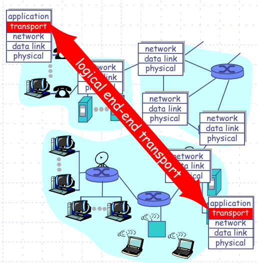
ECE453 – Introduction to Computer Networks

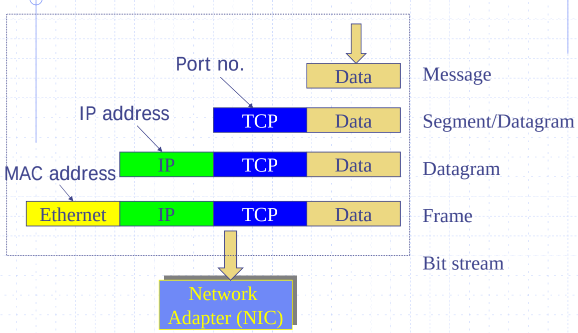
Lecture 14 – Transport Layer (I)

Transport Layer Service Model

- provide *logical communication* between application processes running on different hosts
- transport layer protocols are implemented in the end systems



Layering: Protocol Stack



Transport Layer Services

- Decompose/compose message into/from 4-PDU (TPDU)
- Demultiplexing/multiplexing
- Connection-oriented Service (TCP)
  - Reliable data transfer (RDT)
  - Flow control
  - Congestion control
  - TCP connection management
- Connectionless Service (UDP)
  - checksum

Source Port		Destination Port	
Sequence Number			
Acknowledge Number			
HLEN	Reserved	Code Bits	Window
Checksum		Urgent Pointer	
Options (if any)			Padding
Data			
…			

Source Port	Destination Port
Length	Checksum
Data	

Transport Services (1)

- Decompose/compose message into/from 4-PDU (Protocol Data Unit)
  - breaking application message into smaller chunks
  - Add transport-layer header to each chunk
  - 4-PDU for TCP is called **segment**
  - 4-PDU for UDP is called **datagram**

Transport Service (2)

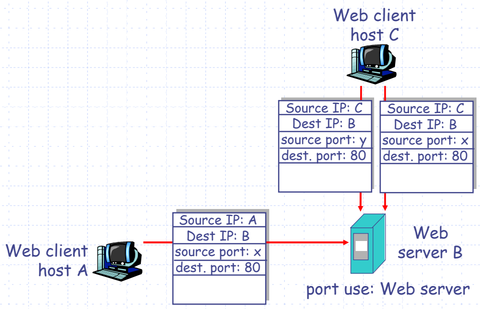
- Demultiplexing/multiplexing
  - IP only delivers data between end systems identified with unique IP address
  - IP does not deliver data between the application processes
  - Demultiplexing: delivering the data in a transport-layer segment to the correct application process
  - Multiplexing: gathering data at the source host from different application processes, enveloping data with header information to create segments and passing the segments to the network layer

Transport Service (3)

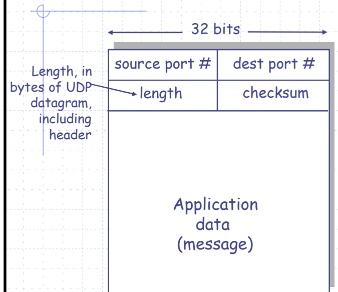
- Connection-oriented Service
  - TCP
  - Reliable data transfer
  - TCP flow control
  - TCP congestion control
  - TCP connection management
- Connectionless Service
  - UDP

Through socket programming

Multiplexing/demultiplexing: Examples



UDP (User Datagram Protocol)



UDP datagram format

- Bare bone transport protocol
- Best-effort service
- Connectionless
- RFC 768
- Eg. DNS
- Pros
  - No connection delay
  - No connection state
  - Small packet header overhead (8 bytes vs. TCP's 20 bytes)
  - Unregulated send rate

Applications Using UDP

- Multicast
- Real-time applications
- Multimedia applications
- Tolerate a small fraction of packet loss

TCP Segment Structure

0 4 16 24 31

<table>
<tr><td colspan="3">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="5">Sequence Number</td></tr>
<tr><td colspan="5">Acknowledge Number</td></tr>
<tr><td>HLEN</td><td>Reserved</td><td>Code Bits</td><td colspan="2">Window</td></tr>
<tr><td colspan="3">Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td colspan="4">Options (if any)</td><td>Padding</td></tr>
<tr><td colspan="5">Data</td></tr>
<tr><td colspan="5">…</td></tr>
</table>

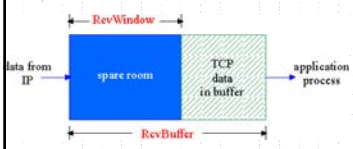
TCP Segment Structure

0 4 16 24 31

<table>
<tr><td colspan="3">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="5">Sequence Number</td></tr>
<tr><td colspan="5">Acknowledge Number</td></tr>
<tr><td>HLEN</td><td>Reserved</td><td>Code Bits</td><td colspan="2">Window</td></tr>
<tr><td colspan="3">Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td colspan="4">Options (if any)</td><td>Padding</td></tr>
<tr><td colspan="5">Data</td></tr>
<tr><td colspan="5">…</td></tr>
</table>

TCP Flow Control

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



receiver buffering

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

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

Congestion Control Problem

- When intermediate machines (routers) become overloaded, the condition is called congestion, the mechanisms to solve the problem are called congestion control mechanism
- Difference from flow control?

Congestion Control Approaches

- End-end congestion control
  - Network layer provides no explicit support to the transport layer for congestion-control purposes
- Network-assisted congestion control
  - Routers provide explicit feedback to the sender regarding the congestion state in the network

Congestion Window

- At any time, TCP window size:
  LastByteSent-LastByteAcked <=
  Allowed window = min(receiver advertisement, congestion window)

Response To Congestion

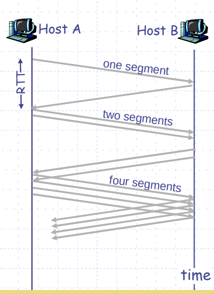
- Two recommended techniques:
  - Slow start
  - Congestion avoidance
    - Additive increase
    - Multiplicative techniques

TCP Slowstart

Slowstart algorithm

```
initialize: Congwin = 1
for (each segment ACKed)
      Congwin++
until (loss event OR
      CongWin > threshold)
```

- exponential increase (per RTT) in window size
- loss event: timeout and/or three duplicate ACKs

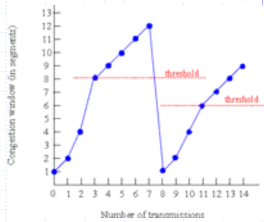


How many RTT is needed before TCP can send N segments?

TCP Congestion Avoidance

Congestion avoidance

```
/* slowstart is over       */
/* Congwin > threshold */
Until (loss event) {
  every w segments ACKed:
      Congwin++
  }
threshold = Congwin/2
Congwin = 1
perform slowstart
```



AIMD – TCP Congestion Avoidance

- AIMD: additive increase, multiplicative decrease
  - increase window by 1 per RTT
  - decrease threshold by factor of 2 on loss event

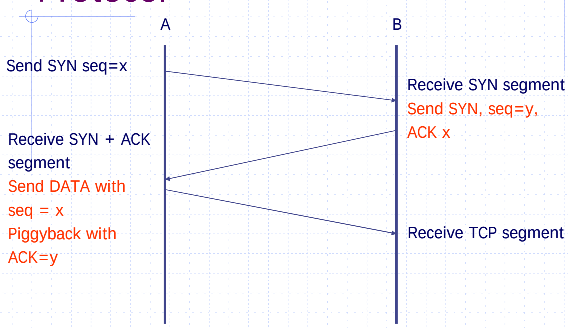
TCP Timeout and Retransmission

- Adaptive retransmission algorithm
  - Next_RTT = (α * last_estimated_RTT) + ($1 - \alpha$) * newly_collected_RTT_sample, $0 < \alpha < 1$
  - Typically, $\alpha = 7/8$

TCP Connection Management

- Establishing a connection
- Initial Sequence Number (ISN)
- Closing a TCP connection
- Connection Reset
- TCP state machine

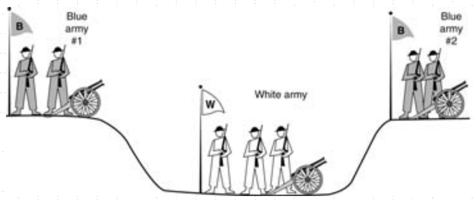
Three-way Handshake Protocol



ISN: Initial Sequence Number

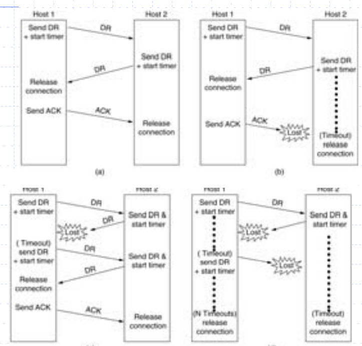
- Two tasks have been accomplished by 3-way handshaking
  - Both sides are ready to transfer data
  - Both sides agree on ISN, ISN is chosen randomly
- Action taken in the three messages
  - A node chooses *x*, B node ACKs *x+1*, which specifies that B will expect octet from *x+1*
  - B node piggybacks his chosen ISN, *y*, on second message, with SYN set
  - A node ACKs this *y* by *y+1*

TCP: Closing Connection

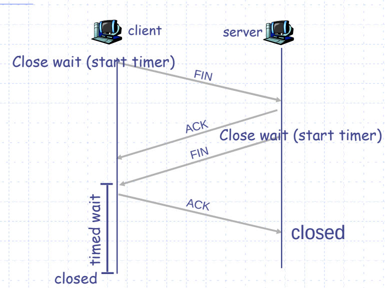


The Two-Army Problem

Connection Release (4 Scenarios)



TCP: Close Connection



TCP: Connection Reset

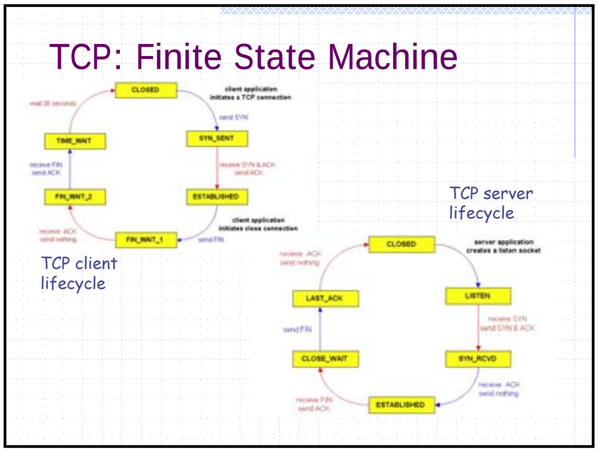
- Normally, TCP closes connection, analogous to closing a file
- Under abnormal conditions, an RST bit indicates immediate abort, resources such as buffers are released

TCP Segment Structure

<table>
<tr><td>0</td><td>4</td><td>16</td><td>24</td><td>31</td></tr>
<tr><td colspan="3">Source Port</td><td colspan="2">Destination Port</td></tr>
<tr><td colspan="5">Sequence Number</td></tr>
<tr><td colspan="5">Acknowledge Number</td></tr>
<tr><td>HLEN</td><td>Reserved</td><td>Code Bits</td><td colspan="2">Window</td></tr>
<tr><td colspan="3">Checksum</td><td colspan="2">Urgent Pointer</td></tr>
<tr><td colspan="4">Options (if any)</td><td>Padding</td></tr>
<tr><td colspan="5">Data</td></tr>
<tr><td colspan="5">…</td></tr>
</table>

TCP Code Bits

Bit (left to right)	Meaning if bit set to 1
URG	Urgent pointer field is valid
ACK	Acknowledgement field is valid
PSH	The segment requires a push
RST	Reset the connection
SYN	Synchronize sequence number
FIN	Sender has reached end of its byte stream


TCP: Finite State Machine
CLOSED
client application
initiates a TCP connection
send SYN
SYN_SENT
receive SYN & ACK
send ACK
ESTABLISHED
client application
initiates close connection
send FIN
FIN_WAIT_1
receive ACK
send nothing
FIN_WAIT_2
receive FIN
send ACK
TIME_WAIT
wait 30 seconds
TCP client
lifecycle
TCP server
lifecycle
CLOSED
server application
creates a listen socket
LISTEN
receive SYN
send SYN & ACK
SYN_RCVD
receive ACK
send nothing
ESTABLISHED
receive FIN
send ACK
CLOSE_WAIT
send FIN
LAST_ACK
receive ACK
send nothing