Internet Transport Protocols

- Two Transport Protocols Available
  - *Transmission Control Protocol (**TCP**)*
    - *connection oriented*
    - *most applications use TCP*
  - *User Datagram Protocol (**UDP**)*
    - *connectionless*

Transport layer addressing

- Communications endpoint addressed by:
  - IP address (32 bit) **in IP Header**
  - Port number (16 bit) **in TP Header**[1]
  - Transport protocol (TCP or UDP) **in IP Header**

[1] TP => Transport Protocol (UDP or TCP)

Some standard services and port numbers

PORT NUMBERS
(last updated 2004-01-16)
http://www.iana.org/assignments/port-numbers

service	tcp	udp
echo	7	7
daytime	13	13
netstat	15	
ftp-data	20	
ftp	21	
telnet	23	
smtp	25	
time	37	37
domain	53	53
finger	79	
http	80	
pop-2	109	
pop	110	
sunrpc	111	111
uucp-path	117	
nntp	119	
talk		517

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 the application
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

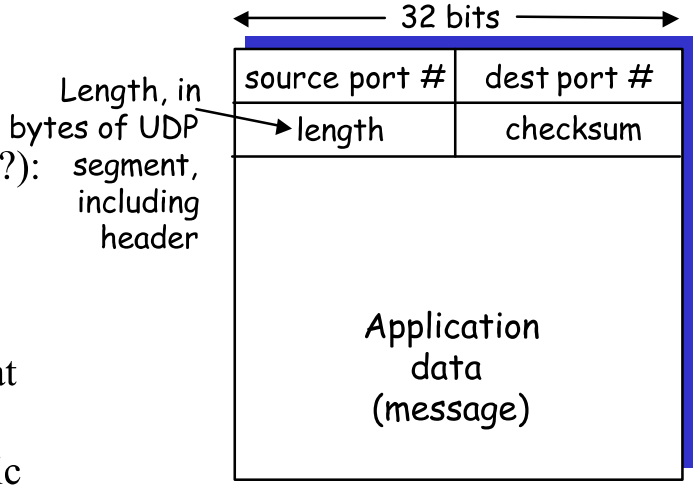
- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

UDP Port Management

- Source (client)
  - Obtains a free port number
  - Specifies port of destination (server)
- Destination
  - Receives datagram
  - Sends datagram to destination IP:port
  - Can send replies to source IP:port

UDP: more

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



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 may be errors nonetheless?* More later ….

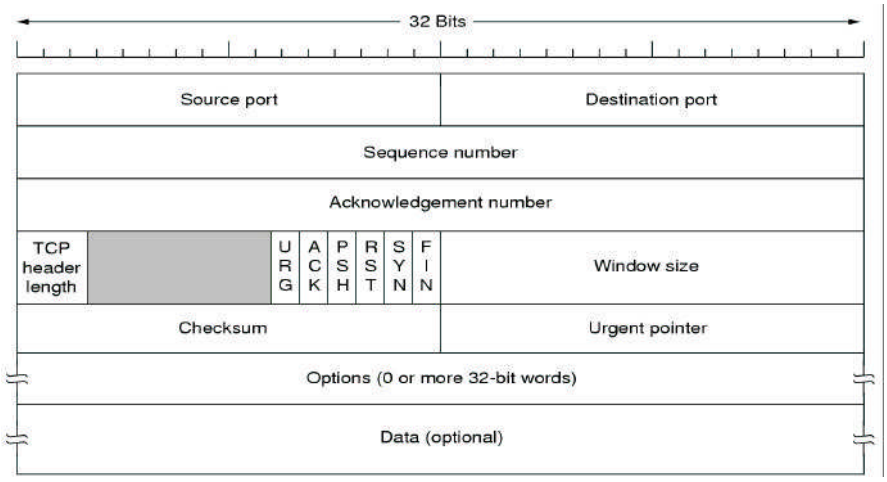
Transmission Control Protocol (TCP)

- Connection-oriented service
- Point-to-point
- Full-duplex communication
- Stream interface (*no message boundary*!)
- Stream divided into segments for transmission
- Each segment encapsulated in IP datagram
- Uses protocol ports to identify applications

TCP Port Management

- When a connection is established
  - Source (client)
    - Obtains a free port number
    - Specifies IP:port of destination (server)
  - Destination
    - Receives connection request
    - Sends data to destination IP:port
- The 4-tuple (source IP:port, destination IP:port) identifies where data goes

TCP Segment



- Sequence number specifies where in stream data belongs
- Few segments contain options

TCP Segment Format

- **Segment divided into two parts**
  - Header
  - Payload area (zero or more bytes of data)

- **Header contains**
  - Protocol port numbers to identify
    - Sending application
    - Receiving application
  - Bits to specify items such as
    - SYN
    - FIN
    - ACK
  - Fields for window advertisement, acknowledgment, etc.

Reliability in an Unreliable World

- IP offers best-effort (unreliable) delivery
- TCP uses IP
- TCP provides completely reliable transfer
- How is this possible? How can TCP realize:
  - Reliable connection startup?
  - Reliable data transmission?
  - Graceful connection shutdown?

Reliable Data Transmission

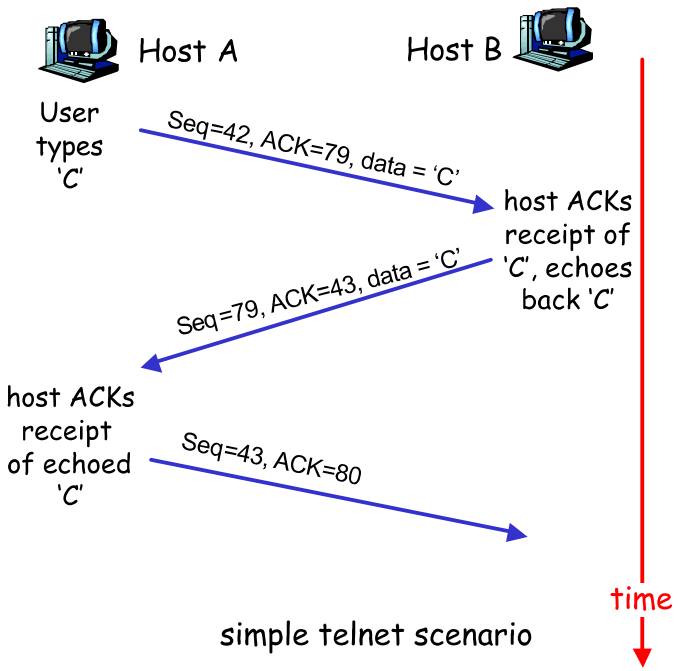
- Positive acknowledgment
  - Receiver returns short message when data arrives
  - Called *acknowledgment*
- Retransmission
  - Sender starts timer whenever message is transmitted
  - If timer expires before acknowledgment arrives, sender *retransmits* message

TCP seq. #'s and ACKs

Seq. #'s:
- byte stream "number" of first byte in segment's data

ACKs:
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments
- A: TCP spec doesn't say, - up to implementer



simple telnet scenario

Timing Problem!

The delay required for data to reach a destination and an acknowledgment to return depends on traffic in the internet as well as the distance to the destination. Because it allows multiple application programs to communicate with multiple destinations concurrently, TCP must handle a variety of delays that can change rapidly.

How does TCP handle this .....

Solving Timing Problem

- Keep estimate of round trip time on each connection
- Use current estimate to set retransmission timer
- Known as ***adaptive retransmission***
- Key to TCP's success

TCP Flow Control

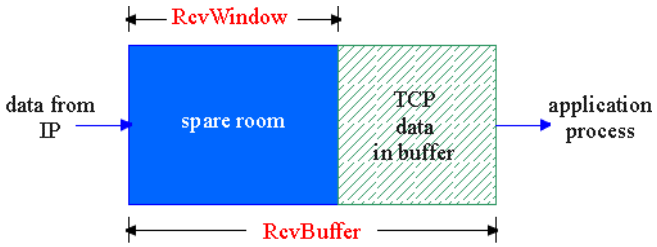
- Receiver
  - Advertises available buffer space
  - Called *window*
- Sender
  - Can send up to entire window before ACK arrives
- Each acknowledgment carries new window information
  - Called *window advertisement*
  - Can be zero (called *closed window*)
- Interpretation: I have received up through X, and can take *Y* more octets

TCP Flow Control

flow control
sender won't overrun receiver's buffers by transmitting too much, too fast

`RcvBuffer` = size or TCP Receive Buffer

`RcvWindow` = amount of spare room in Buffer



receiver buffering

receiver: explicitly informs sender of (dynamically changing) amount of free buffer space
  - **`RcvWindow` field** in TCP segment

sender: keeps the amount of transmitted, unACKed data less than most recently received **`RcvWindow`**

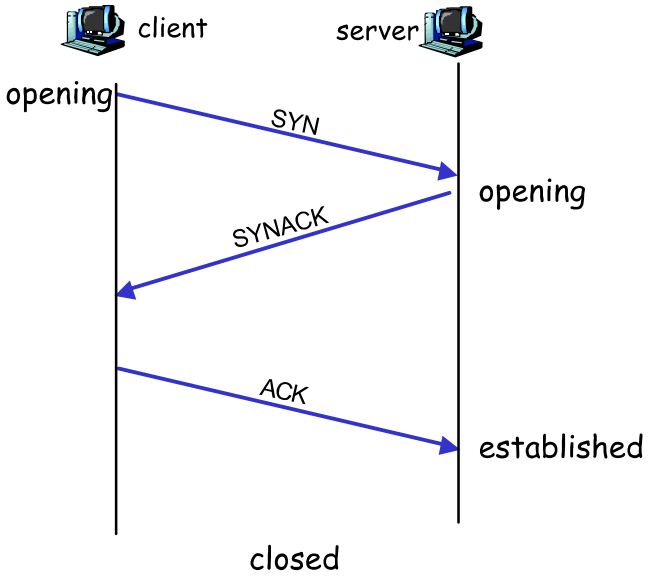
Why Startup/ Shutdown Difficult?

- Segments can be
  - Lost
  - Duplicated
  - Delayed
  - Delivered out of order
  - Either side can crash
  - Either side can reboot
- Need to avoid duplicate ‘‘shutdown’’ message from affecting later connection

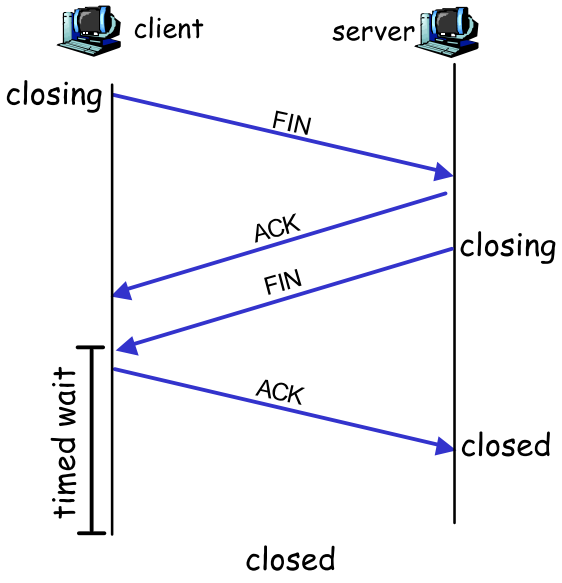
TCP’s Startup/ Shutdown Solution

- Uses three-message exchange known as *3-way handshake*
- Necessary and sufficient for
  - Unambiguous, reliable startup
  - Unambiguous, graceful shutdown
- *SYN* used for startup, *FIN* used for shutdown

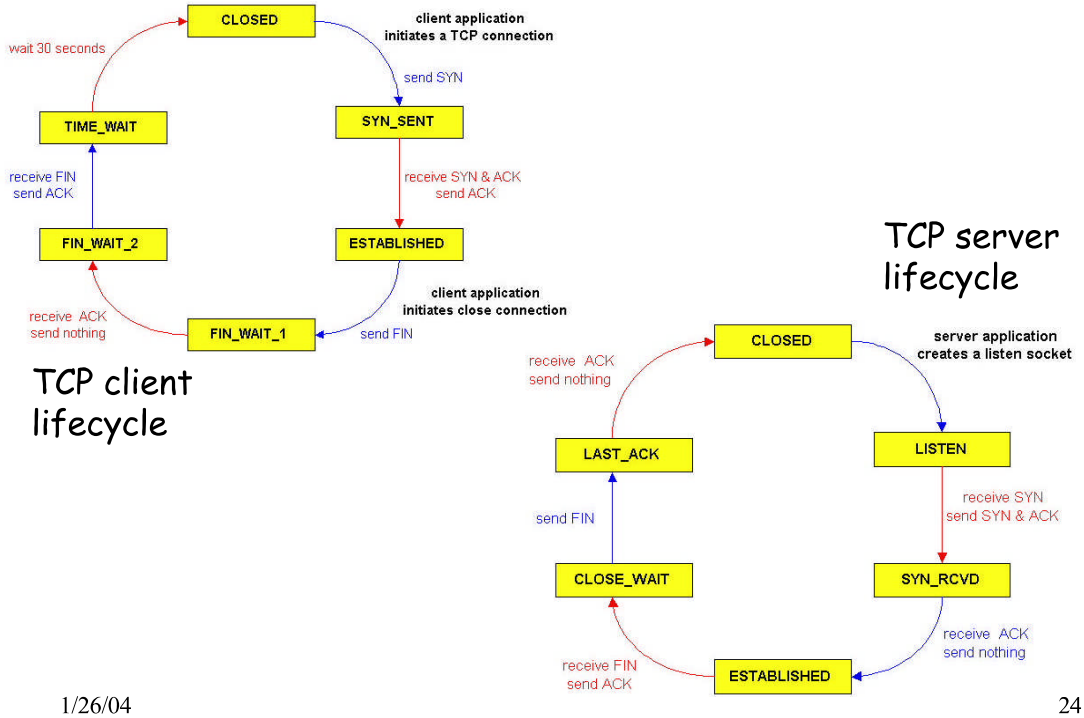
TCP Connection Management (OPEN)



TCP Connection Management (CLOSE)



TCP Connection Management (cont)



Transport Protocol Summary

- Transport protocols fit between applications and Internet Protocol
- Two transport protocols in TCP/IP suite
  - User Datagram Protocol (UDP)
  - Transmission Control Protocol (TCP)
- UDP
  - Unreliable
  - Message-oriented interface
- TCP
  - Major transport protocol used in Internet
  - Complete reliability
  - Stream-oriented interface
  - Uses adaptive retransmission