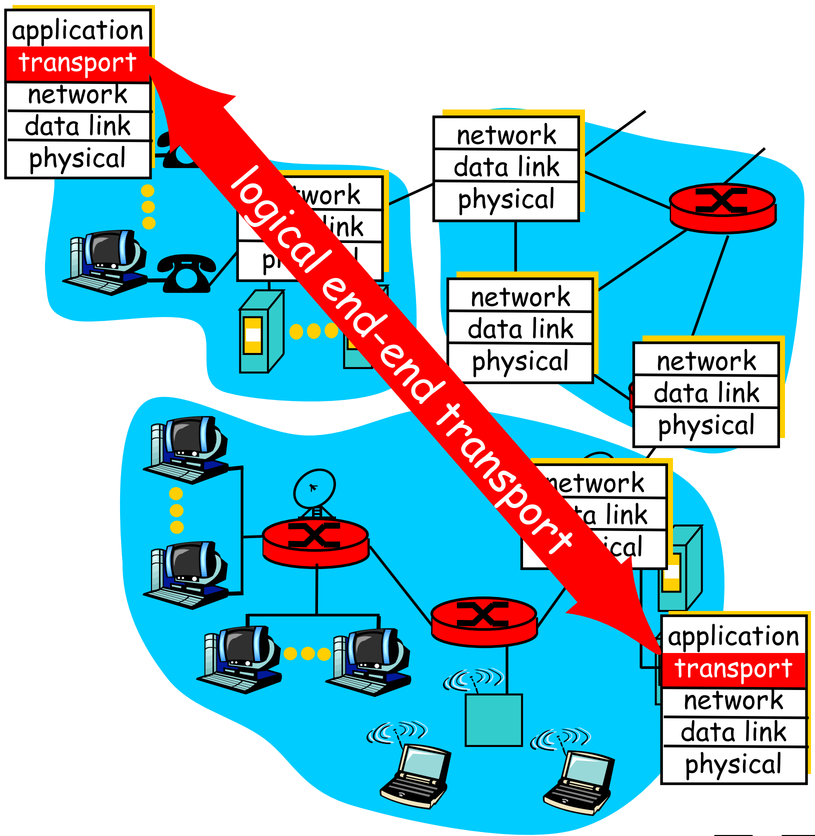
Transport Services and Protocols

- **transport-layer protocol** provides *logical communication* between *app. processes* running on different hosts
- *logical communication*: (from application's perspective)
  - seems like the hosts running the *processes* were directly connected
  - in reality, connected via numerous routers and various link types
- *app. processes* use the *logical communication* provided by **transport layer** to send messages to each other
  - free from the worry of the details of the physical infrastructure carrying the messages



Transport Services and Protocols

- transport-layer protocols run in ***end systems***, not in network routers
  - sender
    - breaks app. messages into **segments**, then passes to network layer
  - network router
    - do not examine **segments**
  - receiver
    - reassembles **segments** into messages, then passes to app. layer
- more than one transport protocol available to app.
  - Internet: TCP and UDP



Transport Vs. Network Layer

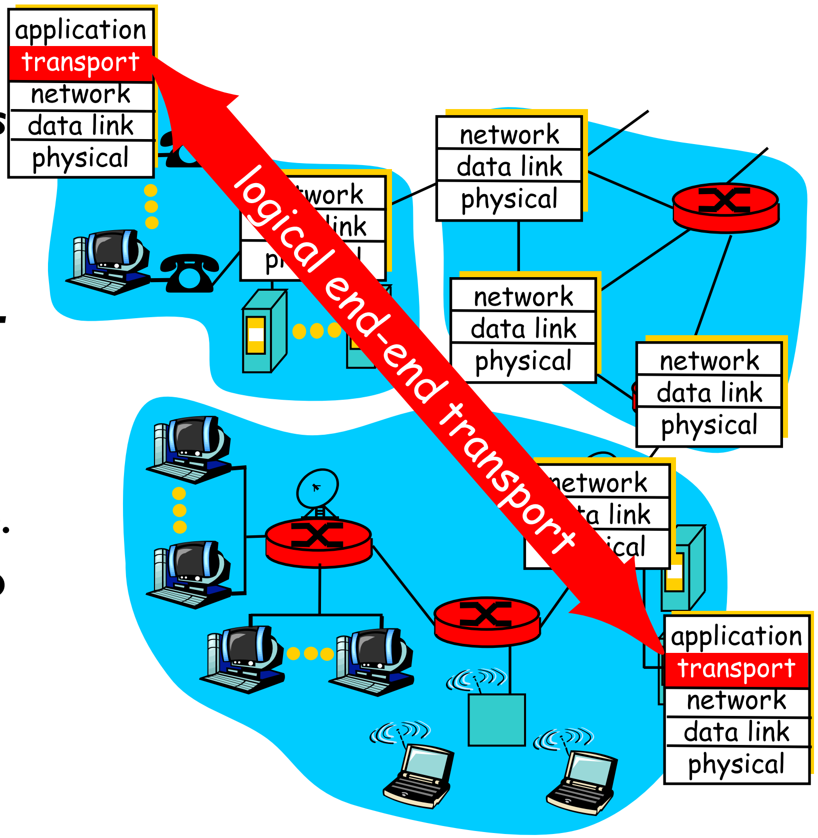
- *Transport layer:*
  - *logical communication* between *processes*
  - relies on network layer services
    - transport layer lies above network layer

- *Network layer:*
  - *logical communication* between *hosts*

Internet Transport Layer Protocols

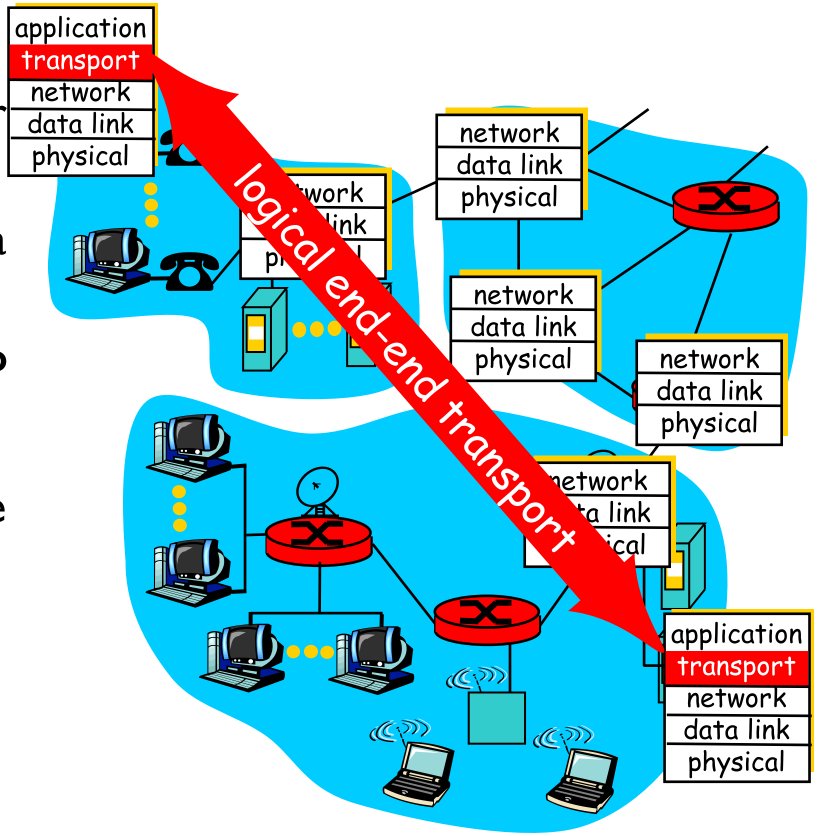
- Two-distinct transport-layer protocols:
  - ***UDP*** (User Datagram Protocol)
    - provide ***unreliable***, ***connectionless*** service to the invoking app.
  - ***TCP*** (Transmission Control Protocol)
    - provide ***reliable***, ***connection-oriented*** service to the invoking app.

- When designing net. app., the app. developer must specify one of these two transport protocols



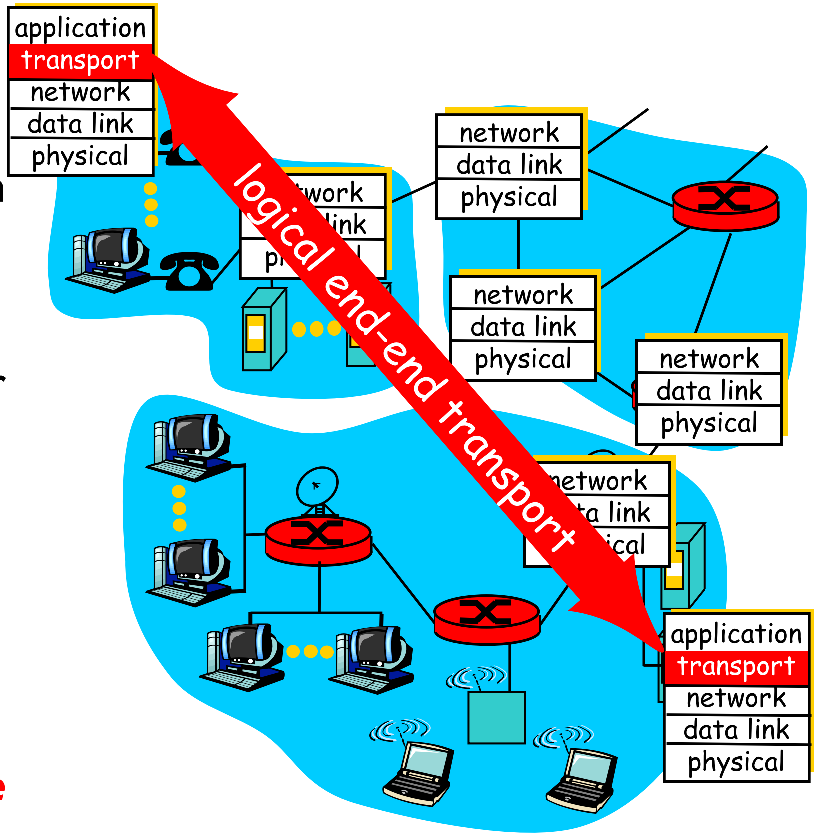
Internet Transport Layer Protocols

- In an Internet context, the transport-layer packet is called ***segment***
  - refers to the transport-layer packet for TCP as a ***segment***
  - refers to the packet for UDP as a ***datagram***
- It is less confusing to refer to both TCP and UDP packets as ***segment***
  - reserve the term ***datagram*** for the network-layer packet



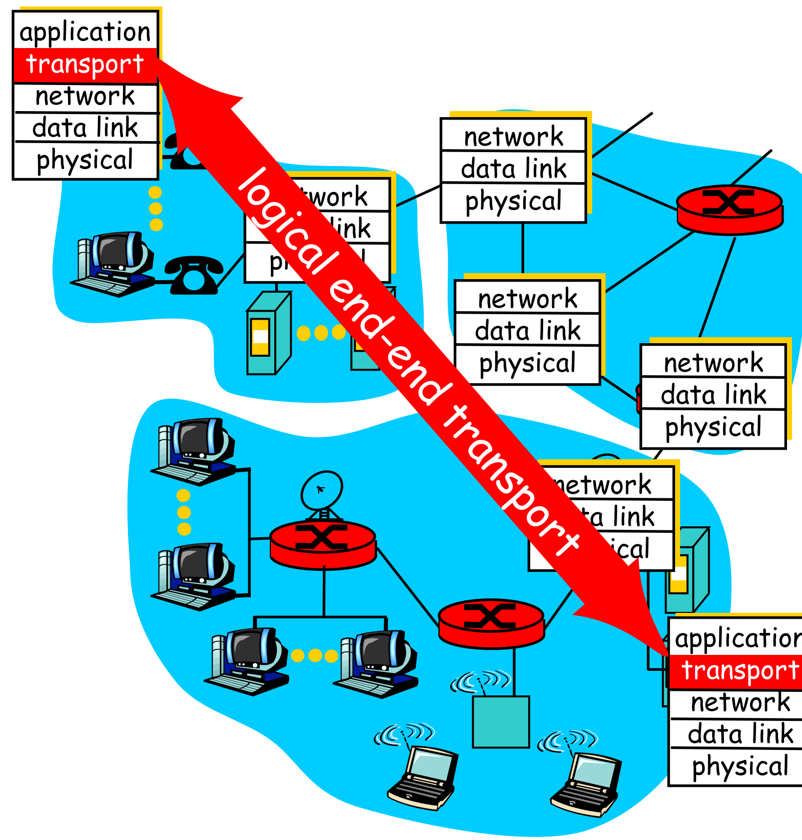
Internet Transport Layer Protocols

- A few words about netw. layer
  - netw. layer protocol
    - Internet Protocol (IP)
  - IP provides logical commu. between hosts
  - IP service: best-effort delivery service
    - making its "best effort" to deliver segments
    - ***making no guarantees on***
      - *segment delivery*
      - *orderly delivery*
      - *integrity of data*
  - IP service is said to be ***unreliable service***



Internet Transport Layer Protocols

- The most fundamental responsibility of UDP and TCP:
  - extend IP's delivery service between two end systems to a delivery service between two processes running on the end systems
  - extending host-to-host delivery to process-to-process delivery is called ***transport-layer multiplexing*** and ***demultiplexing***



Internet Transport Layer Protocols

- reliable, connection-oriented: **TCP**
  - connection setup
  - flow control
  - sequence number
  - acknowledgement
  - timer

  data delivered from sending side to receiving side correctly and in order

  - congestion control
    - a service for the Internet
    - prevents any one TCP connection from swamping the links and routers between comm. hosts
    - strives to give each connection traversing a congested link an equal share of the link bandwidth
  - integrity checking

Internet Transport Layer Protocols

- unreliable, connectionless: **UDP**
  - process-to-process delivery
  - integrity checking
    - including error detection fields in segments' header
  - unregulated traffic
    - app. can send at any rate it pleases, for as long as it pleases

Multiplexing & Demultiplexing

extending the host-to-host delivery service provided by the network layer to a process-to-process delivery service for applications running on the hosts

- at the destination host,
  - the transport layer receives segments from the network layer
  - transport layer
    - delivers the data in segments to the ***appropriate application process*** running in the host

How?

Multiplexing & Demultiplexing

- ***socket***
  - door through which data passes from the network to the process and through which data passes from the process to the network
  - the transport layer in the receiving host does not actually deliver data directly to a process, but instead to an intermediary *socket*
  - because at any given time there can be more than one socket in the receiving host, *each socket has a unique identifier*
  - *the format of the identifier* depends on whether the socket is a UDP or a TCP socket

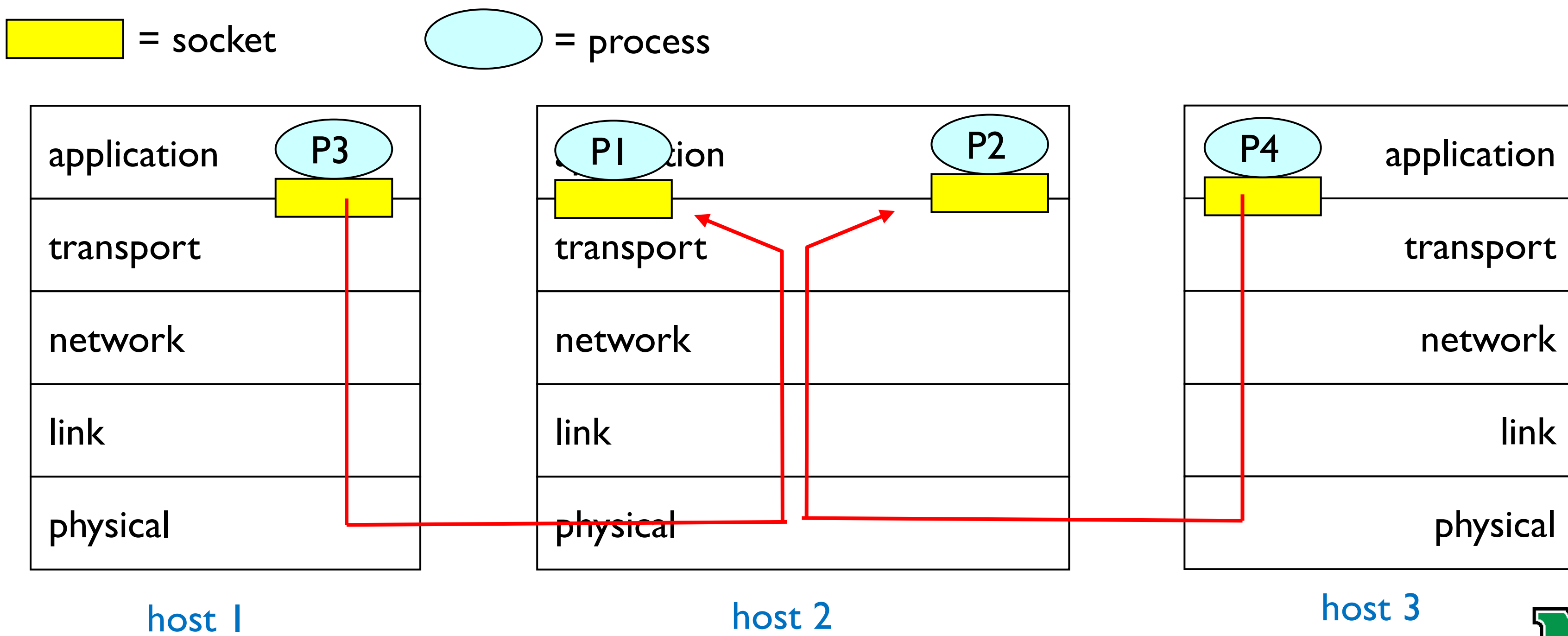
Multiplexing & Demultiplexing

Multiplexing at sending host:

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

Demultiplexing at receiving host:

Use header info to deliver received segments to correct socket



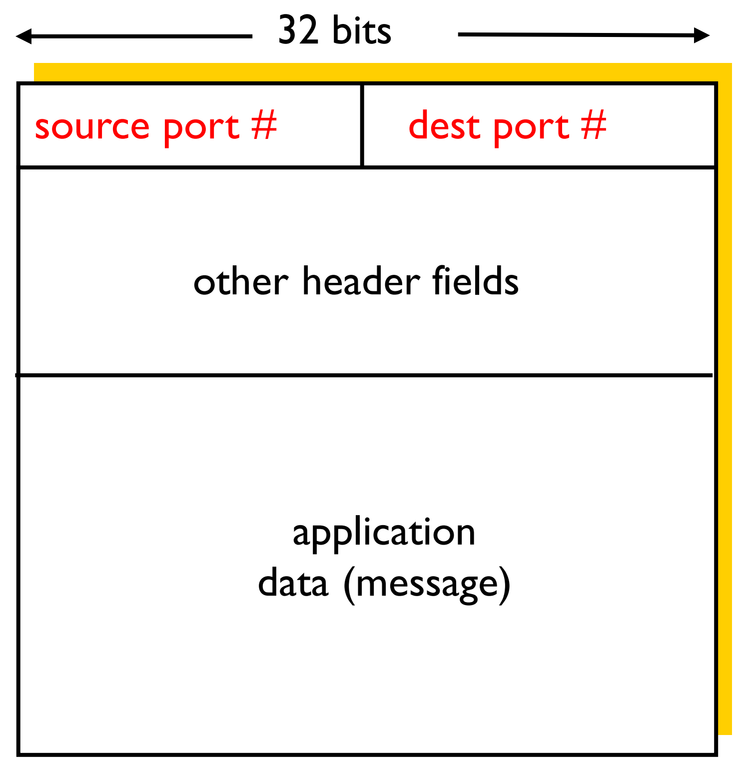
Multiplexing & Demultiplexing

Q: How a receiving host directs an incoming transport layer segment to the appropriate socket?

- ***transport-layer multiplexing*** requires
  - (i) sockets have unique identifiers
  - (ii) each segment has special fields that indicate the socket to which the segment is to be delivered

How Demultiplexing Works

- Special fields
  - **source port # field**
  - **destination port # field**
- Each port # is a 16-bit number
  - ranging from 0 to 65535
  - 0 to 1023
    - well-known port #
    - restricted for use
- How to implement demultiplexing?
  - host uses IP addresses & port # to direct segment to appropriate socket

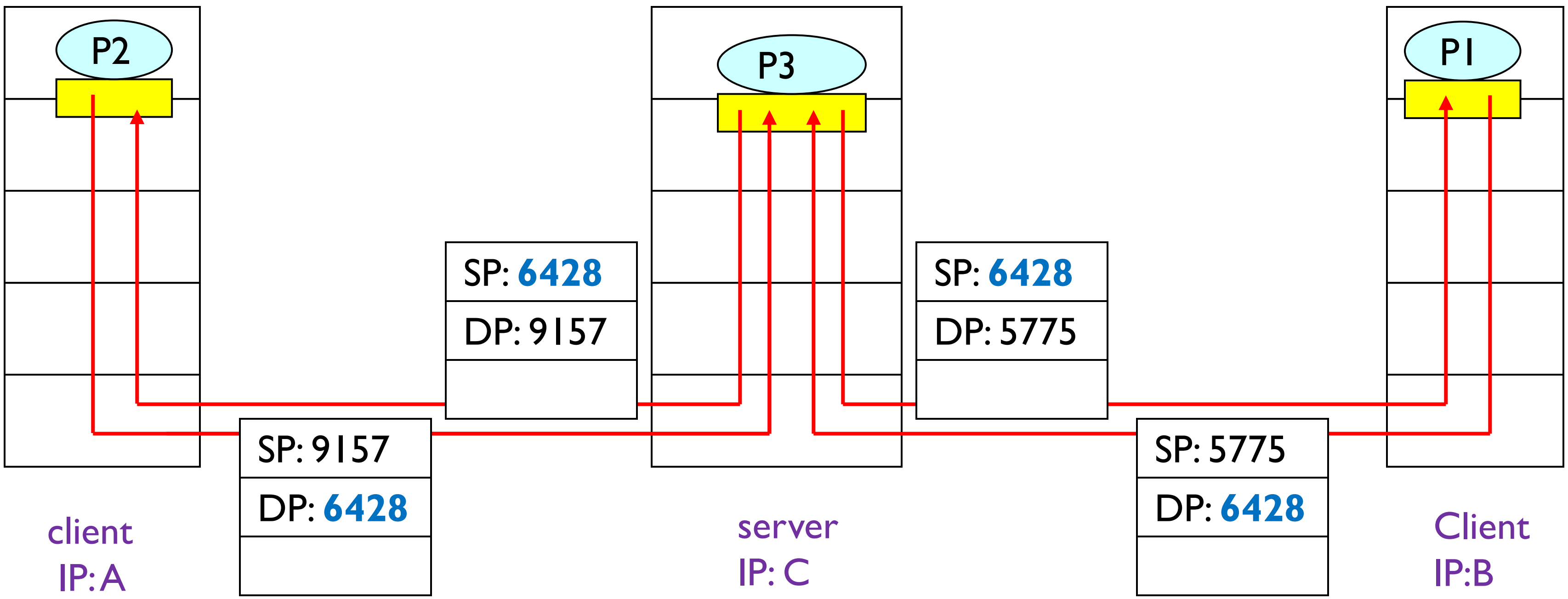


TCP/UDP segment format

Connectionless (UDP) Demultiplexing

- Host A wants to send data to Host B
  - Host A
    - create segment including data, source port #, destination port #
    - UDP socket identified by **two-tuple**
      - *destination IP address*
      - *destination port #*
    - pass the resulting segment to the network layer
  - Host B
    - (transport layer) checks *destination port #* in segment
    - directs UDP segment to socket with that port #
- IP datagrams with same destination port#, but different source IP addresses and/or source port #,
  - will be directed to same socket at destination
  - because UDP socket is fully identified by **two-tuple:** *destination* IP address and *destination* port #

Connectionless (UDP) Demultiplexing (cont.)



What is the purpose of SP?

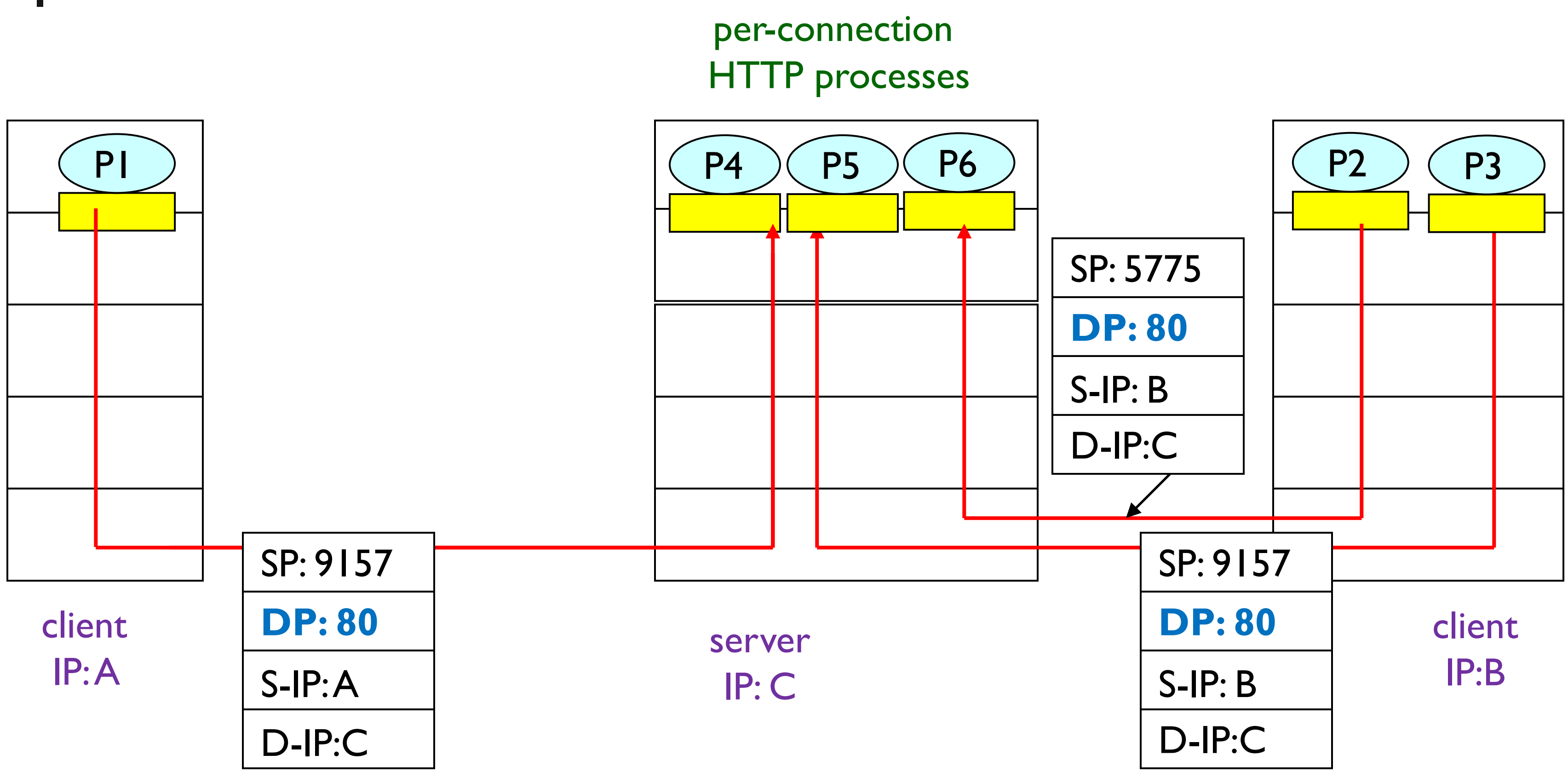
SP provides "return address"

Connection-oriented (TCP) Demux

- TCP **socket** identified by **4-tuple**:
  - *source IP address*
  - *source port #*
  - *destination IP address*
  - *destination port #*
- Demux: receiving host uses *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
- Web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

Connection-oriented Demux (cont.)



Three segments, all destined to IP address: C,
dest port: 80 are demultiplexed to different sockets.

UDP: User Datagram Protocol [RFC 768]

- UDP does about as little as a trans. protocol can do
  - multiplexing/demultiplexing function
  - some light error checking
  - nothing else
- app. chooses UDP?
  - UDP takes messages from app. process
  - attaches source and destination port #
  - adds two other small fields
  - passes the resulting segment to netw. layer
  - netw. layer encapsulates segment into IP datagram
  - makes a best-effort attempt to deliver to the receiving host
  - segment arrives at the receiving host
    - UDP uses destination port # to deliver to app. process

UDP: User Datagram Protocol [RFC 768]

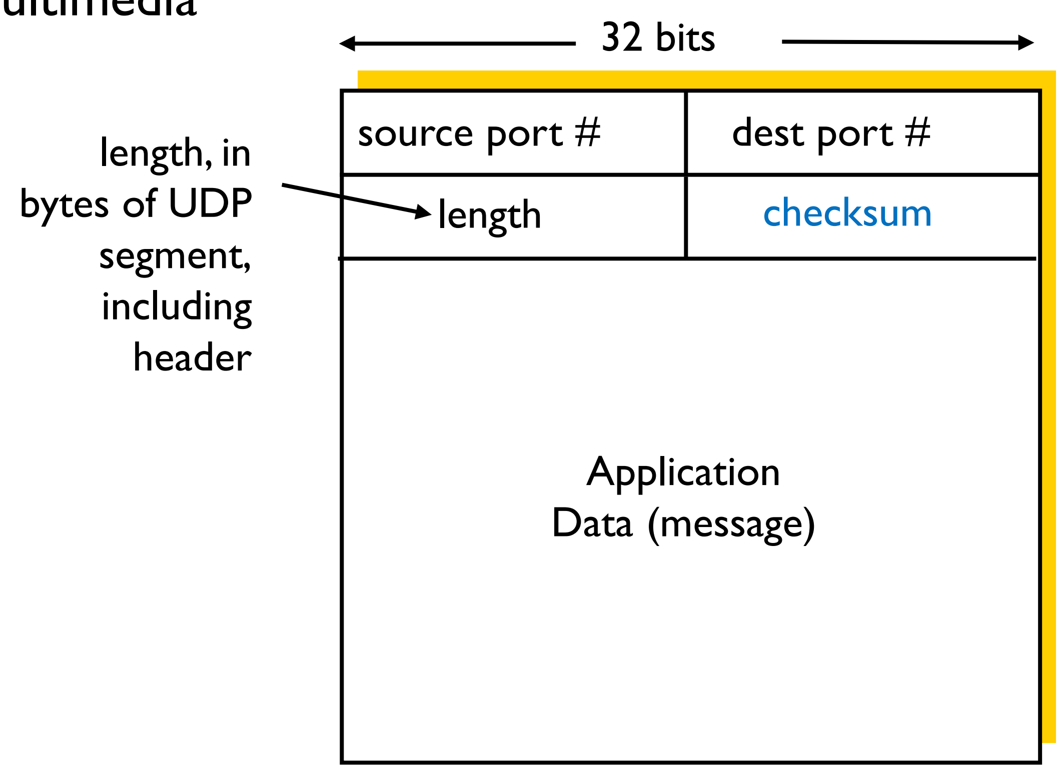
- *connectionless:*
  - no handshaking between UDP sender and receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender and receiver
- small segment header
  - TCP: 20 bytes of header overhead
  - UDP: 8 bytes
- no congestion control:
  - UDP can blast away as fast as desired

UDP: More

- often used for streaming multimedia apps
  - loss tolerant
- other UDP uses
  - DNS



UDP segment format

UDP Checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:

- performs the 1s complement of the sum of all the 16-bit words in the segment
  - with any overflow encountered during the sum being wrapped around
- puts the result in the checksum field of UDP segment

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected

Internet Checksum Example

- **NOTE**:
  - when adding numbers, a carryout from the most significant bit needs to be added to the result
- example: add two 16-bit integers

	1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
	1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
wraparound	1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
sum	1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum	0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1

Internet Checksum Example

- **NOTE**:
  - the addition had overflow, which was wrapped around

- The 1s complement is obtained by converting all the 0s to 1s and converting all the 1s to 0s