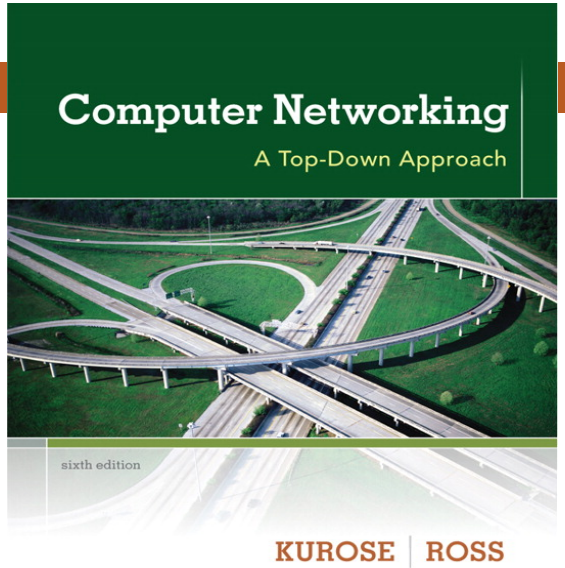
Transport Layer

Services, multiplexing and demultiplexing

Connectionless Transport: UDP.

Chapter 3
Transport Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

Chapter 3: Transport Layer

our goals:

- 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

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

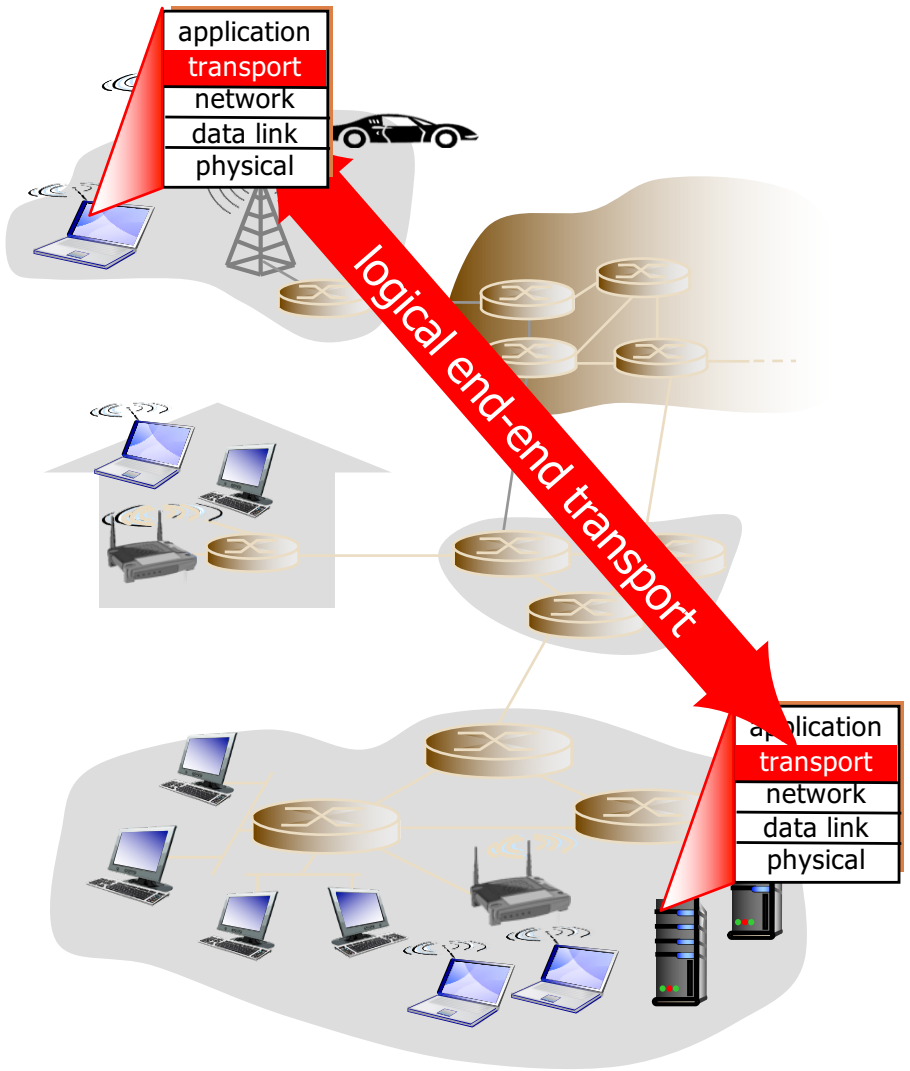
- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
  - (appearing as directly connected)
- 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 (*basic function*)
  - relies on, enhances, network layer services
  - *Complementary* functions: reliable transfer, flow control, congestion control

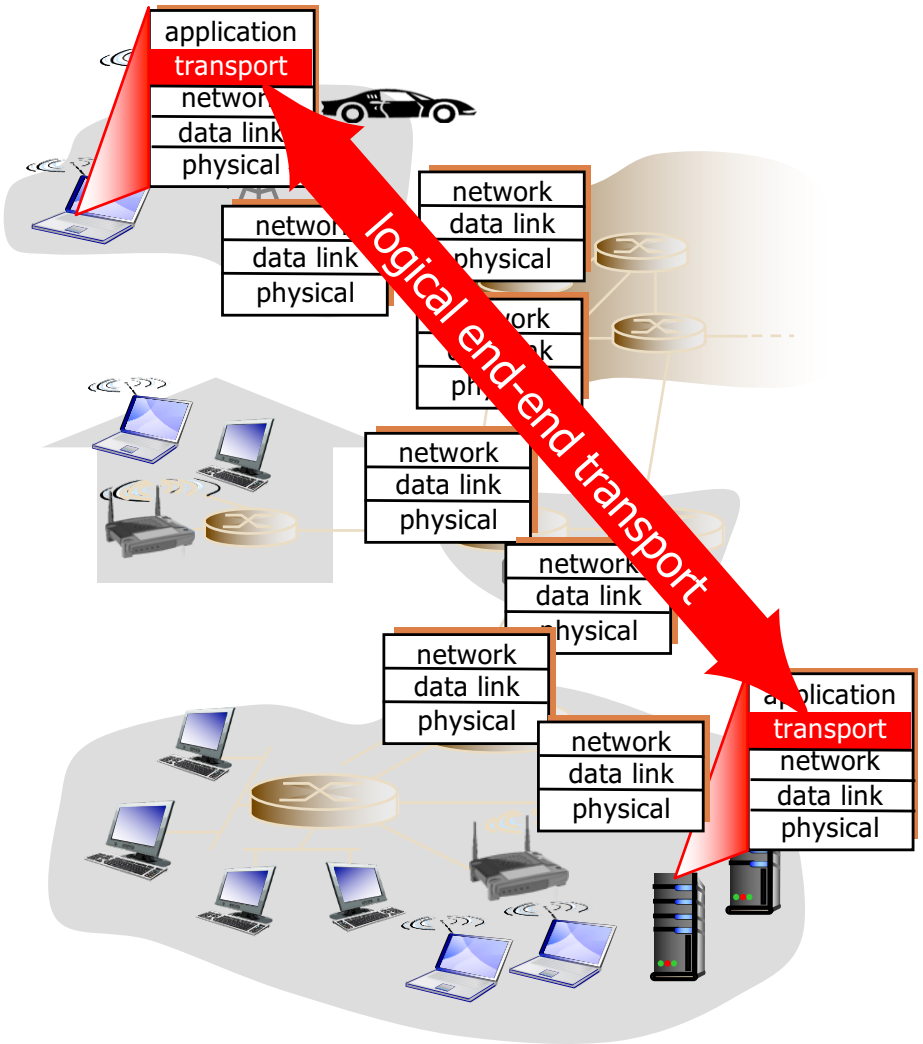
household analogy:

12 kids in Ann's house sending letters to 12 kids in Bill's house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- 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



Transport Layer: Introduction

- Relationship between Network and Application Layer.
  - Through logical operations of multiplexing and demultiplexing.
    - Multiple transport processes use network layer services.
  - Transport Protocols
    - Only run at end systems, not at intermediate systems (routers, switches).
    - Use basic services of network layer (IP).
    - Independent of how IP datagramms are transferred through the network
    - IP datagrams = IP Packets = Network Layer PDUs at Internet.
    - Transport PDUs = SEGMENTS
      - Transport layer **segments** data received from application before transmitting them
      - Receiver reassembles the received segments

*: See 3.1.2 for PDU nomenclature at different layers.

Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

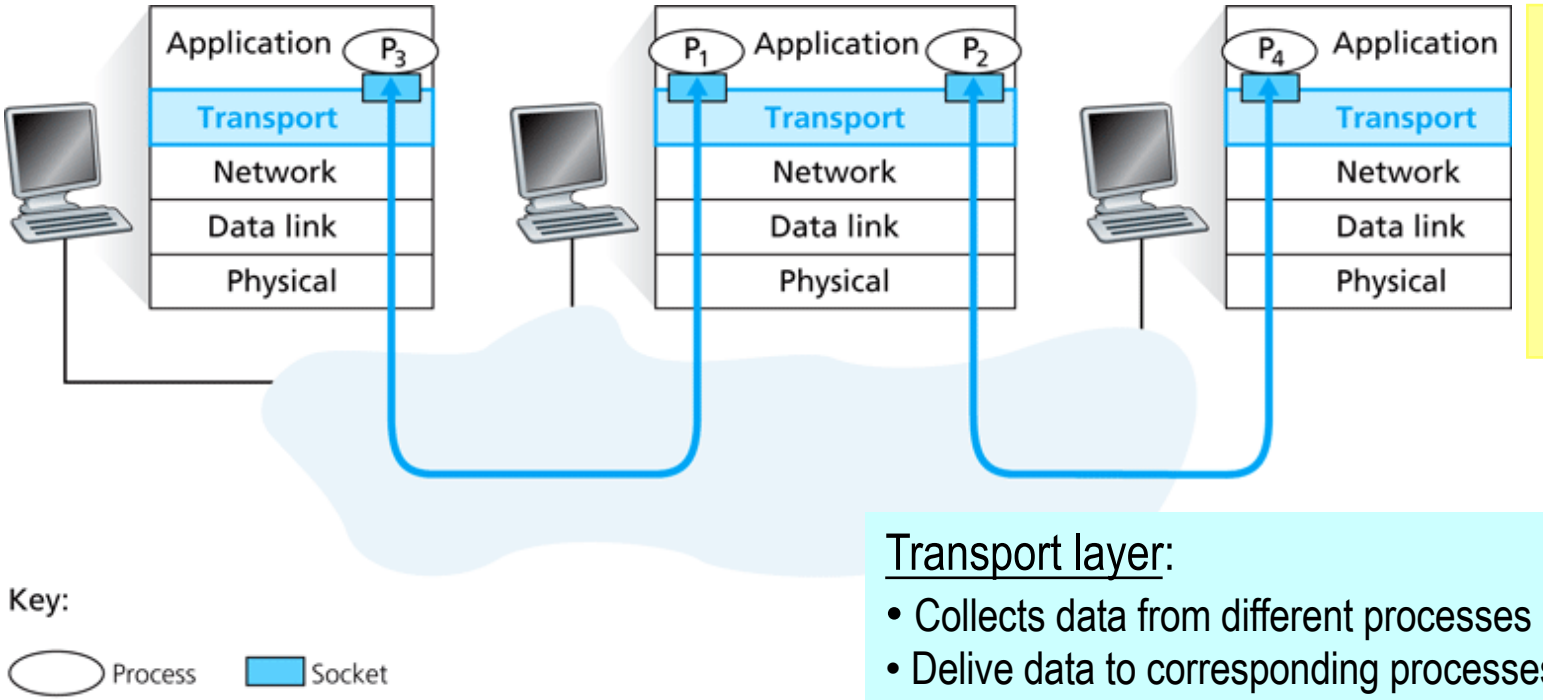
3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

Transport Layer: mux/demux.



Socket: abstraction to identify **every end of a logic communication between processes at application layer**, located at different hosts.

"Gate" to transfer data between application and transport layer

Transport layer:

- Collects data from different processes at source system ➔ Mx.
- Delive data to corresponding processes at destination ➔ DMx.

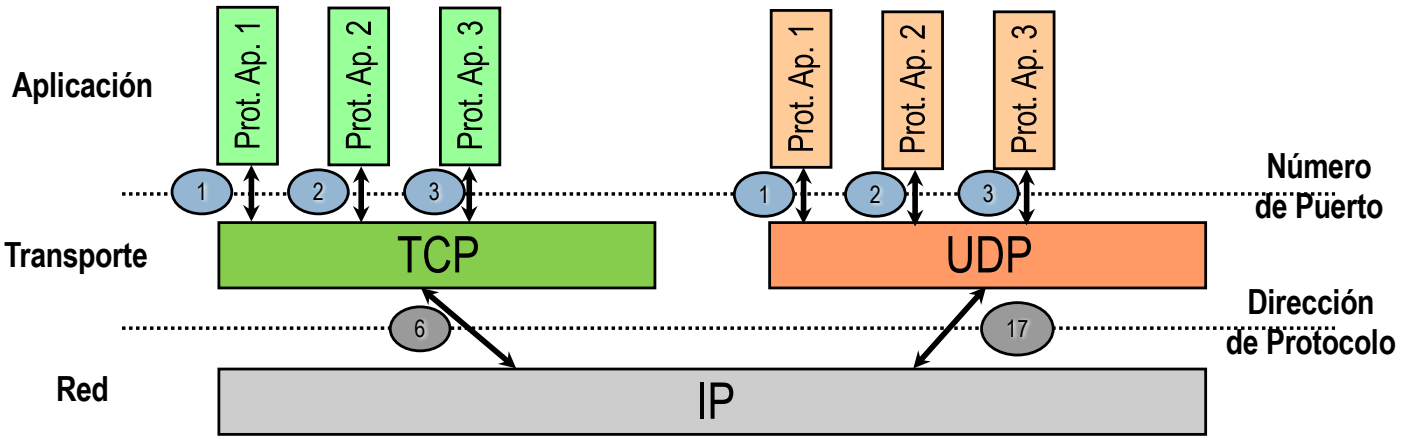
Figure 3.2 ♦ Transport-layer multiplexing and demultiplexing

Different processes of a system may need to use the same services of transport layer (same transport protocol).

- P.e: DNS and TFTP use UDP; Web, FTP, email use TCP.
- Nbr of port is used to distinguish them.

Transport Layer.

- Addressing levels at hosts
  - Three addressing levels are needed at hosts to:
    - Identify the host (host address).
      - Located at header of IP datagram.
    - Identify the transport protocol used.
      - At header of IP datagram.
    - Identify the process at host
      - Located at UDP/TCP segment header (port number).



Transport Layer: Multiplexing and demultiplexing

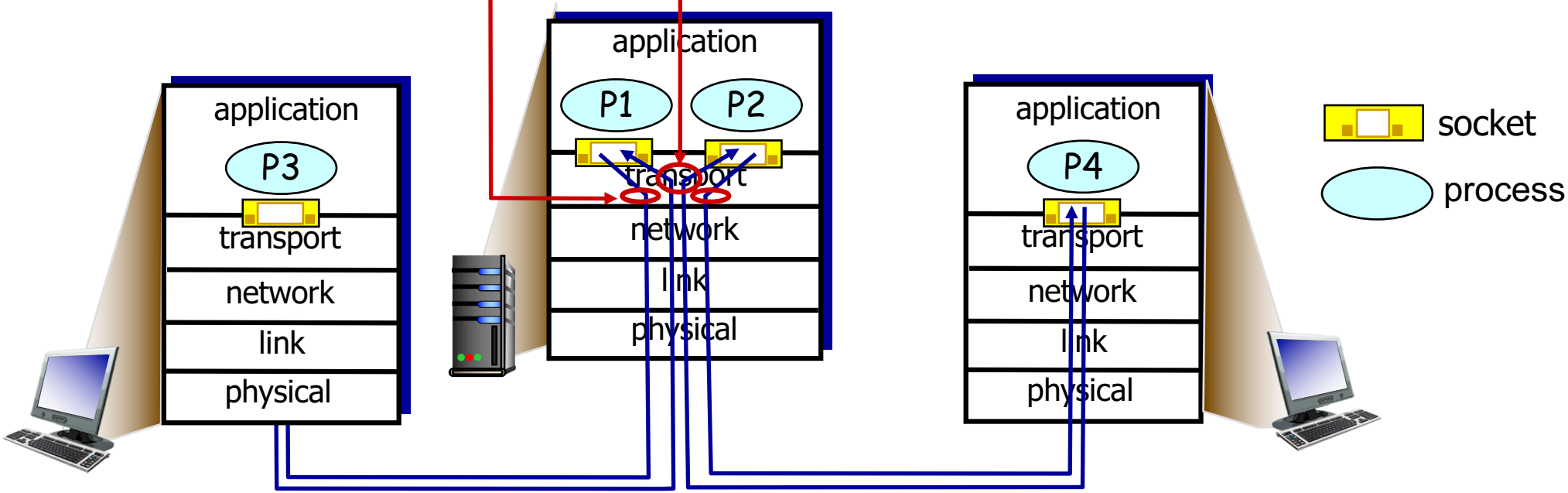
- Multiplexing and demultiplexing*.
  - Service provided by layer “N” protocol to several “N+1” processes.
  - Logic gate for Mx and DMx: SAP, *Services Access Point* (OSI term).
    - Named as “port” in transport layer.
  - Multiplexing (Mx).
    - When several processes (protocols) of layer “N+1” carry data or use a layer “N” service.
  - Demultiplexing (DMx).
    - When a layer “N” carries data to several processes (protocols) of layer“N+1”.

*: **Logical** multiplexing/demultiplexing, different to physical multiplexing (physical layer).

Multiplexing/demultiplexing

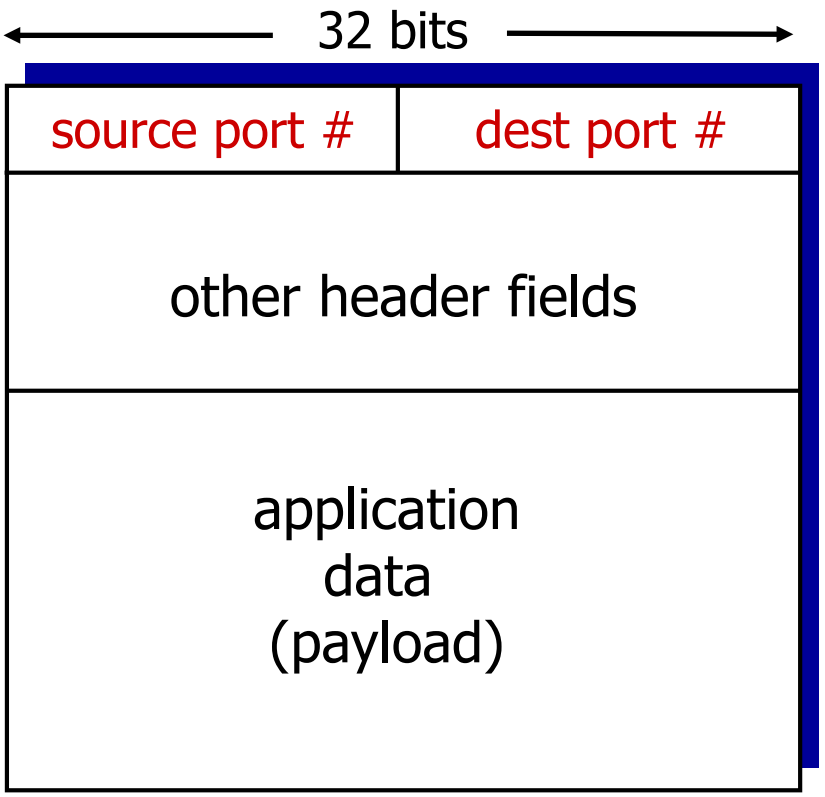
multiplexing at sender:
handle data from multiple sockets, add transport header (later used for demultiplexing)

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
- *If port or IP address is different: different socket*



TCP/UDP segment format

Transport Layer: mux /demux.

- Socket identification. Unique identifier:
  - *UDP Socket* , by two elements.
    - IP address, port number.
    - Identified only by destination.
  - *TCP Socket*, identified by four elements
    - Source IP address, source port number.
    - Destination IP address, destination port number.
    - Identified by source and destination.

Connectionless demultiplexing

- *recall:* created socket has host-local port #:

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

- *recall:* when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

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

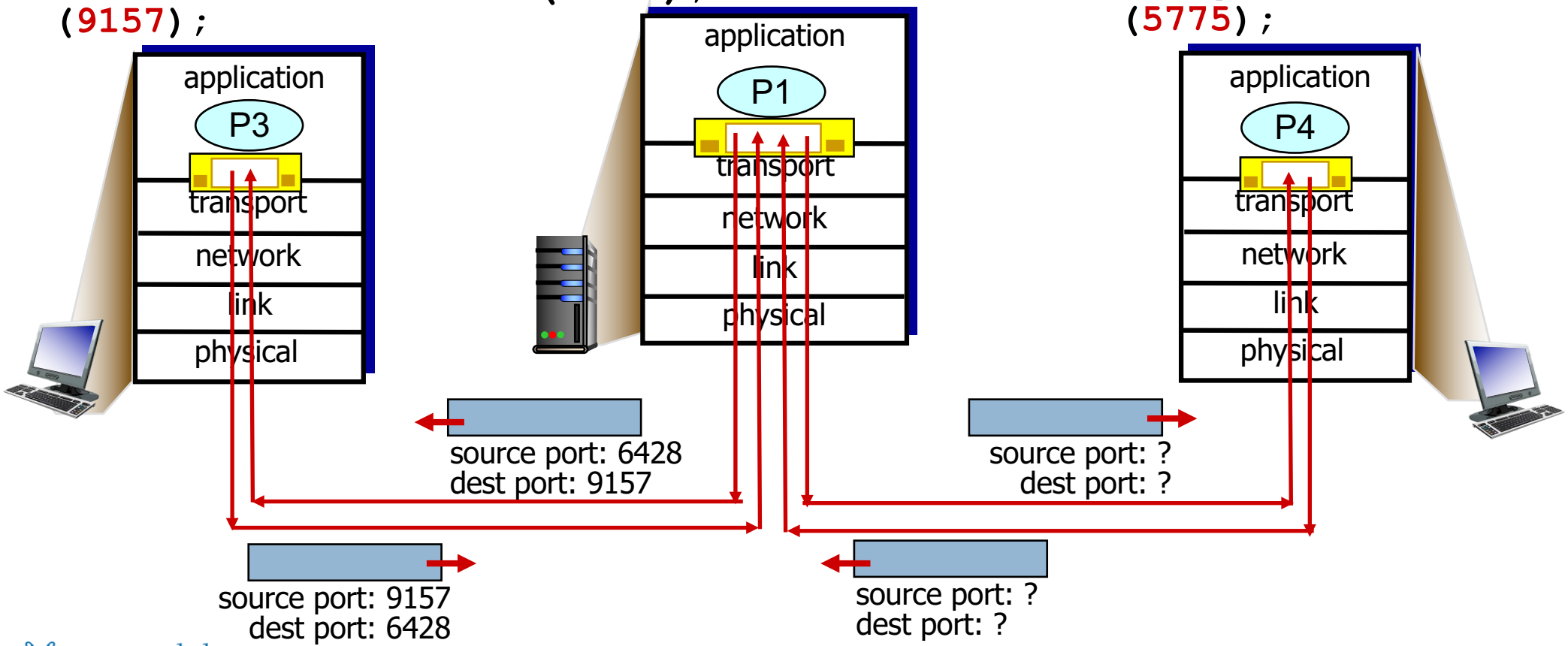
IP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

Connectionless demux: example


```
DatagramSocket
 mySocket2 = new
 DatagramSocket
 (9157);
```

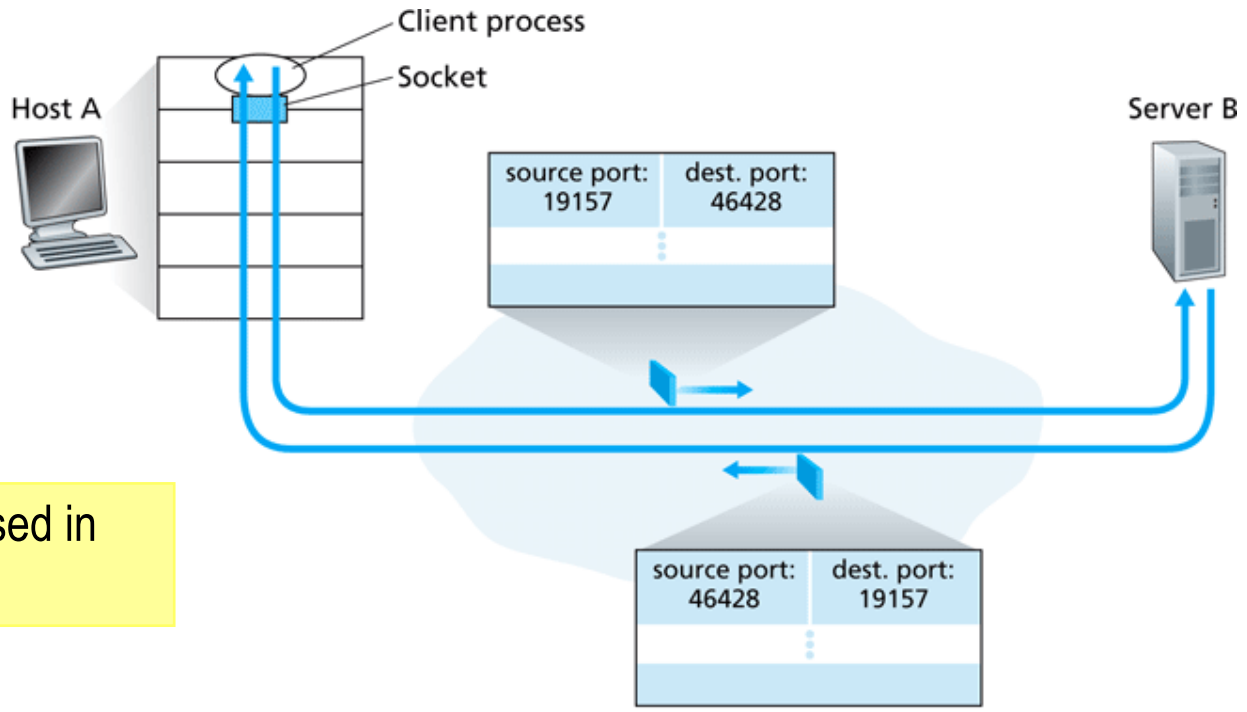
```
DatagramSocket
  serverSocket = new
  DatagramSocket
  (6428);
```

```
DatagramSocket
 mySocket1 = new
 DatagramSocket
 (5775);
```



Transport Layer: mux /demux.

- Non-connective mode demux (UDP) (II).
  - What are source IP address and port number used for ?
    - As return address, if response is needed.



Ports are reversed in response

Figure 3.4 ♦ The inversion of source and destination port numbers

Transport Layer: mux /demux.

- Demultiplexing in connective mode (TCP) (II).
  - A server may handle multiple *sockets* open simultaneously.
    - Each socket identified by its four element tuple.
    - A "welcome" *socket*.
    - As many "connection" *sockets* as simultaneous TCP connections.

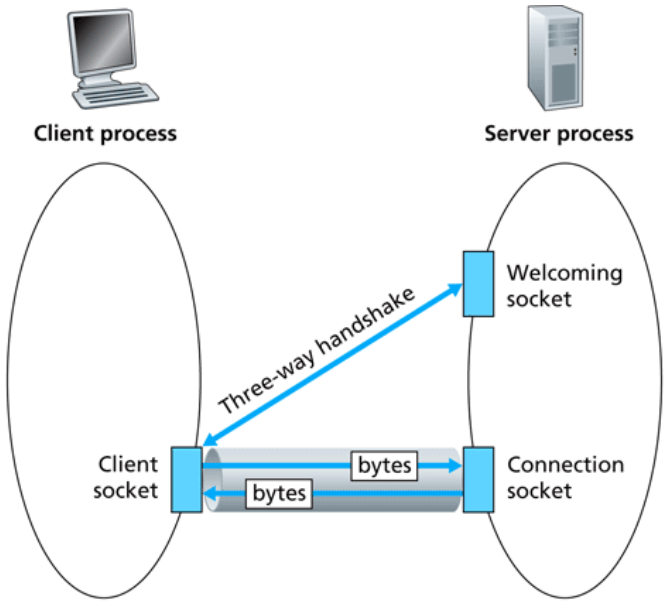


Figure 2.31 ♦ Client-socket, welcoming socket, and connection socket

Transport Layer: Multiplexing and demultiplexing

- Demux in connective mode (TCP) (III).
  - e.g, Web server.
    - All request to set up initial TCP connection and objects are sent to TCP port 80.
      - Server discriminates requests according to origin (IP address and source port) .
    - Sets different *sockets* per connected HTTP client.
      - HTTP with non persistent TCP ➔ Different *sockets* , for each request.
      - HTTP with persistent TCP ➔ Same *socket* for all requests.
    - Per TCP connection, an application process is created at server.
      - Every process accessed via its appropriate connection *socket*.
      - Not always one-to-one processes-sockets correspondence.
      - On high performance servers, one-to-many
        - Main process creates multiple threads (sub processes).
        - One *socket* per thread

Transport Layer: mux/demux.

- Demultiplexing in connective mode (TCP) (IV).

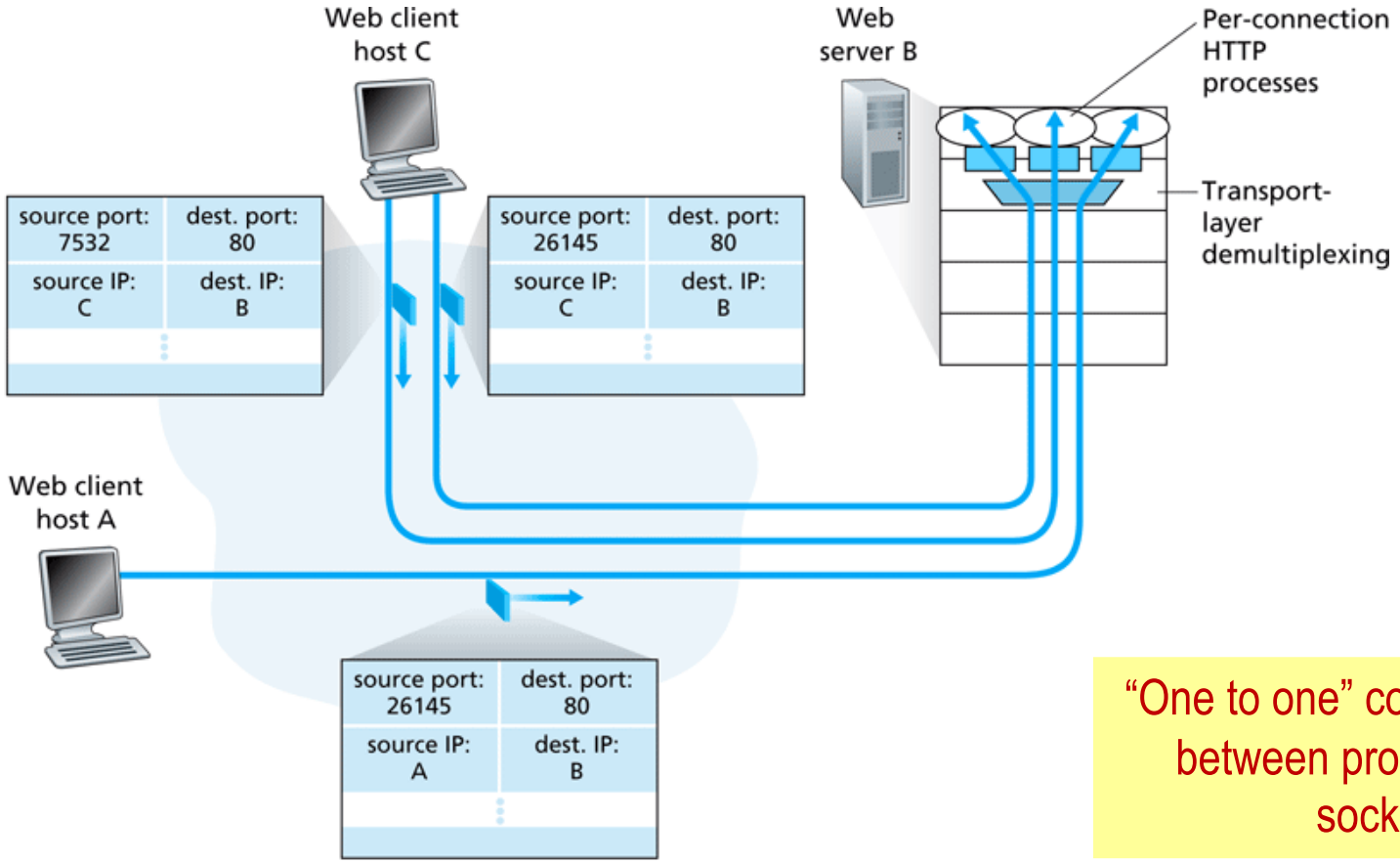
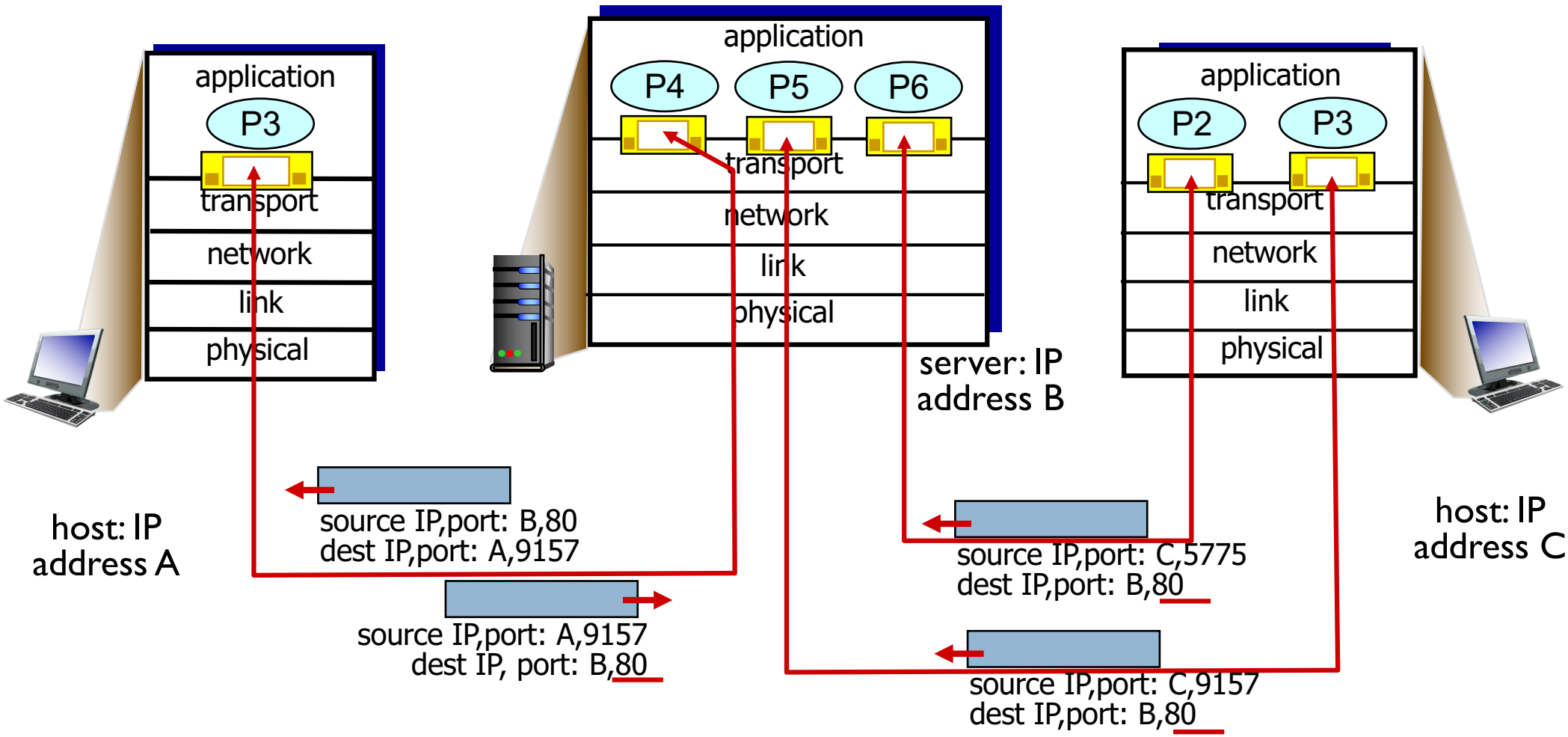


Figure 3.5 ♦ Two clients, using the same destination port number (80) to communicate with the same Web server application

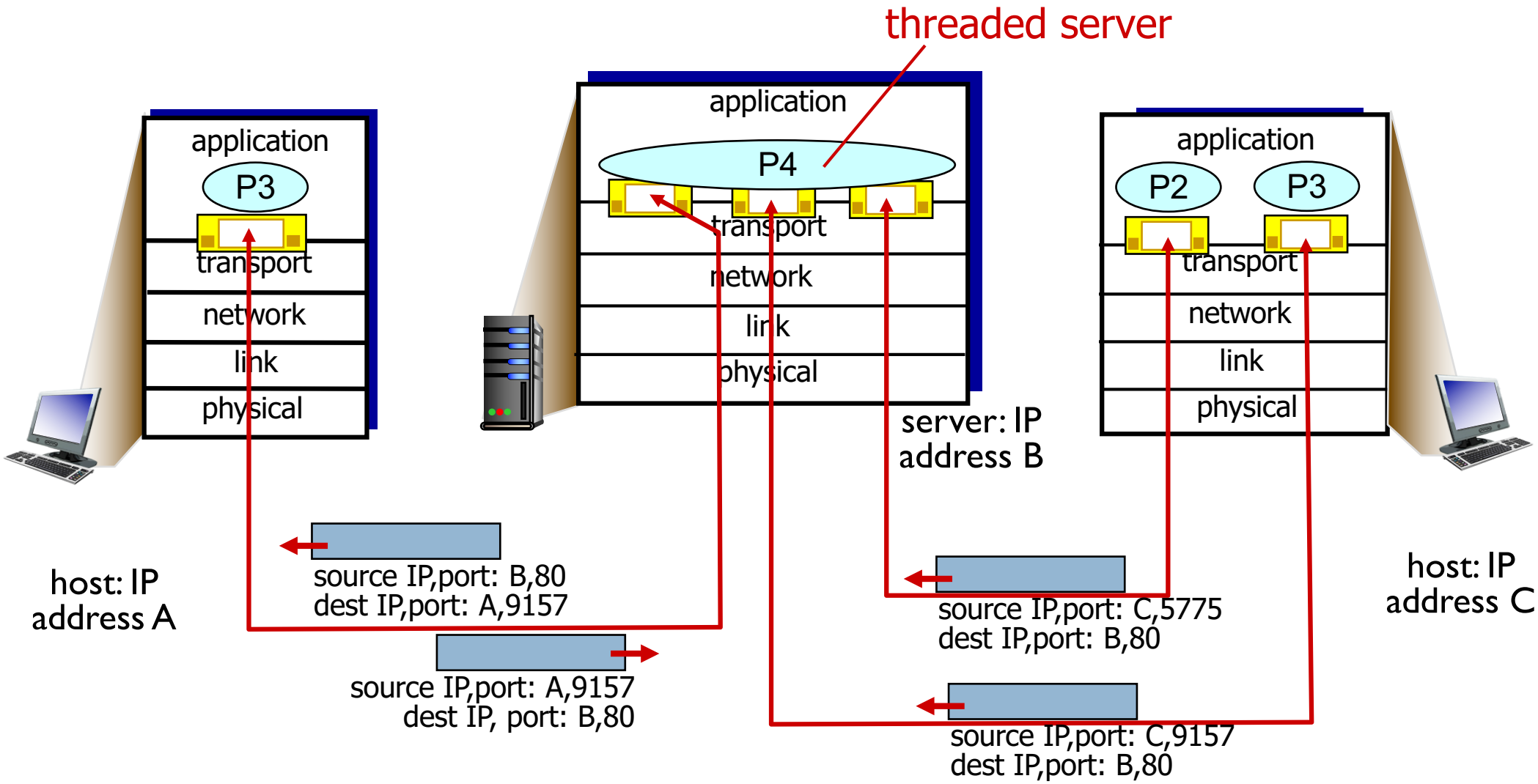
"One to one" correspondence between processes and sockets.

Connection-oriented demux: example



three segments, all destined to IP address: B, dest port: 80 are demultiplexed to *different* sockets

Connection-oriented demux: example



Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

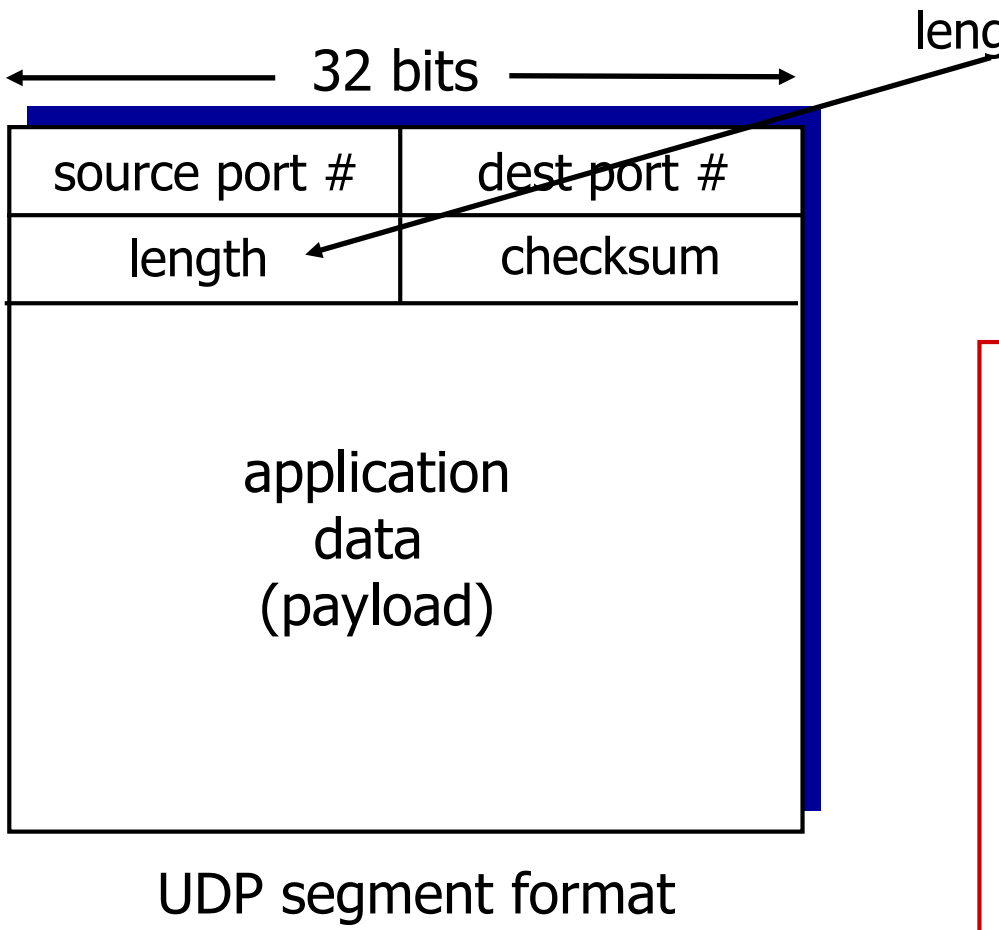
3.7 TCP congestion control

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

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

UDP: segment header



UDP segment format

why is there a UDP?

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

UDP checksum

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

sender:

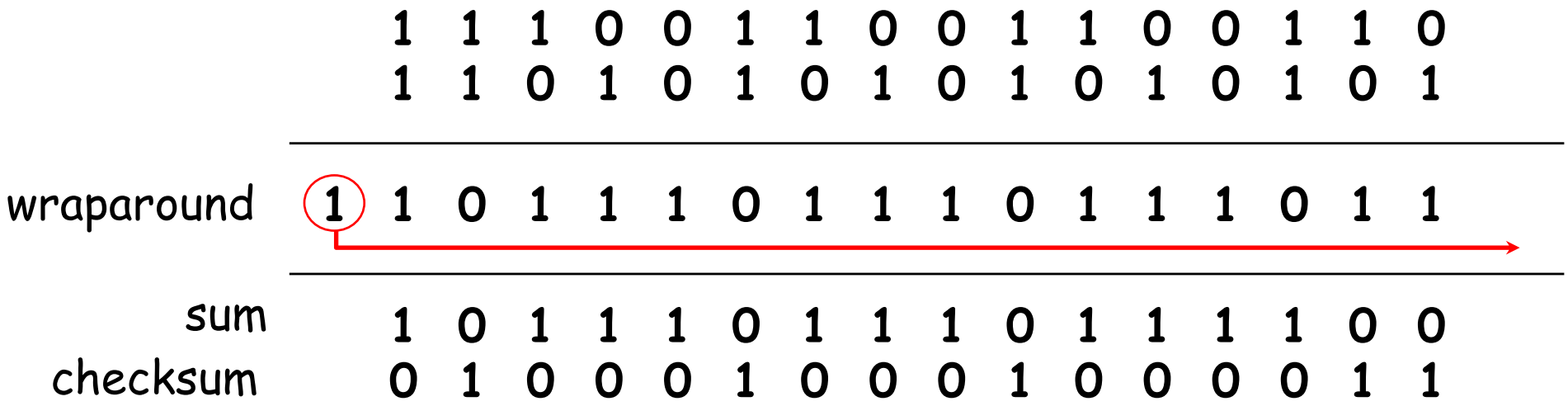
- treat segment contents, including header fields, as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

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

example: add two 16-bit integers



Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

La capa de Transporte: UDP.

- UDP, *User Datagram Protocol* (RFC 768).
  - Very light transport protocol, minimalist .
    - Mux and demux.
    - Error detection.
    - ➔ Application Layer practically communicates directly with IP layer.
  - Non-connective. Without mechanisms for:
    - Segmentation, reliability, flow control, congestion control, sequence control.
  - Stateless ➔ No need to manage control parameters like:
    - Seq numbers, timers, acks, temporary buffers, etc.
    - ➔ A server may handle many simultaneous clients.
  - UDP introduces **low encapsulation overhead**.

Transport Layer: UDP.

- What type of applications need UDP services?
  - Applications that need themselves to **control which data and when** are.
    - UDP passes data to IP layer as soon as received from application.
  - Real time applications
    - Require transmission speed.
    - Non delay tolerant.
    - Loss tolerant.
  - Fast applications like: network management (SNMP), DNS, IP routing, etc.
  - If required, they must implement services not provided by UDP and pay penalties for it (like reliable transfer in TFTP).

Transport Layer: UDP and TCP .

- Popular internet applications versus transport protocols.

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

Figure 3.6 ♦ Popular Internet applications and their underlying transport protocols

Transport Layer: UDP.

- Potential problems with UDP.
  - UDP may create congestion, it has no congestion control mechanisms:
    - Many simultaneous applications may congest the network and high loses due to buffer overflow.
    - Result:
      - UDP application may have high error rates.
      - TCP sessions may be “suffocated”.
  - Consequences.
    - Some ISP block UDP traffic.
    - TCP is more and more used for transport of multimedia traffic.

Transport Layer : UDP.

- UDP header fields.
  - Source Port (optional, cero if not used*).
    - Nbr of sender port at originating host.
      - Also use as receiver port for response messages, unles otherwise specified.
  - Destination Port.
    - Nbr of receiver port at destination host
  - Length.
    - Size in bytes of UDP segment including headers and body.
  - *Checksum* . Optional* at IPv4, mandatory at IPv6.
    - For error detection .
    - One's complement of the sum of all 16 bit words of UDP segment plus **pseudo header**, with carrying on the lowest bit.
      - See procedure at paragraph 3.3.2, Kurose book.

Transport Layer: UDP.

- How does UDP verifies *checksum*?
  1. Obtains **pseudo header** the received IP packet, carrying UDP segment.
  2. Stores *checksum* value of received UDP segment.
  3. Sets to cero *checksum* field of UDP segment and computes *checksum* (of UDP segment plus pseudoheader*).
     - *Checksum* calculated = *Checksum* received ➔ UDP Segment without error.
     - *Checksum* calculated ≠ *Checksum* receoved ➔ UDP Segment with error.

*: destination IP value for checksum computation is obtained **from receiver host IP address** instead of IP header , to detect routing errors

La capa de Transporte: UDP.

- *Checksum* field
  - Only used for *error detection* *end to end* (not error recovery) because_
    - Intermediate links may not have error detection mechanisms.
    - Errors may be introduced at intermediate nodes.
- What does UDP after checksum error is detected?
  - Corrupted segment is discarded, or
  - Data are passed to the application with a warning.

Transport Layer: UDP.

- UDP summary.

PDU name	Segment/Datagram UDP
Header size (PCI)	8 bytes
Connection oriented	NO
Segmentation of application data	NO
Error detection	Optional
Reliable transport	NO
Sequence control	NO
Flow control	NO
Congestion control	NO