CSE 3461: Introduction to Computer Networking and Internet Technologies

# Transport Protocols

Presentation H

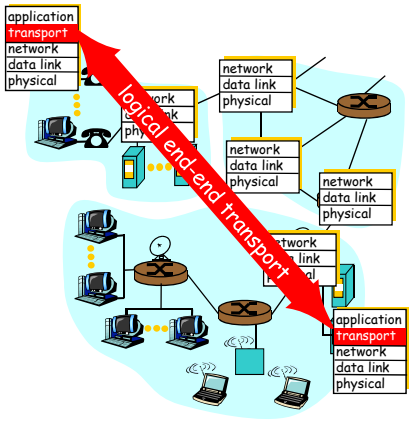
Gojko Babić

Study: 20.1, 20.2, 20.4

10-30-2012

## Transport services and protocols

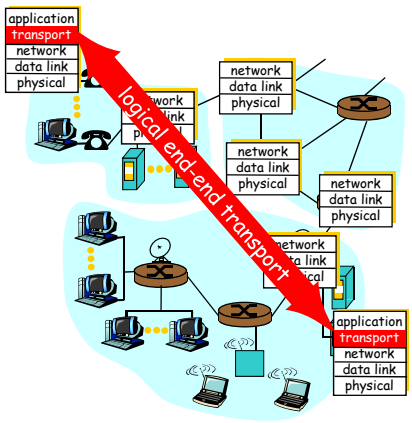
- provide *logical communication* between app' processes running on different hosts
- transport protocols run in end systems
- transport vs network layer services:
- *network layer:* data transfer between end systems
- *transport layer:* data transfer between processes
  - relies on, enhances, network layer services



# Transport-layer protocols

Internet transport services:

- reliable, in-order unicast delivery (TCP)
  - congestion
  - flow control
  - connection setup
- unreliable ("best-effort"), unordered unicast or multicast delivery: UDP
- services not available:
  - real-time
  - bandwidth guarantees
  - reliable multicast




d. xuan

3

# TCP: Transmission Control Protocol

- Connection oriented
- Reliable communication between pairs of processes
- Across variety of reliable and unreliable networks and internets
- Source IP address + port number and destination IP address + port number uniquely determine a TCP connection.
- Problems to resolve:
  - Ordered Delivery
  - Flow control
  - Retransmission strategy
  - Duplication detection
  - Connection establishment
  - Connection termination
  - Crash recovery

g. babic Presentation H 4

# TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*


- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver



# TCP Header

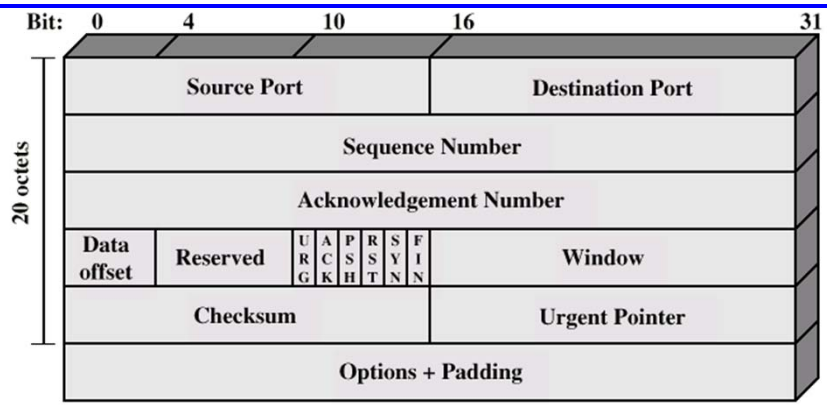



**Figure 2.3**

ACK = 1 → *Acknowledgement Number* present
SYN = 1 → Connection Establishment Request/Response
RST = 1 → Connection Rejected
FIN = 1 → Connection Termination
URG and PSH generally not used

# Credit Based Flow Control

- Improvement over fixed sliding window protocol
- Greater control on reliable network and more effective on unreliable network
- Decouples flow control from acknowledgment
  - — May acknowledge without granting credit and vice versa
- Each octet (byte) has its sequence number
- When sending a data segment, *Sequence Number* is that of first octet (byte) in segment
- ACK includes *Acknowledge Number* AN=i, and *Window* W=j
- This means all octets through *Sequence Number* SN=i-1 are acknowledged, i.e. next expected octet is i
- And permission to send additional window of W=j octets, i.e. octets through i+j-1



# Credit Allocation

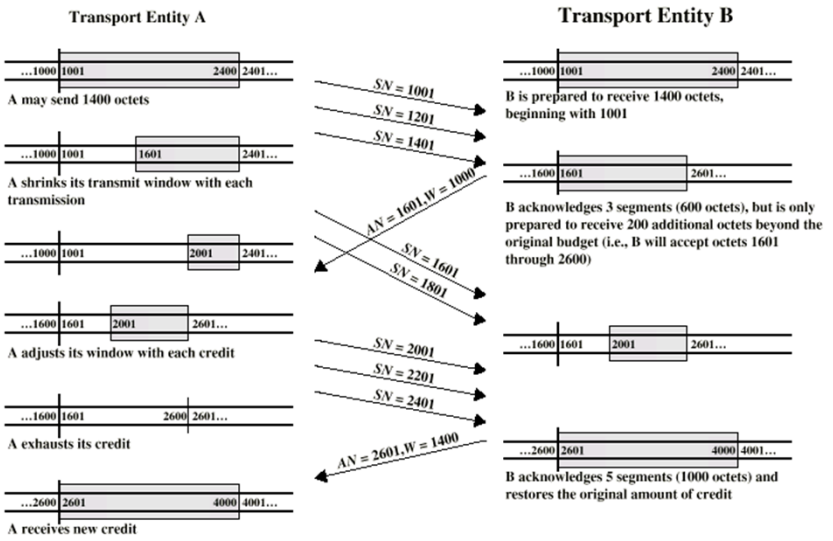



**Figure 20.1**

# Sending and Receiving Perspective

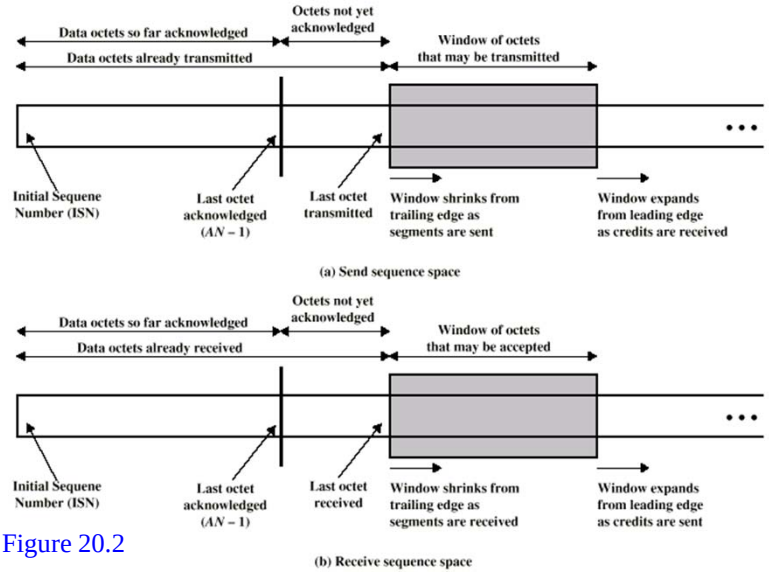



Figure 20.2



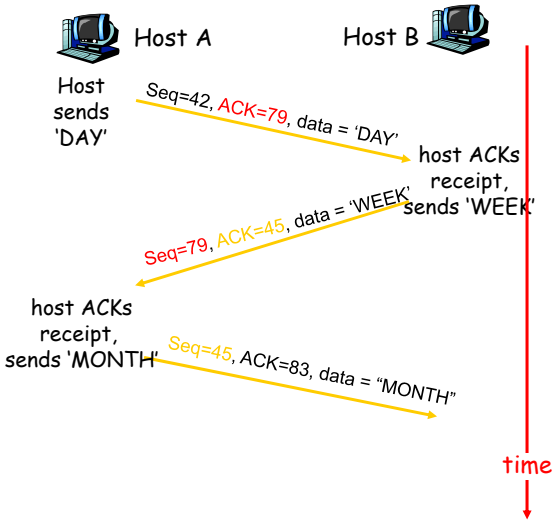
# TCP seq. #'s and ACKs

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

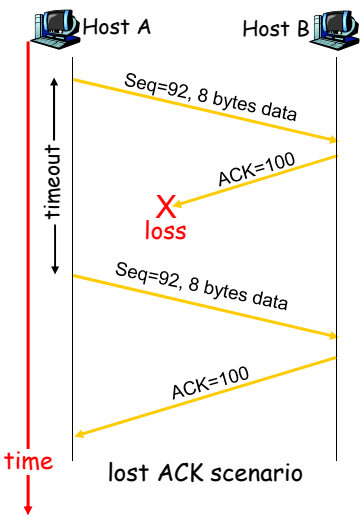


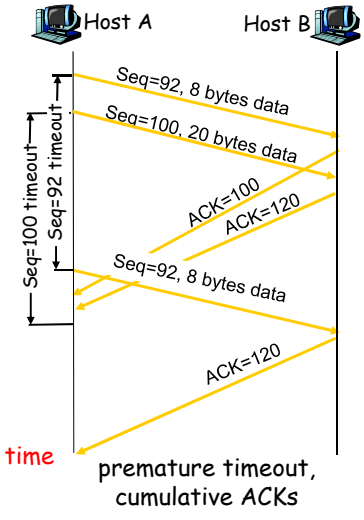
## Retransmission & Duplication Detection

- Receiver must acknowledge successful receipt
- Segment may be damaged in transit or segment fails to arrive
- Transmitter does not know of failure
- Time out waiting for ACK triggers re-transmission (at sender)
  - Fixed timer
  - Adaptive schemas
- Use cumulative acknowledgement compensates for lost ACKs.
- If ACK lost, segment is re-transmitted
- Receiver must recognize duplicates
  - Receiver assumes ACK lost and ACK is duplicated
  - Sender must not get confused with multiple ACKs
- Sequence number space large enough to not cycle within maximum life of segment → Octets numbered modulo $2^{32}$



## TCP: retransmission scenarios




d. xuan

12

## Basic 3-Way Handshake Connection Sync.

| | TCP A | | | TCP B |
|---|---|---|---|---|
| 1. | CLOSED | | | LISTEN |
| 2. | SYN-SENT | → | <SEQ=100><CTL=SYN> → | SYN-RECEIVED |
| 3. | ESTABLISHED | ← | <SEQ=300><ACK=101><CTL=SYN,ACK> ← | SYN-RECEIVED |
| 4. | ESTABLISHED | → | <SEQ=101><ACK=301><CTL=ACK> → | ESTABLISHED |
| 5. | ESTABLISHED | → | <SEQ=101><ACK=301><CTL=ACK><DATA> → | ESTABLISHED |

In line 2, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100.



## Simultaneous Connection Synchronization

| | TCP A | | | TCP B |
|---|---|---|---|---|
| 1. | CLOSED | | | CLOSED |
| 2. | SYN-SENT | → | <SEQ=100><CTL=SYN> ... | |
| 3. | SYN-RECEIVED | ← | <SEQ=300><CTL=SYN> ← | SYN-SENT |
| 4. | | ... | <SEQ=100><CTL=SYN> → | SYN-RECEIVED |
| 5. | SYN-RECEIVED | → | <SEQ=100><ACK=301><CTL=SYN,ACK> ... | |
| 6. | ESTABLISHED | ← | <SEQ=300><ACK=101><CTL=SYN,ACK> ← | SYN-RECEIVED |
| 7. | | ... | <SEQ=101><ACK=301><CTL=ACK> → | ESTABLISHED |

Simultaneous initiation is only slightly more complex, as is shown above. Each TCP cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED

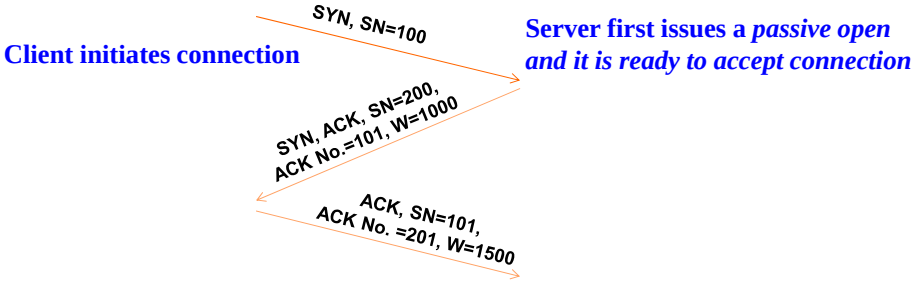
## Recovery from Old Duplicate SYN

| | TCP A | | | TCP B |
|---|---|---|---|---|
| 1. | CLOSED | | | LISTEN |
| 2. | SYN-SENT | → | <SEQ=100><CTL=SYN> ... | |
| 3. | (old duplicate) ... | | <SEQ=90><CTL=SYN> | → SYN-RECEIVED |
| 4. | SYN-SENT | ← | <SEQ=300><ACK=91><CTL=SYN,ACK> | ← SYN-RECEIVED |
| 5. | SYN-SENT | → | <SEQ=91><CTL=RST> | → LISTEN |
| 6. | ... | | <SEQ=100><CTL=SYN> | → SYN-RECEIVED |
| 7. | SYN-SENT | ← | <SEQ=400><ACK=101><CTL=SYN,ACK> | ← SYN-RECEIVED |
| 8. | ESTABLISHED | → | <SEQ=101><ACK=401><CTL=ACK> | → ESTABLISHED |

At line 3, an old duplicate SYN arrives at TCP B. TCP B cannot tell that this is an old duplicate, so it responds normally (line 4). TCP A detects that the ACK field is incorrect and returns a RST (reset) with its SEQ field selected to make the segment believable. TCP B, on receiving the RST, returns to the LISTEN state. When the original SYN finally arrives at line 6, the synchronization proceeds normally. If the SYN at line 6 had arrived before the RST, a more complex exchange might have occurred with RST's sent in both directions.



## Connection Establishment: Windows




- Client sends a SYN segment with a sequence number; can't have data.
- Server sends SYN+ACK segment with its sequence number and acknowledges the receipt of SYN segment; because it contains acknowledge it needs also to define its receive window; this segment can't carry data.
- Client sends ACK segment acknowledging the receipt of SYN+ACK segment from server and needs to defines its receive window; this segment may carry data. The example doesn't assume any data.
- What are the windows after this connection has been established?

## Normal Close Sequence

**TCP A** **TCP B**

**1. ESTABLISHED ESTABLISHED**

**2. (Close from user)**

**FIN-WAIT-1 → <SEQ=10000><ACK=900><CTL=FIN,ACK> → CLOSE-WAIT**

**3. FIN-WAIT-2 ← <SEQ=900><ACK=100001><CTL=ACK> ← CLOSE-WAIT**

**4. (Close from user)**

**TIME-WAIT ← <SEQ=30000><ACK=100001><CTL=FIN,ACK> ← LAST-ACK**

**5. TIME-WAIT → <SEQ=10001><ACK=30001><CTL=ACK> → CLOSED**

**6. After 2 MSL CLOSED**

**A local user initiates the close. TCP A constructs a FIN segment and places it on the outgoing segment queue. No further SENDs from the user will be accepted by the TCP, and it enters the FIN-WAIT-1 state. RECEIVEs are allowed in this state. All segments preceding and including FIN will be retransmitted until acknowledged. When the other TCP has both acknowledged the FIN and sent a FIN of its own, the first TCP can ACK this FIN. Note that a TCP receiving a FIN will ACK but not send its own FIN until its user has CLOSED the connection also.**



## Simultaneous Close Sequence

**TCP A** **TCP B**

**1. ESTABLISHED ESTABLISHED**

**2. (Close from user)**

**FIN-WAIT-1 → <SEQ=100><ACK=300><CTL=FIN,ACK> ... (Close from user)**

**← <SEQ=300><ACK=100><CTL=FIN,ACK> ← FIN-WAIT-1**

**... <SEQ=100><ACK=300><CTL=FIN,ACK> →**

**3. CLOSING → <SEQ=101><ACK=301><CTL=ACK> ...**

**<-- <SEQ=301><ACK=101><CTL=ACK> ← CLOSING**

**... <SEQ=101><ACK=301><CTL=ACK> →**

**4. TIME-WAIT TIME-WAIT**

**(2 MSL) (2 MSL)**

**CLOSED CLOSED**

A simultaneous CLOSE by users at both ends of a connection causes FIN segments to be exchanged. When all segments preceding the FINs have been processed and acknowledged, each TCP can ACK the FIN it has received. Both will, upon receiving these ACKs, delete the connection.

# 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

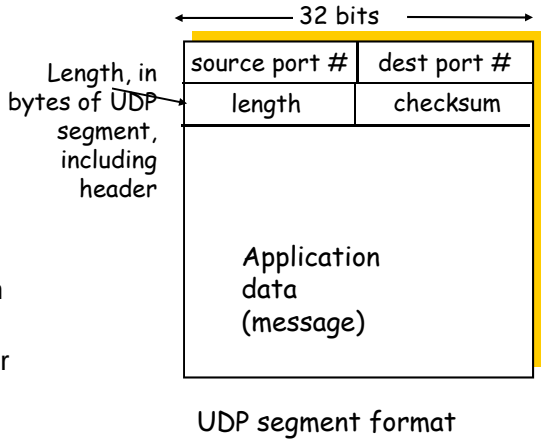
Why is there a UDP?

- 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: User Datagram (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

## Example: TCP/IP in Action

- Let us consider Unix command: ftp bsdi.com, by which a user wants to start some file transfer with the host named bsdi.com.

1. The application (ftp client) calls an appropriate function to convert a host name (bsdi.com) into 32-bit IP address. This function is called resolver, and conversion is done using DNS (Domain Name System).
2. The FTP client now asks its TCP to establish connection with that IP address and (usually well known) port number.
3. TCP sends a connection request segment (SYN) to remote host by sending IP datagram to its IP address.
4. If the destination host is on an another network, the IP routing function sends the IP datagram (containing SYN) to a locally attached next-hop router, in a MAC frame with the router's MAC address as its destination address. All subsequent IP datagrams for this TCP connection are send and received from this router.



## Example: TCP/IP in Action (continued)

5. If the destination host is on the same LAN, the sending host must find its MAC address. It is ARP's (Address Resolution Protocol) function to perform this task.
6. ARP sends an MAC frame caled an ARP request by broadcast. The ARP request contains the IP address of the destination host and it is the request "if you are the owner of this IP address, please respond to me with your MAC address".
7. The destination host's ARP layer receives this broadcast, recognizes that the sender is asking an ARP replay. This replay contains the IP address and the corresponding MAC address. Note that all other hosts in the LAN ignore the ARP request.
8. The ARP replay is received and the IP datagram that forced ARP request-replay sequence can now be sent.
9. The IP datagram (with SYN) is sent to the destination host.