Digital Communication in the Modern World

Transport Layer Multiplexing, UDP

http://www.cs.huji.ac.il/~com1
com1@cs.huji.ac.il

Some of the slides have been borrowed from:
Computer Networking: A Top Down Approach Featuring the Internet, 2nd edition.
Jim Kurose, Keith Ross
Addison-Wesley, July 2002.

IPC: Files, Named pipes and Sockets

- Files
  - slow
  - unsecure
- Named pipes
  - not suitable for network
- Sockets
  - suitable for networking

Transport vs. network layer

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

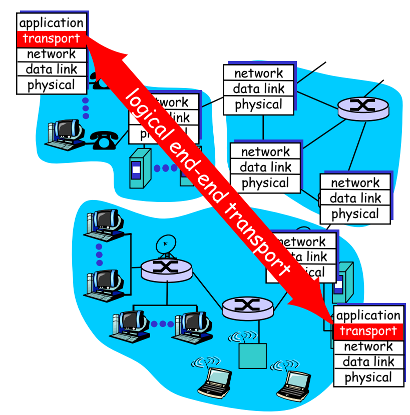
Household analogy:

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

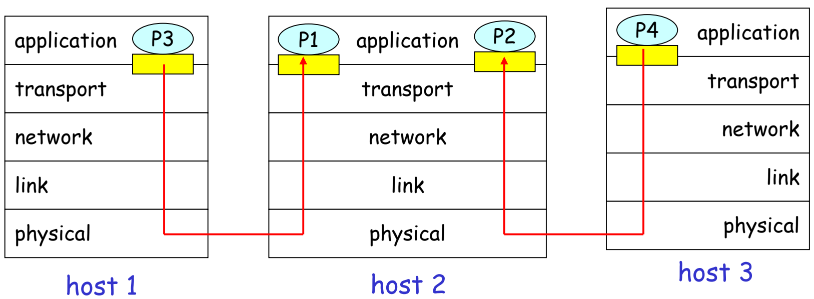


Multiplexing/demultiplexing

Demultiplexing at rcv host:
delivering received segments to correct socket

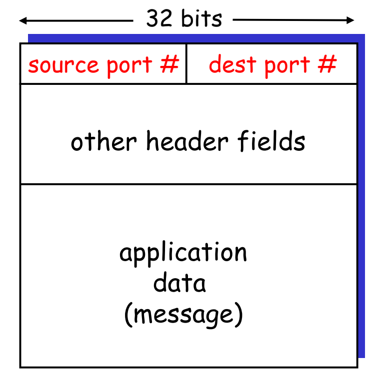
Multiplexing at send host:
gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

= 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 (recall: well-known port numbers for specific applications)
- 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(99111);
DatagramSocket mySocket2 = new
   DatagramSocket(99222);
```

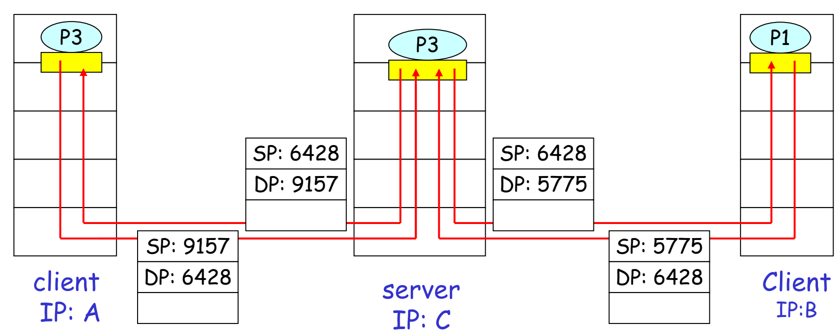
- UDP socket identified by two-tuple:

(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);
```

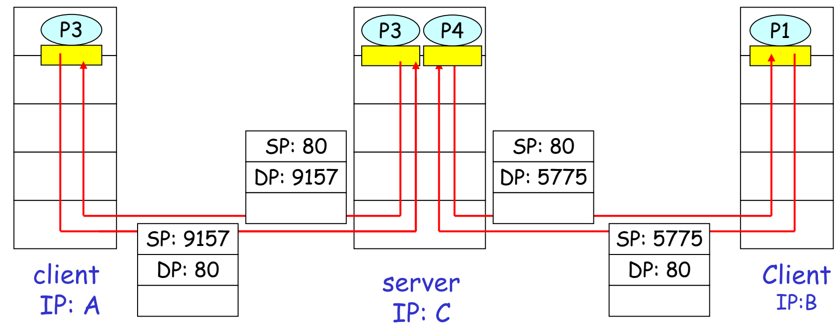


SP provides "return address"

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)



UDP: User Datagram Protocol [RFC 768]

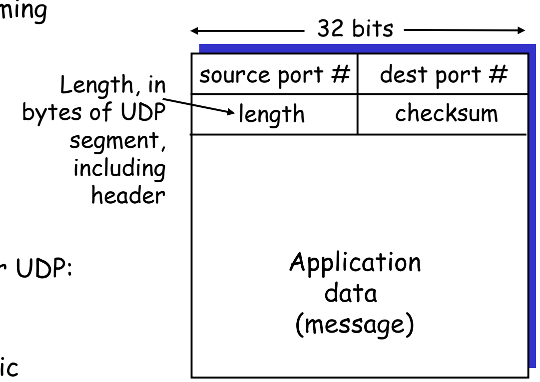
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

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!



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?*

UDP checksum - example

Suppose we have three 16-bit words
0110011001100110
0101010101010101
0000111100001111

The sum of the first of these three words
0110011001100110
0101010101010101
1011101110111011

UDP checksum - example

Adding the third word to the above sum give
1011101110111011
0000111100001111
1100101011001010

The 1's complement is obtained by converting all the 0s to 1s and all the 1s to 0s. Thus the 1's complement of the sum 110010101100101 is 0011010100110101, which becomes the checksum.
At the receiver all three words and the checksum are added. If no errors were added then the sum will be: 1111111111111111.