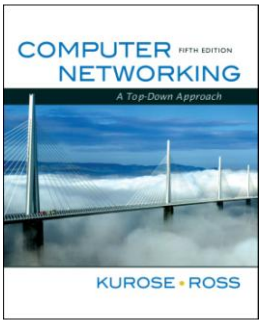
RSC
Part III: Transport Layer
1. Basic Concepts



Redes y Servicios de Comunicaciones
Universidad Carlos III de Madrid

These slides are, mainly, part of the companion slides to the book “Computer Networking: A Top Down Approach” generously made available by their authors (see copyright below). The slides have been adapted, where required, to the teaching needs of the subject above.

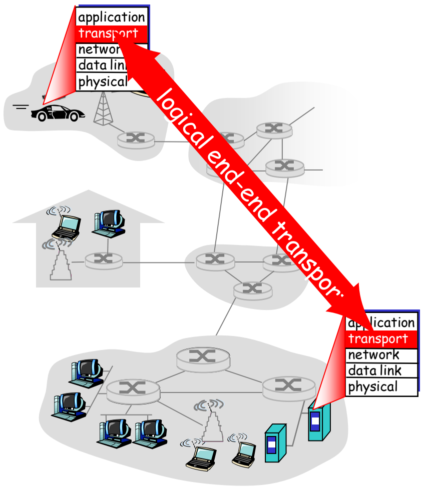
Computer Networking: A Top Down Approach
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.

RSC Part III: Transport Layer

- ❒ III. 1 Basic Transport layer concepts
  - ❍ Transport layer Principles
  - ❍ Transport layer Services
  - ❍ Multiplexing and Demultiplexing
- ❒ III.2 UDP
  - ❍ UDP Segment format
  - ❍ UDP cheksum
- ❒ III.3 TCP
  - ❍ TCP connection
  - ❍ TCP Segment, sequence and ack numbers
  - ❍ RTT Estimation and Timeout
  - ❍ Reliable Data Transfer
  - ❍ Flow Control
  - ❍ TCP connection Management
  - ❍ TCP Congestion Control

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Transport vs. network layer

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

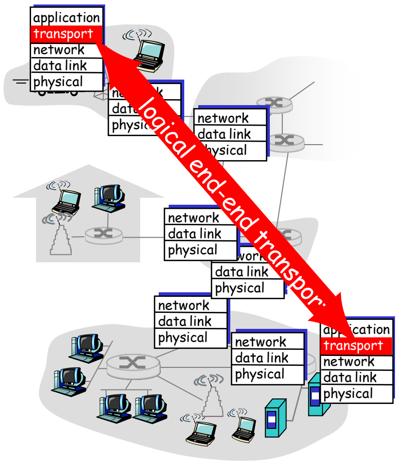
Household analogy:

12 kids sending letters to 12 kids

- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



RSC Part III: Transport Layer

- III. 1 Basic Transport layer concepts
  - Transport layer Principles
  - Transport layer Services
  - Multiplexing and Demultiplexing
- III.2 UDP
  - UDP Segment format
  - UDP cheksum
- III.3 TCP
  - TCP connection
  - TCP Segment, sequence and ack numbers
  - RTT Estimation and Timeout
  - Reliable Data Transfer
  - Flow Control
  - TCP connection Management
  - TCP Congestion Control

Multiplexing/demultiplexing

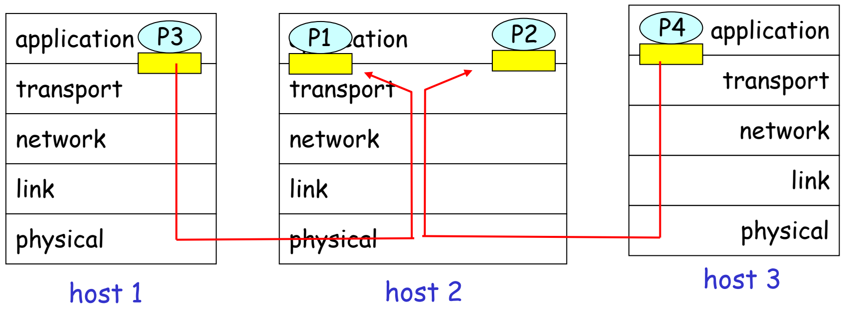
Demultiplexing at rcv host:

delivering received segments to correct socket

Multiplexing at send host:

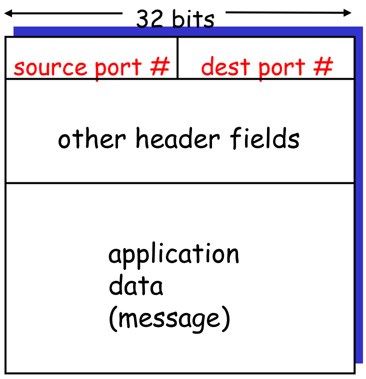
gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

🟨 = socket ◯ = process



How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 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 demultiplexing

- Create sockets with port numbers:

```
DatagramSocket mySocket1 = new
   DatagramSocket(12534);
DatagramSocket mySocket2 = new
   DatagramSocket(12535);
```

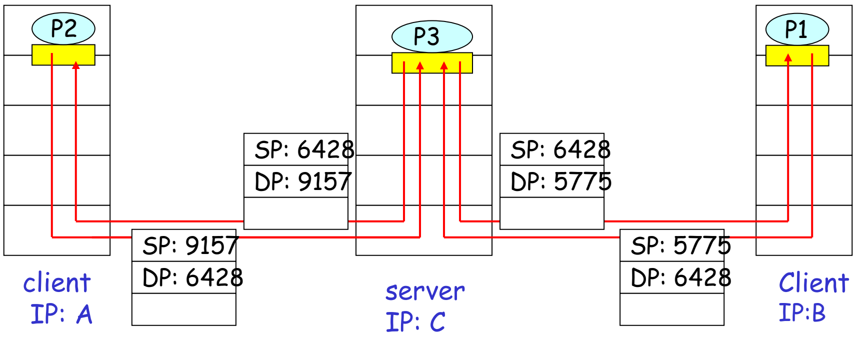
- UDP socket identified by two-tuple:

(dest IP address, dest port number)

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers directed to same socket

Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

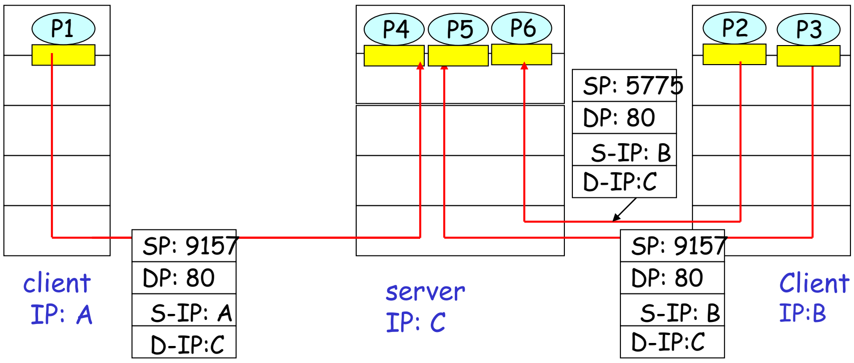


SP provides "return address"

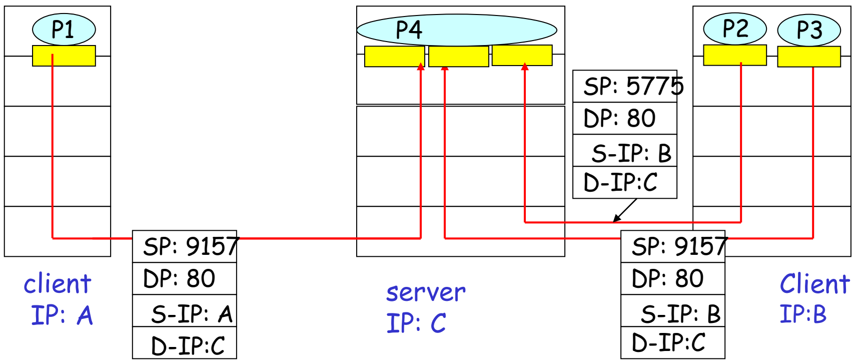
Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv 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)



Connection-oriented demux: Threaded Web Server



RSC Part III: Transport Layer

- III. 1 Basic Transport layer concepts
  - Transport layer Principles
  - Transport layer Services
  - Multiplexing and Demultiplexing
- III.2 UDP
  - UDP Segment format
  - UDP cheksum
- III.3 TCP
  - TCP connection
  - TCP Segment, sequence and ack numbers
  - RTT Estimation and Timeout
  - Reliable Data Transfer
  - Flow Control
  - TCP connection Management
  - TCP Congestion Control

UDP: User Datagram Protocol [RFC 768]

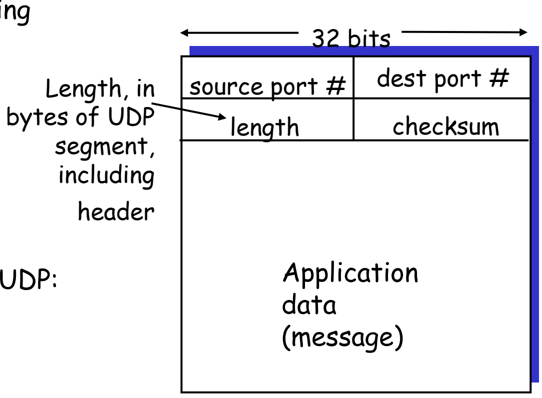
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



32 bits

source port #	dest port #
length	checksum

Application data (message)

Length, in bytes of UDP segment, including header

UDP segment format

UDP checksum

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

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later ....

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
```