Lecture 23: TCP

Prof. Alan Mislove (amislove@ccs.neu.edu)

Slides used with permissions from Edward W. Knightly, T. S. Eugene Ng, Ion Stoica, Hui Zhang
Transport Layer in Internet

- Purpose 1: (De)multiplexing of data streams to different application processes
- Purpose 2: Provide value-added services that many applications want
  - Recall network layer in Internet provides a “Best-effort” service only, transport layer can add value to that
    - Application may want reliability, etc
  - No need to reinvent the wheel each time you write a new application
Transport Protocols Concern only End Hosts, not Routers

• Lowest level end-to-end protocol.
  – Header generated by sender is interpreted only by the destination
  – Routers view transport header as part of the payload

• Adds functionality to the best-effort packet delivery IP service.
  – Make up for the shortcomings of the core network
(Possible) Transport Protocol Functions
(Possible) Transport Protocol Functions

- Multiplexing/demultiplexing for multiple applications.
  - Port abstraction
(Possible) Transport Protocol Functions

• Multiplexing/demultiplexing for multiple applications.
  – Port abstraction
• Connection establishment.
  – Logical end-to-end connection
  – Connection state to optimize performance
(Possible) Transport Protocol Functions

• Multiplexing/demultiplexing for multiple applications.
  – Port abstraction

• Connection establishment.
  – Logical end-to-end connection
  – Connection state to optimize performance

• Error control.
  – Hide unreliability of the network layer from applications
  – Many types of errors: corruption, loss, duplication, reordering.
(Possible) Transport Protocol Functions

- Multiplexing/demultiplexing for multiple applications.
  - Port abstraction
- Connection establishment.
  - Logical end-to-end connection
  - Connection state to optimize performance
- Error control.
  - Hide unreliability of the network layer from applications
  - Many types of errors: corruption, loss, duplication, reordering.
- End-to-end flow control.
  - Avoid flooding the receiver
(Possible) Transport Protocol Functions

• Multiplexing/demultiplexing for multiple applications.
  – Port abstraction

• Connection establishment.
  – Logical end-to-end connection
  – Connection state to optimize performance

• Error control.
  – Hide unreliability of the network layer from applications
  – Many types of errors: corruption, loss, duplication, reordering.

• End-to-end flow control.
  – Avoid flooding the receiver

• Congestion control.
  – Avoid flooding the network
(Possible) Transport Protocol Functions

- Multiplexing/demultiplexing for multiple applications.
  - Port abstraction
- Connection establishment.
  - Logical end-to-end connection
  - Connection state to optimize performance
- Error control.
  - Hide unreliability of the network layer from applications
  - Many types of errors: corruption, loss, duplication, reordering.
- End-to-end flow control.
  - Avoid flooding the receiver
- Congestion control.
  - Avoid flooding the network
- More…. 
User Datagram Protocol (UDP)

- Connectionless datagram
  - Socket: SOCK_DGRAM
- Port number used for (de)multiplexing
  - Port numbers = connection/application endpoint
- Adds end-to-end reliability through optional checksum
  - Protects against data corruption errors between source and destination (links, switches/routers, bus)
  - Does not protect against packet loss, duplication or reordering
Using UDP

• Custom protocols/applications can be implemented on top of UDP
  – use the port addressing provided by UDP
  – implement own reliability, flow control, ordering, congestion control as it sees fit

• Examples:
  – remote procedure call
  – Multimedia streaming (real time protocol)
  – distributed computing communication libraries
# Transmission Control Protocol (TCP)

<table>
<thead>
<tr>
<th>Field</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>0</td>
</tr>
<tr>
<td>Dest. Port</td>
<td>16</td>
</tr>
<tr>
<td>Sequence Number</td>
<td>32</td>
</tr>
<tr>
<td>Acknowledgment Number</td>
<td></td>
</tr>
<tr>
<td>HL/Flags</td>
<td></td>
</tr>
<tr>
<td>Advertised Win.</td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
</tr>
<tr>
<td>Urgent Pointer</td>
<td></td>
</tr>
<tr>
<td>Options...</td>
<td></td>
</tr>
</tbody>
</table>
Transmission Control Protocol (TCP)

- Reliable bidirectional in-order byte stream
  - Socket: SOCK_STREAM

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>0</td>
</tr>
<tr>
<td>Dest. Port</td>
<td>16</td>
</tr>
<tr>
<td>Sequence Number</td>
<td>32</td>
</tr>
<tr>
<td>Acknowledgment Number</td>
<td></td>
</tr>
<tr>
<td>HL/Flags</td>
<td></td>
</tr>
<tr>
<td>Advertised Win.</td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td></td>
</tr>
<tr>
<td>Urgent Pointer</td>
<td></td>
</tr>
<tr>
<td>Options..</td>
<td></td>
</tr>
</tbody>
</table>
Transmission Control Protocol (TCP)

- Reliable bidirectional in-order byte stream
  - Socket: SOCK_STREAM
- Connections established & torn down

<table>
<thead>
<tr>
<th>0</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Dest. Port</td>
<td></td>
</tr>
<tr>
<td>Sequence Number</td>
<td>Acknowledgment Number</td>
<td></td>
</tr>
<tr>
<td>HL/Flags</td>
<td>Advertised Win.</td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent Pointer</td>
<td></td>
</tr>
<tr>
<td>Options..</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Transmission Control Protocol (TCP)

- Reliable bidirectional in-order byte stream
  - Socket: SOCK_STREAM
- Connections established & torn down
- Multiplexing/ demultiplexing
  - Ports at both ends
Transmission Control Protocol (TCP)

- Reliable bidirectional in-order byte stream
  - Socket: SOCK_STREAM
- Connections established & torn down
- Multiplexing/ demultiplexing
  - Ports at both ends
- Error control
  - Users see correct, ordered byte sequences

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Dest. Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment Number</td>
<td></td>
</tr>
<tr>
<td>HL/Flags</td>
<td>Advertised Win.</td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>Options..</td>
<td></td>
</tr>
</tbody>
</table>
Transmission Control Protocol (TCP)

- Reliable bidirectional in-order byte stream
  - Socket: SOCK_STREAM
- Connections established & torn down
- Multiplexing/ demultiplexing
  - Ports at both ends
- Error control
  - Users see correct, ordered byte sequences
- End-end flow control
  - Avoid overwhelming machines at each end

Source Port
0
16
32
Dest. Port
Sequence Number
Acknowledgment Number
HL/Flags
Advertised Win.
Checksum
Urgent Pointer
Options..
Transmission Control Protocol (TCP)

• Reliable bidirectional in-order byte stream
  – Socket: SOCK_STREAM
• Connections established & torn down
• Multiplexing/ demultiplexing
  – Ports at both ends
• Error control
  – Users see correct, ordered byte sequences
• End-end flow control
  – Avoid overwhelming machines at each end
• Congestion avoidance
  – Avoid creating traffic jams within network
High Level TCP Features

• Sliding window protocol
  – Use sequence numbers

• Bi-directional
  – Each host can be a receiver and a sender simultaneously
    – For clarity, we will usually discuss only one direction
Connection Setup

• Why need connection setup?

• Mainly to agree on starting sequence numbers
  – Starting sequence number is randomly chosen
  – Reason, to reduce the chance that sequence numbers of old and new connections from overlapping
Important TCP Flags

• SYN: Synchronize
  – Used when setting up connection

• FIN: Finish
  – Used when tearing down connection

• ACK
  – Acknowledging received data
Establishing Connection

- **Three-Way Handshake**
  - Each side notifies other of starting sequence number it will use for sending
  - Each side acknowledges other’s sequence number
    - SYN-ACK: Acknowledge sequence number + 1
    - Can combine second SYN with first ACK
TCP State Diagram: Connection Setup

- **CLOSED**
  - passive OPEN
  - active OPEN
  - CLOSED
  - CLOSE

- **LISTEN**
  - SEND
  - delete TCB

- **SYN RCVD**
  - rcv SYN
  - snd SYN ACK

- **SYN SENT**
  - Rcv SYN, ACK
  - Snd ACK

- **ESTAB**
  - rcv ACK of SYN
  - snd ACK

- **Send FIN**

- **Snd SYN**
TCP State Diagram: Connection Setup

CLOSED

SYN RCVD

rcv SYN
snd SYN ACK

rcv ACK of SYN

CLOSE
Send FIN

LISTEN

SYN SENT

rcv SYN
snd SYN

SEND
snd SYN

Rcv SYN, ACK
Snd ACK

ESTAB

CLOSE
delete TCB

active OPEN
Snd SYN

Client
Tearing Down Connection

• Either Side Can Initiate Tear Down
  – Send FIN signal
  – “I’m not going to send any more data”

• Other Side Can Continue Sending Data
  – Half open connection
  – Must continue to acknowledge

• Acknowledging FIN
  – Acknowledge last sequence number + 1
State Diagram: Connection Tear-down

- **CLOSE**
  - send FIN

- **FIN WAIT-1**
  - send FIN
  - rcv FIN
  - snd ACK
  - rcv FIN+ACK
  - snd ACK
  - rcv FIN
  - snd ACK

- **FIN WAIT-2**
  - ACK
  - snd ACK
  - rcv ACK of FIN

- **ESTAB**
  - rcv FIN
  - snd ACK

- **CLOSING**
  - rcv ACK of FIN

- **TIME WAIT**
  - rcv ACK of FIN
  - Timeout=2 MSL

- **CLOSE WAIT**
  - CLOSE
  - snd FIN

- **LAST-ACK**
  - rcv ACK of FIN

- **CLOSED**
State Diagram: Connection Tear-down

- **CLOSE**: send FIN
- **ESTAB**: rcv FIN
- **Active Close**: send ACK
- **TIME WAIT**: rcv FIN
- **FIN WAIT-2**: rcv FIN
- **FIN WAIT-1**: snd ACK
- **CLOSE WAIT**: snd FIN
- **CLOSING**: rcv FIN
- **LAST-ACK**: rcv FIN+ACK
- **CLOSED**: rcv ACK of FIN
- **Timeout=2 MSL**:
State Diagram: Connection Tear-down

**Active Close**

- FIN WAIT-1
  - FIN
    - Send FIN
  - ACK
    - Send ACK
  - FIN WAIT-2
    - FIN
      - Send FIN
    - ACK
      - Send ACK

**Passive Close**

- LAST-ACK
  - CLOSE
    - Send FIN
  - rcv ACK of FIN
    - Send ACK
  - CLOSED
    - Timeout = 2 MSL

**Estab**

- CLOSE
  - Send FIN
  - rcv FIN
    - Send ACK
  - TIME WAIT
    - rcv ACK of FIN
  - CLOSED
    - Timeout = 2 MSL
Sequence Number Space

- Each byte in byte stream is numbered.
  - 32 bit value
  - Wraps around
  - Initial values selected at start up time
- TCP breaks up the byte stream in packets ("segments")
  - Packet size is limited to the Maximum Segment Size
  - Set to prevent packet fragmentation
- Each segment has a sequence number.
  - Indicates where it fits in the byte stream
Sequence Numbers

• 32 Bits, Unsigned

• Why So Big?
  – For sliding window, must have
    \[ |\text{Sequence Space}| \geq 2^2 \times |\text{Sending Window}| \]
    • \(2^{32} \gg 2 \times 2^{16}\). No problem
  – Also, want to guard against stray packets
    • With IP, assume packets have maximum segment lifetime (MSL) of 120s
      – i.e. can linger in network for upto 120s
    • Sequence number would wrap around in this time at 286Mbps
Error Control
Error Control

• Checksum (mostly) guarantees end-end data integrity.
Error Control

• Checksum (mostly) guarantees end-end data integrity.
• Sequence numbers detect packet sequencing problems:
  – Duplicate: ignore
  – Reordered: reorder or drop
  – Lost: retransmit
Error Control

• Checksum (mostly) guarantees end-end data integrity.
• Sequence numbers detect packet sequencing problems:
  – Duplicate: ignore
  – Reordered: reorder or drop
  – Lost: retransmit
• Lost segments detected by sender.
  – Use time out to detect lack of acknowledgment
  – Need estimate of the roundtrip time to set timeout
Error Control

- Checksum (mostly) guarantees end-end data integrity.
- Sequence numbers detect packet sequencing problems:
  - Duplicate: ignore
  - Reordered: reorder or drop
  - Lost: retransmit
- Lost segments detected by sender.
  - Use timeout to detect lack of acknowledgment
  - Need estimate of the roundtrip time to set timeout
- Retransmission requires that sender keep copy of the data.
  - Copy is discarded when ack is received
Bidirectional Communication

• Each Side of Connection can Send and Receive

• What this Means
  – Maintain different sequence numbers for each direction
  – Single segment can contain new data for one direction, plus acknowledgement for other
  • But some contain only data & others only acknowledgement
TCP Flow Control

• Sliding window protocol
  – For window size $n$, can send up to $n$ bytes without receiving an acknowledgement
  – When the data are acknowledged then the window slides forward

• Window size determines
  – How much unacknowledged data can the sender sends

• But there is more detail
Complication!

- TCP receiver can delete acknowledged data only after the data has been delivered to the application
- So, depending on how fast the application is reading the data, the receiver’s window size may change!!!
Solution

• Receiver tells sender what is the current window size in every packet it transmits to the sender
• Sender uses this current window size instead of a fixed value

• Window size (also called Advertised window) is continuously changing
• Can go to zero!
  – Sender not allowed to send anything!
Window Flow Control: Receive Side

Receive buffer

[Green] Acked but not delivered to user

[Yellow] Not yet acked

Window
Window Flow Control: Send Side

- Sent and acked
- Sent but not acked
- Not yet sent

Next to be sent

Window
Window Flow Control: Send Side

- Sent and acked
- Sent but not acked
- Not yet sent

Next to be sent

Must retain for possible retransmission
Window Flow Control: Send Side

Packet Sent

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Dest. Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>HL/Flags</td>
<td>Window</td>
</tr>
<tr>
<td>D. Checksum</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>Options..</td>
<td></td>
</tr>
</tbody>
</table>

Packet Received

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Dest. Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number</td>
<td>Acknowledgment</td>
</tr>
<tr>
<td>HL/Flags</td>
<td>Window</td>
</tr>
<tr>
<td>D. Checksum</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>Options..</td>
<td></td>
</tr>
</tbody>
</table>

App write

acknowledged  sent  to be sent  outside window
Ongoing Communication

• Bidirectional Communication
  – Each side acts as sender & receiver
  – Every message contains acknowledgement of received sequence
    • Even if no new data have been received
  – Every message advertises window size
    • Size of its receiving window
  – Every message contains sent sequence number
    • Even if no new data being sent

• When Does Sender Actually Send Message?
  – When sending buffer contains at least max. segment size (- header sizes) bytes
  – When application tells it
    • Set PUSH flag for last segment sent
  – When timer expires
What is Congestion?

• The load placed on the network is higher than the capacity of the network
  – Not surprising: independent senders place load on network
• Results in packet loss: routers have no choice
  – Can only buffer finite amount of data
  – End-to-end protocol will typically react, e.g. TCP
Why is Congestion Bad?

- Wasted bandwidth: retransmission of dropped packets
- Poor user service: unpredictable delay, low user goodput
- Increased load can even result in lower network goodput
  - Switched nets: packet losses create lots of retransmissions
  - Broadcast Ethernet: high demand -> many collisions

Goodput vs Load graph showing "congestion collapse"
Sending Rate of Sliding Window Protocol
Sending Rate of Sliding Window Protocol

- Suppose A uses a sliding window protocol to transmit a large data file to B
Sending Rate of Sliding Window Protocol

• Suppose A uses a sliding window protocol to transmit a large data file to B
• Window size = 64KB
Sending Rate of Sliding Window Protocol

• Suppose A uses a sliding window protocol to transmit a large data file to B
• Window size = 64KB
• Network round-trip delay is 1 second
Sending Rate of Sliding Window Protocol

- Suppose A uses a sliding window protocol to transmit a large data file to B
- Window size = 64KB
- Network round-trip delay is 1 second
Sending Rate of Sliding Window Protocol

- Suppose A uses a sliding window protocol to transmit a large data file to B
- Window size = 64KB
- Network round-trip delay is 1 second

- What’s the expected sending rate?
Sending Rate of Sliding Window Protocol

• Suppose A uses a sliding window protocol to transmit a large data file to B
• Window size = 64KB
• Network round-trip delay is 1 second

• What’s the expected sending rate?
• 64KB/second
Sending Rate of Sliding Window Protocol

• Suppose A uses a sliding window protocol to transmit a large data file to B
• Window size = 64KB
• Network round-trip delay is 1 second

• What’s the expected sending rate?
• 64KB/second
Sending Rate of Sliding Window Protocol

• Suppose A uses a sliding window protocol to transmit a large data file to B
• Window size = 64KB
• Network round-trip delay is 1 second

• What’s the expected sending rate?
• 64KB/second

• What if a network link is only 64KB/second but there are 1000 people who are transferring files over that link using the sliding window protocol?
Sending Rate of Sliding Window Protocol

• Suppose A uses a sliding window protocol to transmit a large data file to B
• Window size = 64KB
• Network round-trip delay is 1 second

• What’s the expected sending rate?
• 64KB/second

• What if a network link is only 64KB/second but there are 1000 people who are transferring files over that link using the sliding window protocol?
• Packet losses, timeouts, retransmissions, more packet losses… nothing useful gets through, congestion collapse!
TCP Window Flow Control

Packet Sent

- Source Port
- Dest. Port
- Sequence Number
- Acknowledgment
- HL/Flags
- Window
- D. Checksum
- Urgent Pointer
- Options...

Packet Received

- Source Port
- Dest. Port
- Sequence Number
- Acknowledgment
- HL/Flags
- Window
- D. Checksum
- Urgent Pointer
- Options...

App write

acknowledged → sent → to be sent → outside window
TCP Flow Control Alone Is Not Enough

• We have talked about how TCP’s advertised window is used for flow control
  – To keep sender sending faster than the receiver can handle
• If the receiver is sufficiently fast, then the advertised window will be maximized at all time
• But clearly, this will lead to congestion collapse as the previous example if there are too many senders or network is too slow

• Key 1: Window size determines sending rate
• Key 2: Window size must be dynamically adjusted to prevent congestion collapse
How Fast to Send? What’s at Stake?

• Send too slow: link sits idle
  – wastes time

• Send too fast: link is kept busy but....
  – queue builds up in router buffer (delay)
  – overflow buffers in routers (loss)
  – Many retransmissions, many losses
  – Network goodput goes down

![Diagram of Goodput vs. Load with a safe operating point]