CSC 4350 – Computer Networks

Finish Discussion on Application Layer

Begin Talking about Transport Layer and Its Services

Lecture 10

September 24, 2024

Note

- Material in this lecture is heavily borrowed from Kurose & Ross' "Computer Networking: A Top Down Approach, 8th Edition"

Socket Programming – Creating Network Applications

- Recall: typical network application consists of a pair of programs – client and server
- Two types of network applications
  - Implementation whose operation is specified in a protocol standard, like an RFC
    - Referred to as “open” since the rules specifying its operation can be known by all
  - Proprietary network application
    - Application-layer protocol that has not been openly published
    - A single developer/developer team creates client and server programs and the developer has complete control over what goes in the code
    - Other independent developers will not be able to write code that works with the application
- Decisions, decisions: TCP or UDP?

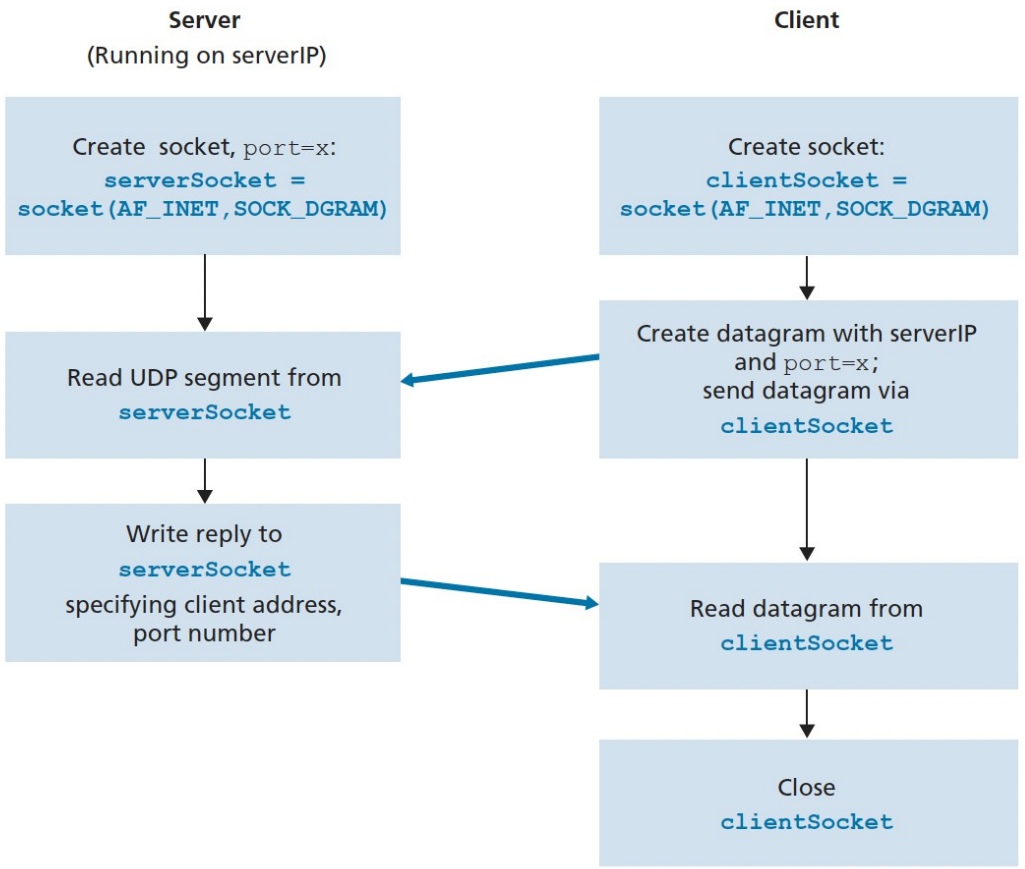
Closer Look – UDP Sockets

- Before the sending process can push a packet of data out the socket door, when using UDP, it must first attach a destination address to the packet
- After the packet passes through the sender's socket, the internet will use this destination address to route the packet through the internet to the socket in the receiving process
- When the packet arrives at the receiving socket, receiving process will retrieve the packet through the socket and then inspect the packet's contents and take appropriate action

UDP Sockets

- What goes into the dest address that is attached to a packet?
  - Dest IP address (route packet through internet to destination)
  - Port number is assigned to application, so included in the dest address
  - Sender's IP address of the source host and port number of source socket
  - But – not usually done by the UDP application code; done automatically by the underlying operating system
- Figure 2.27
  - Client reads a line of characters (data) from its keyboard and sends the data to the server
  - Server receives the data and converts the characters to uppercase
  - Server sends the modified data to the client
  - Client receives the modified data and displays the line on its screen

Figure 2.27 – Client-Server Application Using UDP



Socket Programming – TCP

- Before client and server can start to send data to each other, they first need to handshake and establish a TCP connection
  - One end is attached to the client socket
  - Other end – server socket
- When creating the connection, associate it with the client socket address and the server socket address
- When one side wants to send data to the other, data dropped into the TCP connection via socket
  - Slight difference over UDP, where server must attach dest address to packet
- Server has to be ready/must be running as a process before the client attempts to initiate contact
- Server program must have a special socket that welcomes initial contact from a client process running on some host
- Establish connection, three-way handshake, as discussed before for TCP
  - Server will create a new socket dedicated to that particular client
- Application POV – connection socket is connected directly by a pipe

Figure 2.28 – The TCPServer Process Has 2 Sockets

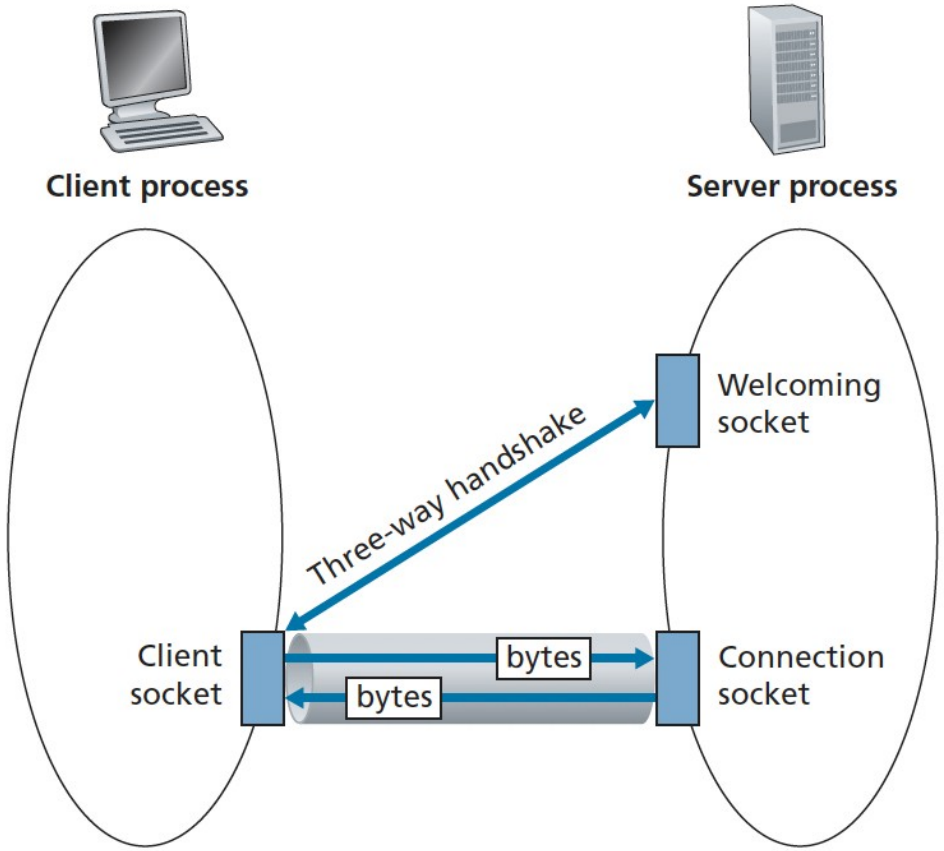
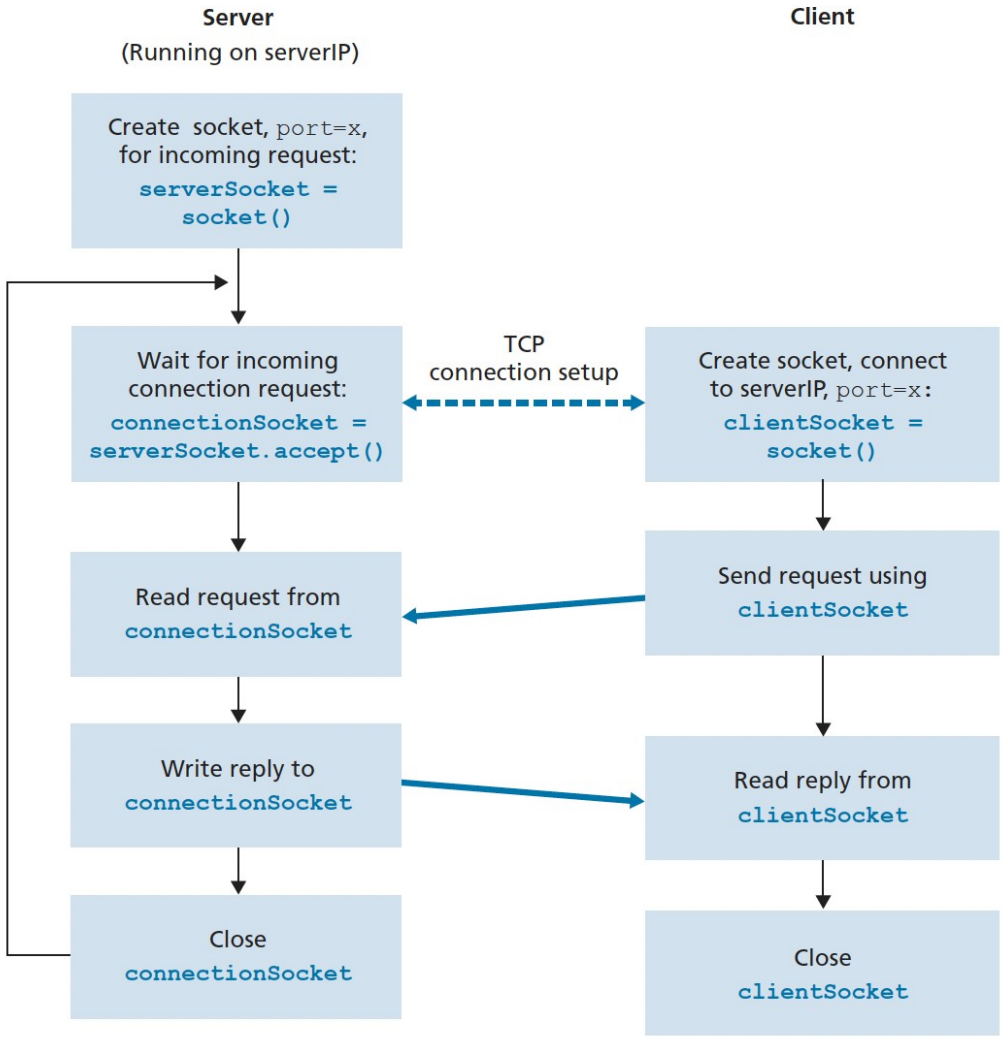


Figure 2.29 – Client-Server Application Using TCP



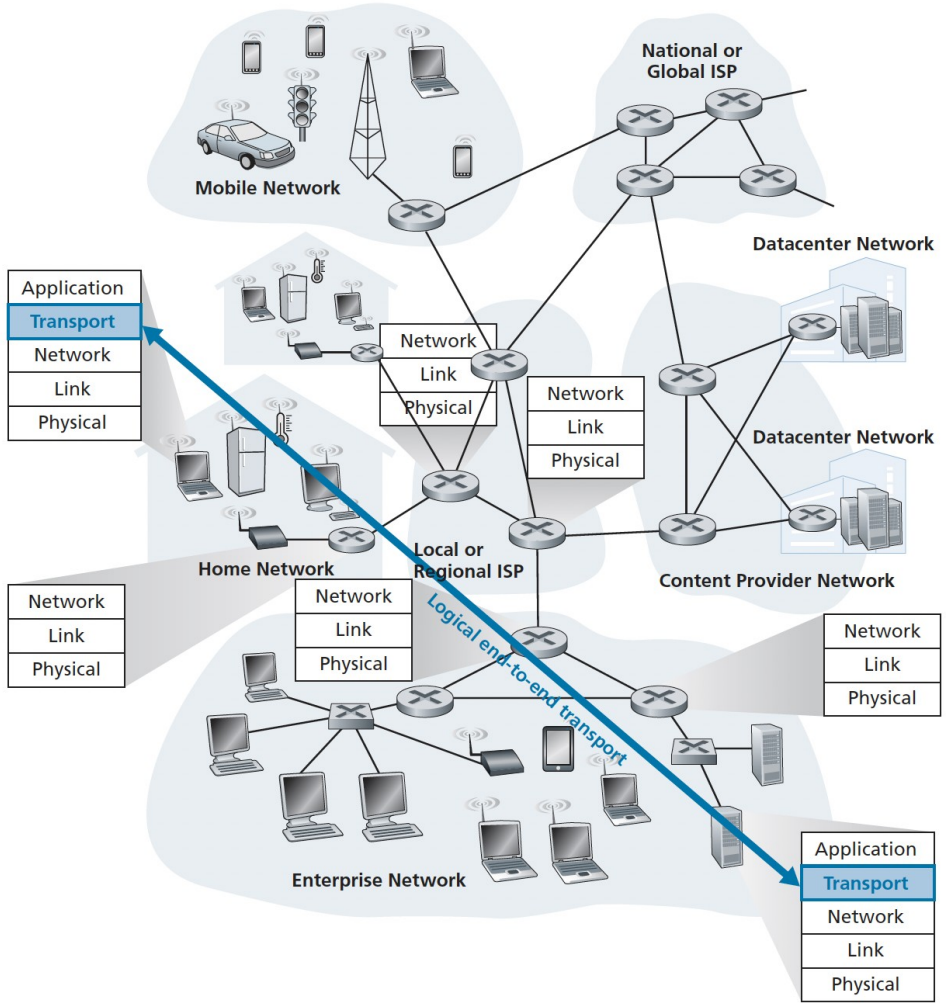
Transport-Layer Services

- Relationship Between Transport and Network Layers
- Overview of the Transport Layer in the Internet

Transport-Layer Services - Introduction

- Transport-layer protocol provides for logical communication between application processes running on different hosts
  - “Logical communication” – from an application’s perspective, it is as if the hosts running the processes were directly connected
- Figure 3.1 (next slide)
  - Implemented in the end systems but not in network routers
  - Sender side
    - Converts the application-layer messages it receives from a sending application process to transport-layer packets, known also as segments
    - Done by breaking application messages into smaller chunks and adding a transport-layer header to each chunk to create the segment
    - Passes segment to network layer
    - Network routers act only on the header information; don’t view contents
  - Receiver side
    - Network layer extracts transport-layer segment and passes it up to the transport layer
    - Transport layer then processes the received segment, making the data in the segment available to the receiving application
- Internet has two transport-layer protocols – TCP and UDP

Figure 3.1 – Transport Layer Providing Logical Rather Than Physical Communication Between Application Processes



Relationship Between Transport, Network Layers

- Transport-layer protocol provides logical communication between processes running on different hosts, network-layer protocol provides logical-communication between hosts
- East Coast, West Coast houses example in text
  - Application messages = letters in envelopes
  - Processes = cousins
  - Hosts (end systems) = houses
  - Transport-layer protocol = Ann and Bill (distribute mail)
  - Network-layer protocol = postal service (including mail carriers)
- Lead-Up to Talking About TCP and UDP

Overview of the Transport Layer in the Internet

- UDP – unreliable, connectionless service to the invoking application
- TCP – reliable, connection-oriented service to the invoking application
- When designing a network application, the application developer must specify one of these two protocols
- Separation of terms (for reference elsewhere)
  - TCP – transport layer packet – segment
  - UDP – datagram
- A paragraph later… (your author's notation)
  - Segments – Transport-Layer packet
  - Datagram – Network-Layer packet

Internet's Network Layer

- Name: IP – Internet Protocol
  - Provides logical communication between hosts
  - Service model: best-effort delivery service
    - IP makes "best effort" to deliver segments between communicating hosts, but no guarantees
      - No guarantee on segment delivery
      - No guarantee on orderly delivery of segments
      - No guarantee on data in the segments
  - Unreliable service
  - Biggest fundamental responsibility for TCP, UDP – extend IP's delivery service between two end systems to a delivery service between two processes running on the end systems
    - Transport layer multiplexing/demultiplexing
  - TCP/UDP provide integrity checking – error detection fields in headers
  - Remember that UDP is not reliable

Internet’s Network Layer

- TCP – several additional services to applications
  - Reliable data transfer – via flow control, sequence numbers, acknowledgments and timers
    - Ensures data gets from sender to receiver
    - Builds off of IP’s unreliable service
  - Congestion Control
    - More of a service for the Internet, as a whole
    - Prevents any one TCP connection from swamping the links and routers between hosts with an excessive amount of traffic
    - Tries to give each link an equal share of link bandwidth – regulate the rate at which sending sides of connections can send traffic into the network
    - Not available via UDP

Multiplexing and Demultiplexing

- Connectionless Multiplexing and Demultiplexing
- Connection-Oriented Multiplexing and Demultiplexing
- Web Servers and TCP

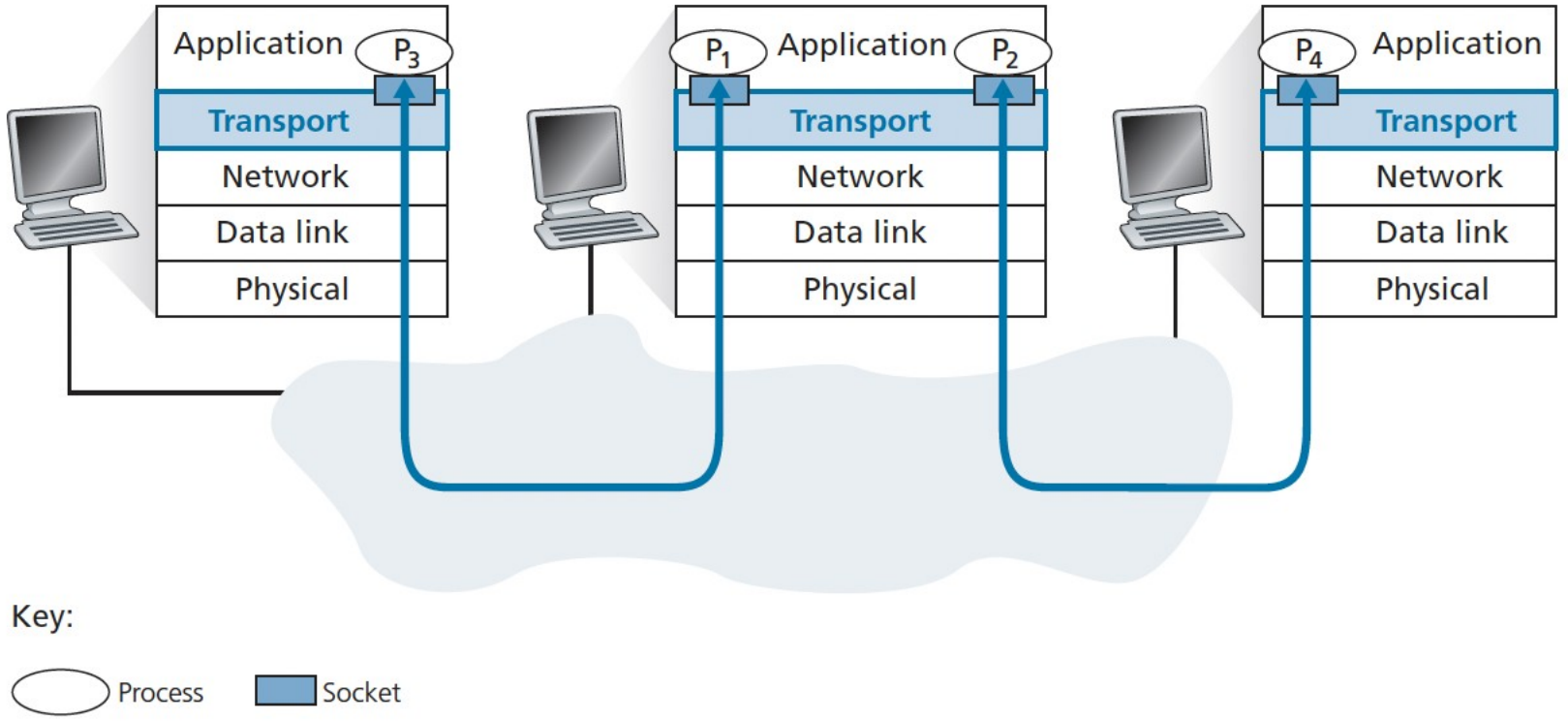
Multiplexing and Demultiplexing

- Author – keeping discussion on the Internet, but useful for any computer network
- Transport layer – responsibility of delivering the data in segments from the network layer just below
  - Has responsibility of delivering data in these segments to the appropriate application process in the host
  - Example: Using web browser, running ftp, utilizing two Telnet sessions
    - When transport layer in computer receives data from the network layer, it needs to direct the received data to one of these processes
- A process can have 1+ sockets
  - Transport layer in Figure 3.2 does not actually deliver data directly to a process, but 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
    - Format depends on whether the socket is a UDP or TCP socket

Multiplexing and Demultiplexing

- Receiving host directs an incoming transport-layer segment to the appropriate socket
  - Each transport-layer segment has a set of fields in the segment for this purpose
  - Receiving end – transport layer examines these fields to identify the receiving socket and then directs the segment to that socket
  - Demultiplexing – job of delivering the data in a transport-layer segment to the correct socket
  - Multiplexing – job of gathering data chunks at the source host from different sockets, encapsulating each data chunk with header information, to create segments, and passing the segments to the network layer

Figure 3.2 – Transport-Layer Multiplexing/Demultiplexing



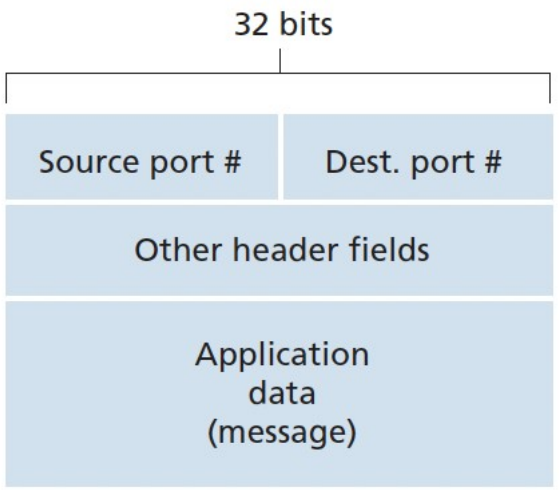
Multiplexing/Demultiplexing

- Transport layer in middle host – Figure 3.2 – must demultiplex segments arriving from the network layer below either P1 or P2
  - Done by directing the arriving segment's data to the corresponding process' socket
  - Must also gather outgoing data from these sockets, form transport-layer segments, and pass these segments down to the network layer
  - Example: get a bunch of mail at your house
    - Demultiplexing – look at who letters are addressed to, then hand-deliver
    - Multiplexing – collecting letters from others in the house and gives the collected mail to the mail carrier

Multiplexing/Demultiplexing – Host Perspective

- Figure 3.3
- Transport-layer multiplexing requires
  - Sockets have unique identifiers
  - Each segment has special fields that indicate the socket to which the segment is to be delivered
    - Source port number field
    - Destination port number field
- Each number is a 16-bit number, ranging from – to 65535
- Port numbers between 0 to 1023 – well-known port numbers
  - Reserved for use by well-known application protocols, like http, ftp, etc.
- Demultiplexing
  - Each socket in the host could be assigned a port number
  - When a segment arrives at the host, transport layer examines destination port number in the segment, directs segment to the corresponding socket
  - Segment's data passes through the socket into the attached process
  - Above description – how UDP implements…

Figure 3.3 – Source and Destination Port-Number Fields in a Transport-Layer Segment



Connectionless Multiplexing and Demultiplexing

- When a UDP socket is created, transport layer automatically assigns a port number to the socket
  - In the range of 1024 to 65535
  - Not currently in use by any other UDP port in the host
- If the application developer writing the code were implementing the server side of a “well-known protocol,” developer would have to assign the corresponding well-known port number
  - Client side – lets transport layer automatically/transparently assign the port number
  - Server side – assigns a specific port number

UDP Multiplexing/Demultiplexing

- Suppose a process in Host A, with UDP port 19157 wants to send a chunk of application data to a process with UDP port 46428 in Host B
- Transport layer in Host A creates a transport-layer segment that includes the application data, source port number, destination port number, and two other values (not necessary to get into here)
- Transport layer passes the resulting segment to the network layer
- Network layer encapsulates the segment in an IP datagram and makes a best-effort attempt to deliver the segment to the receiving host
- If the segment arrives at the receiving Host B, transport layer at the receiving host examines destination port number and delivers the segment to the appropriate socket
- Note: Host B could be running multiple processes, each with its own UDP socket and associated port number
- As segments arrive, Host B demultiplexes each segment to the appropriate socket by examining the port number

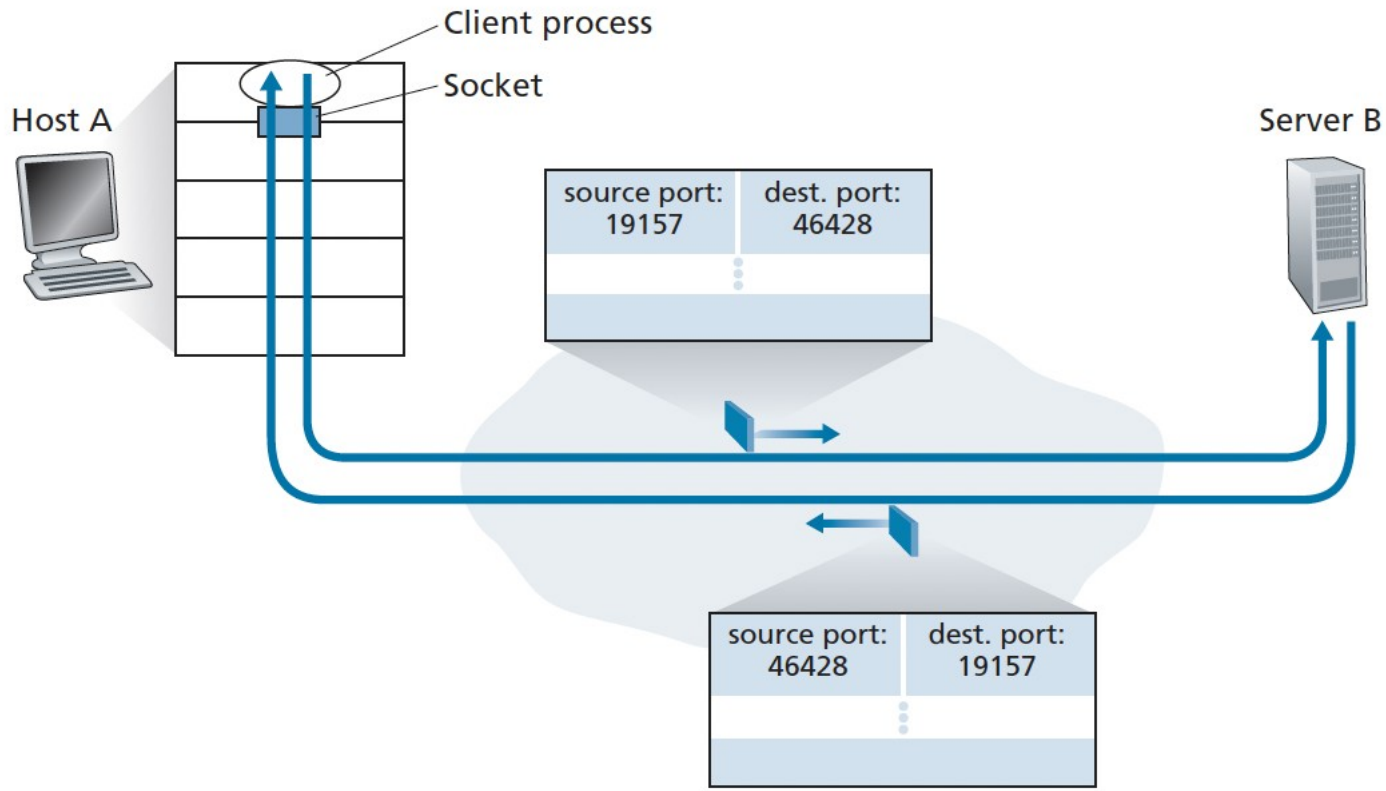
UDP Multiplexing/Demultiplexing

- UDP socket is fully identified by a two-tuple
  - Destination IP address
  - Destination port number
- If two UDP segments have different IP addresses and/or source port numbers, but have same IP address and destination port number, both will be directed to the same address process via the same socket
- Purpose of source port number – serves as “return address” when B wants to send something back to A

Connection-Oriented Multiplexing and Demultiplexing

- TCP sockets and TCP connection establishment
  - TCP socket is identified by a four-tuple
    - Source IP address
    - Source port number
    - Destination IP address
    - Destination port number
- When a TCP segment arrives from the network to a host, the host demultiplexes with the four values to send segment to appropriate socket
- Two arriving TCP segments with different source IP addresses or port numbers will be directed to two different sockets
- Server host may support many simultaneous TCP connection sockets
  - Each socket attached to a process and with each socket identified by its own four-tuple
  - When segment arrives at the host, all four fields are used to direct the segment to the appropriate socket

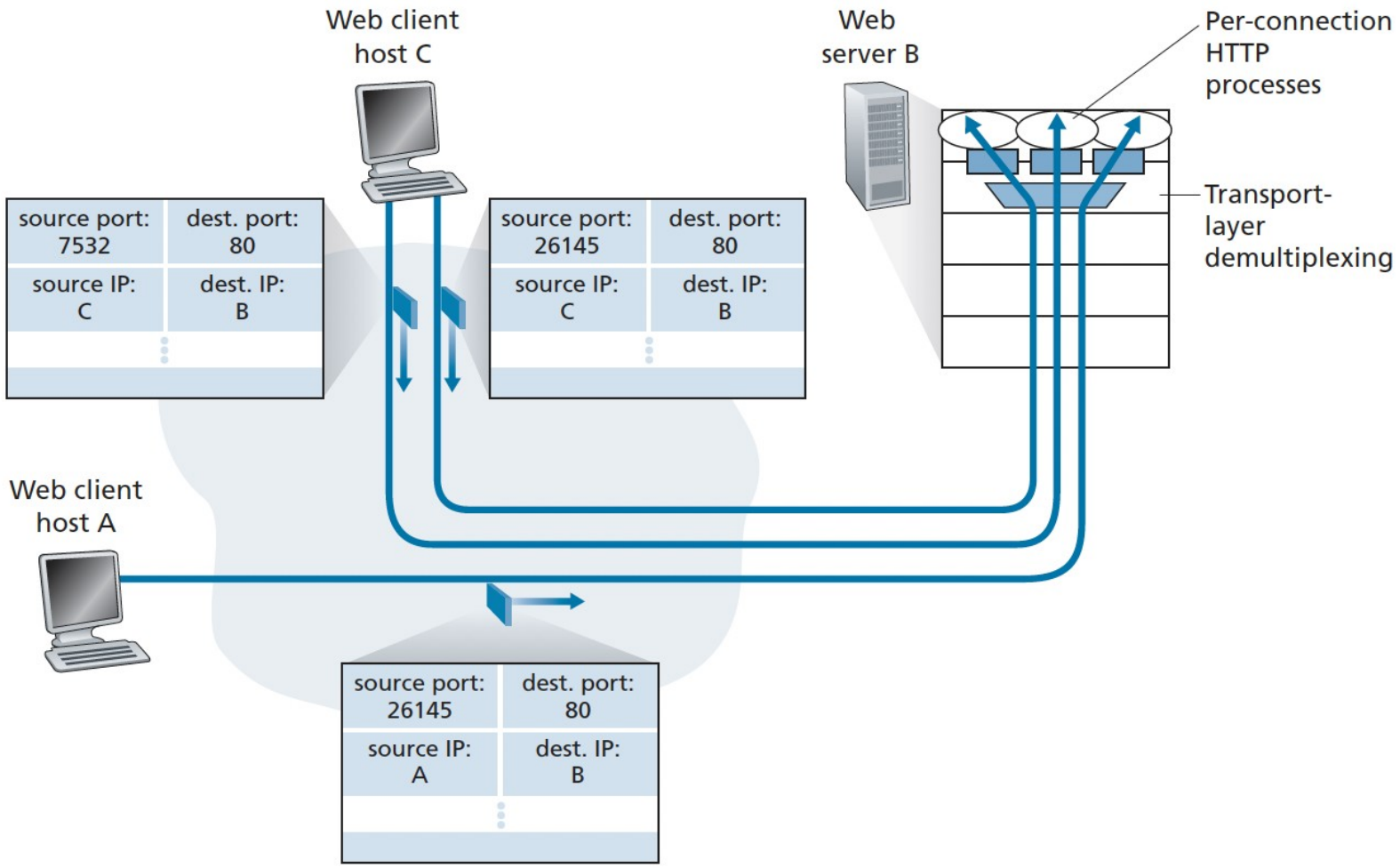
Figure 3.4 – The Inversion of Source and Destination Port Numbers



TCP and Figure 3.5

- Host C initiates two http sessions to server B, and Host A initiates one http session to B
- Hosts A and C and server B have their own unique IP address
- Host C assigns two different source port numbers to its two http connections (26145 and 7532)
- Because Host A is choosing source port numbers independently of C, it might also assign a source port of 26145 to its http connection
  - Not an issue – server B will still be able to correctly demultiplex the two connections having the same source port number since the two connections have two different source IP addresses

Figure 3.5 – Two Clients, Same Destination Port Number (80), To Communicate with Same Web Server Application



Web Servers and TCP

- Host runs on a web server, like an Apache web server, on port 80
  - When clients send segments to the server, all segments will have destination port 80
  - Both the initial connection-establishment segments and segments carrying http request messages will have destination port 80
  - Different clients will have different source IP addresses and port numbers
- Figure 3.5
  - Web server that creates a new process for each connection
  - Each of these processes has its own connection socket through which http requests arrive and responses sent
  - There is not always a 1:1 correspondence between connection sockets and processes
    - Web servers often use only one process and create a new thread with a new connection socket for each client connection

Figure 3.5 Discussion – Continued

- If client and server are using persistent http, then throughout the duration of the persistent connection the client and server exchange http messages via same server socket
- If non-persistent http
  - A new socket is created and later closed for every request/response and a new socket is created and closed for every request/response
  - Can impact performance of a busy web server