CSE/EE 461 – Module 11

Connections

This Time

- More on the Transport Layer

- Focus
  - How do we <u>connect processes</u>?

- Topics
  - Naming processes
  - Connection setup / teardown
  - Flow control

| |
|---|
| Application |
| Presentation |
| Session |
| Transport |
| Network |
| Data Link |
| Physical |

1

Naming Processes/Services

- Process here is an abstract term for your Web browser (HTTP), Email servers (SMTP), hostname translation (DNS), RealAudio player (RTSP), etc.

- How do we identify for remote communication?
  - Process id or memory address are OS-specific and transient

- So TCP and UDP use Ports
  - 16-bit integers representing mailboxes that processes "rent"
    - typically from OS
  - Identify endpoint uniquely as (IP address, protocol, port)
    - OS converts into process-specific channel, like "socket"

Processes as Endpoints
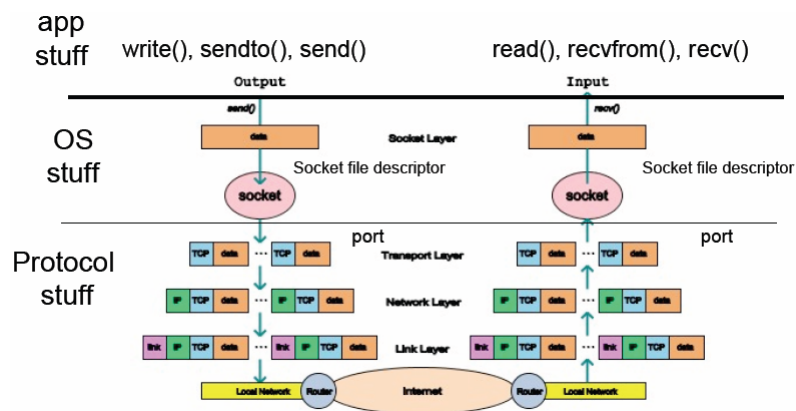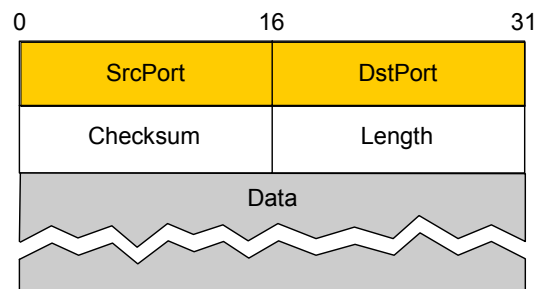
Picking Port Numbers

- We still have the problem of allocating port numbers
  - What port should a Web server use on host X?
  - To what port should you send to contact that Web server?

- Servers typically bind to "well-known" port numbers
  - e.g., HTTP 80, SMTP 25, DNS 53, … look in /etc/services
  - Ports below 1024 reserved for "well-known" services

- Clients use OS-assigned temporary (ephemeral) ports
  - Above 1024, recycled by OS when client finished

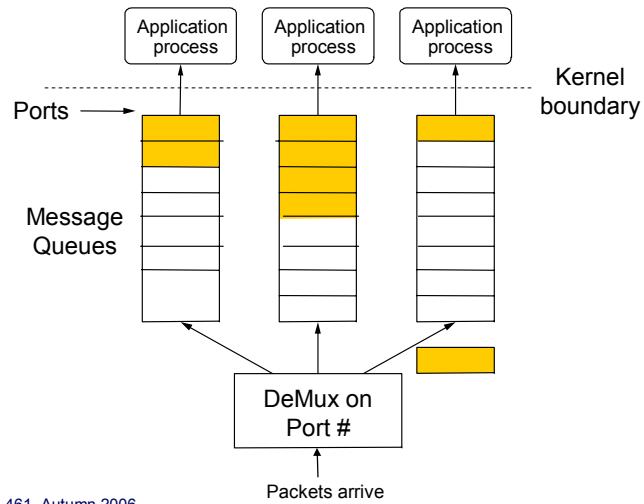## User Datagram Protocol (UDP)

- Provides message delivery between processes
  - Source port filled in by OS as message is sent
  - Destination port identifies UDP delivery queue at endpoint

0	16	31
SrcPort	DstPort	
Checksum	Length	
Data		

UDP Delivery

Application process | Application process | Application process
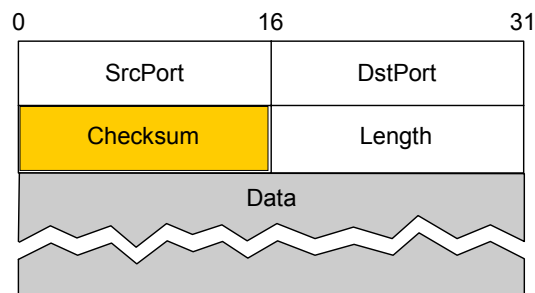
Kernel boundary

Ports →

Message Queues

DeMux on Port #

Packets arrive

UDP Checksum

- UDP includes optional protection against errors
  - Checksum intended as an end-to-end check on delivery
  - So it covers data, UDP header, and IP pseudoheader

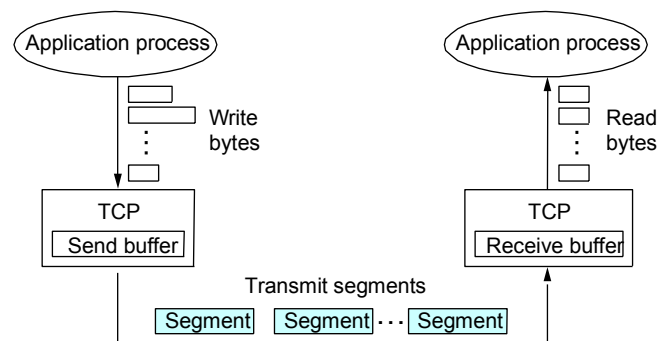| 0	16	31
SrcPort		DstPort
Checksum		Length
Data		

Transmission Control Protocol (TCP)

- Reliable bi-directional bytestream between processes
  - Message boundaries are not preserved

- Connections
  - Conversation between endpoints with beginning and end
- Flow control
  - Prevents sender from over-running receiver buffers
- Congestion control
  - Prevents sender from over-running network buffers

TCP Delivery

TCP Header Format

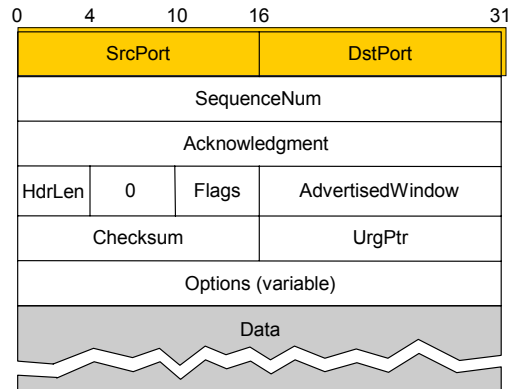- Ports plus IP addresses identify a connection

0	4	10	16	31
SrcPort			DstPort	
SequenceNum				
Acknowledgment				
HdrLen	0	Flags	AdvertisedWindow	
Checksum			UrgPtr	
Options (variable)				
Data				

TCP Header Format

- Sequence, Ack numbers used for the sliding window

0	4	10	16	31
SrcPort			DstPort	
SequenceNum				
Acknowledgment				
HdrLen	0	Flags	AdvertisedWindow	
Checksum			UrgPtr	
Options (variable)				
Data				
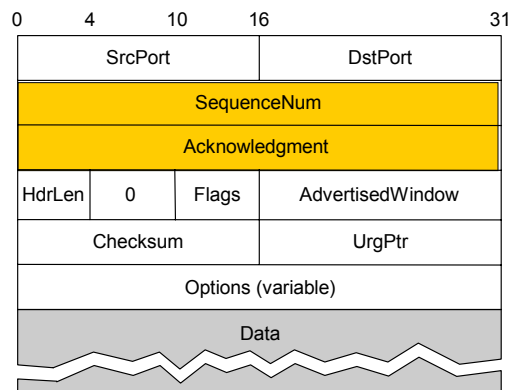
TCP Header Format

- Flags may be URG, ACK, PUSH, RST, SYN, FIN

0	4	10	16	31
SrcPort			DstPort	
SequenceNum				
Acknowledgment				
HdrLen	0	Flags	AdvertisedWindow	
Checksum			UrgPtr	
Options (variable)				
Data				
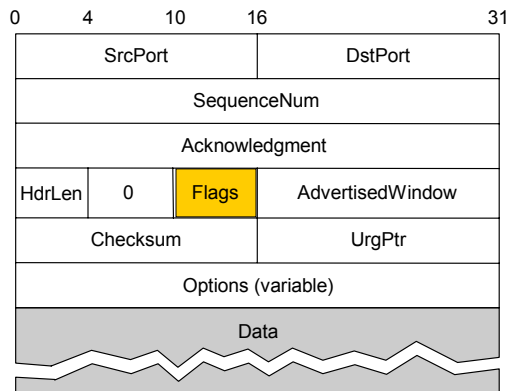
M11.13

TCP Header Format

- Advertised window is used for flow control

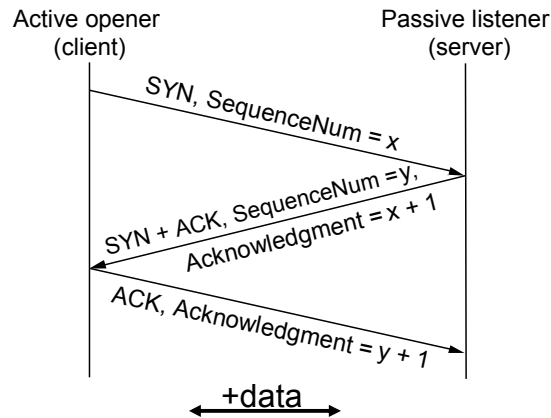| 0	4	10	16	31
SrcPort			DstPort	
SequenceNum				
Acknowledgment				
HdrLen	0	Flags	AdvertisedWindow	
Checksum			UrgPtr	
Options (variable)				
Data				
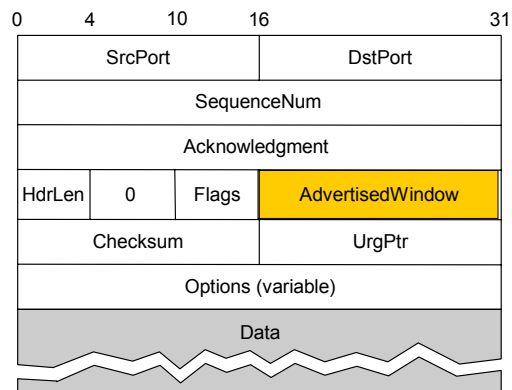
M11.14

TCP Connection Establishment

- Both connecting and closing are (slightly) more complicated than you might expect

- That they *can* work is reasonably straightforward

- Harder is what to do when things go wrong
  - TCP SYN+ACK attack

- Close looks a bit compicated because both sides have to close to be done
  - Conceptually, there are two one-way connections
  - Don't want to hang around forever if other end crashes

TCP Connection Establishment

- Both sender and receiver must be ready before we start to transfer the data
  - Sender and receiver need to agree on a set of parameters
  - e.g., the Maximum Segment Size (MSS)

- This is "signaling"
  - It sets up state at the endpoints
  - Compare to "dialing" in the telephone network

- In TCP a Three-Way Handshake is used

Three-Way Handshake

- Opens both directions for transfer

Active opener
(client)

Passive listener
(server)

SYN, SequenceNum = x

SYN + ACK, SequenceNum = y,
Acknowledgment = x + 1

ACK, Acknowledgment = y + 1
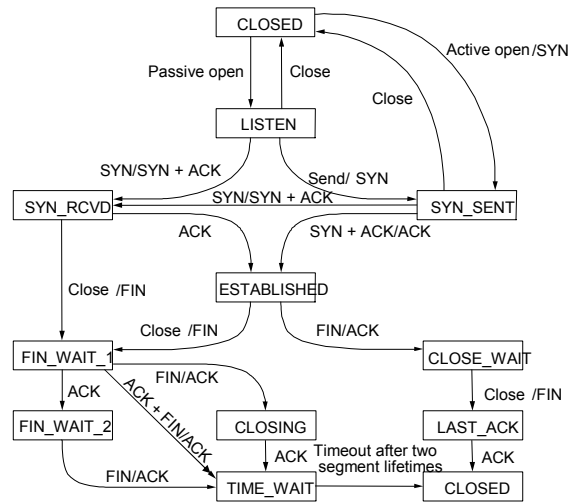
+data

Some Comments

- We could abbreviate this setup, but it was chosen to be robust, especially against delayed duplicates
  - Three-way handshake from Tomlinson 1975

- Choice of changing initial sequence numbers (ISNs) minimizes the chance of hosts that crash getting confused by a previous incarnation of a connection

- But with random ISN it actually proves that two hosts can communicate
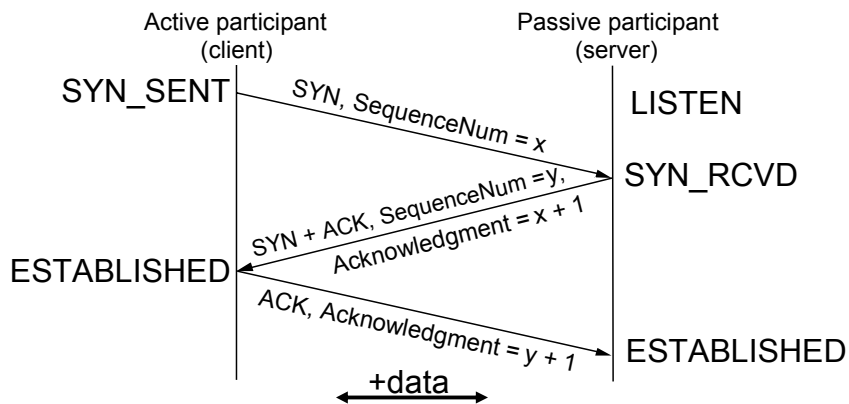 - Weak form of authentication

TCP State Transitions

CSE/EE 461, Autumn 2006

M11.19

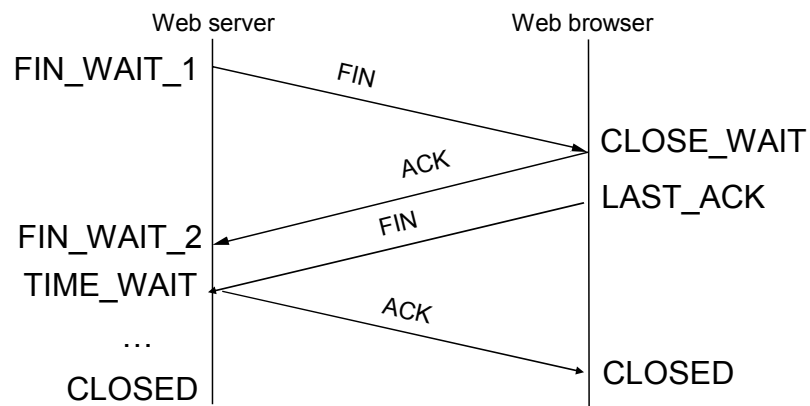
Again, with States

CSE/EE 461, Autumn 2006

M11.20

10

Connection Teardown

- Orderly release by sender and receiver when done
  – Delivers all pending data and "hangs up"

- Cleans up state in sender and receiver

- TCP provides a "symmetric" close
  – both sides shutdown independently

TCP Connection Teardown

Web server   Web browser

FIN_WAIT_1 ——— FIN ———→ CLOSE_WAIT

←——— ACK ———

LAST_ACK

FIN_WAIT_2 ←——— FIN ———

TIME_WAIT

——— ACK ———→
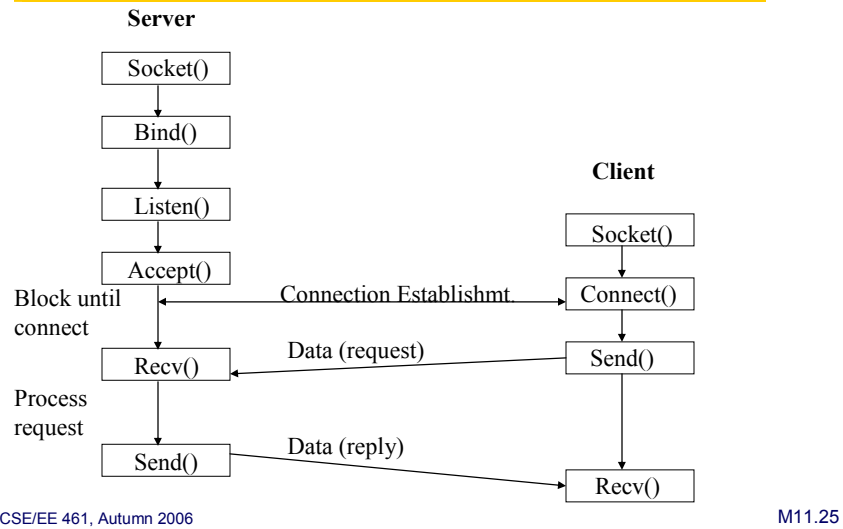
…

CLOSED   CLOSED

The TIME_WAIT State

- We wait 2MSL (two times the maximum segment lifetime of 60 seconds) before completing the close

- Why?

- ACK might have been lost and so FIN will be resent
- Could interfere with a subsequent connection

Berkeley Sockets interface

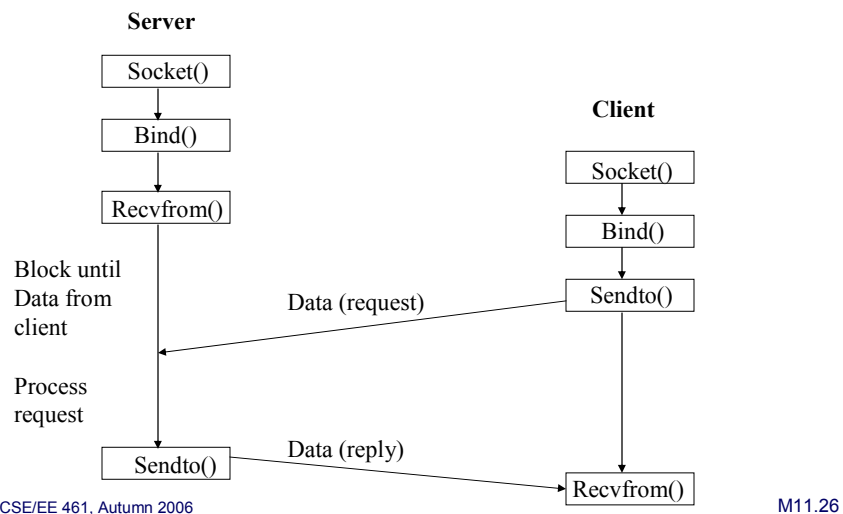- Networking protocols implemented in OS
  - OS must expose a programming API to applications
  - most OSs use the "socket" interface
  - originally provided by BSD 4.1c in ~1982.

- Principle abstraction is a "socket"
  - a point at which an application attaches to the network
  - defines operations for creating connections, attaching to network, sending and receiving data, closing connections

TCP (connection-oriented)

Server

Socket()
↓
Bind()
↓
Listen()

Client

↓
Accept()

Block until connect

Connection Establishmt.

Socket()
↓
Connect()

Recv()

Data (request)

Send()

Process request

Send()

Data (reply)

Recv()

M11.25

UDP (connectionless)

Server

Socket()
↓
Bind()
↓
Recvfrom()

Client

Block until Data from client

Data (request)

Socket()
↓
Bind()
↓
Sendto()

Process request

Sendto()

Data (reply)

Recvfrom()

M11.26

Using Sockets: UDP

- import java.net.*;
- UDP sockets:
  - new DatagramSocket();   // binds to ephemeral port number
  - new DatagramSocket(port); // tries to bind to 'port'
- DatagramPacket
  - Unit of transfer between application and networking software
  - new DatagramPacket(byte[] buf, int len);
  - new DatagramPacket(byte[] buf, int len, InetAddress addr, int port);

- Sending data:
  - Construct a DatagramPacket
  - Set its data field, and its address components
  - myDatagramSocket.*send*(myDatagramPacket)

Java / UDP

- Java also has an interface supporting *connect(SocketAddr addr)*, but it's a layer above UDP
  - Filters incoming packets not from *addr*
  - Filters outgoing packets not from *addr*

- Performance / correctness issue:
  - Is a copy of the data portion of a `DatagramPacket` made when `send()` is invoked, or is a reference to the `byte[] buf` kept?

- Blocking vs. non-blocking IO
  - Non-blocking options
    1. import java.net.*;
       - DatagramSocket.setSOTimeout(int timeout);
    2. import java.nio.*;
       - More general (complicated) support

Using Sockets: TCP

- The TCP distinction between passive and active open is embedded in the (typical) socket interfaces
  - There are two kinds of sockets:
    - Socket
    - ServerSocket

- Server starts, creates a server socket, binds it to a local port, and listens for a client to connect
- Client starts, creates a socket on an ephemeral port, and connects to the server socket
- As a result of the connection, the server socket creates a *new* socket to return to the application
  - Provides a handy way to identify/name a single flow in the application code

---

TCP Server-side: Java

- Create:
  - `ServerSocket ss = new ServerSocket();`
  - `ServerSocket ss = new ServerSocket(port);`

- Listen:
  - `Socket s = ss.accept();`

TCP Client side: Java

- Create:
  - `Socket s = new Socket();`

- Connect:
  - `s.connect(serverAddress);`
  - `S.connect(serverAddress, timeout);`

- Use:
  - It's Java, the sockets support streams, the mind boggles
  - `BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream()));`
    - `in.readLine();`
  - `PrintWriter out = new PrintWriter(s.getOutputStream(), true);`
    - `Out.print(data);`
  - `OutStream outStream = s.getOutputStream();`
    - `outStream.write( buf, 0, n);  // byte[] buf for n bytes starting at offset 0`

Key Concepts

- We use ports to name processes in TCP/UDP
  - "Well-known" ports are used for popular services
- Connection setup and teardown complicated by the effects of the network on messages
  - TCP uses a three-way handshake to set up a connection
  - TCP uses a symmetric disconnect