Later today: Look for message about IP grading
  – Meeting slots first week after break (and during break)
TCP: Draft of assignment out today
  – Read it over before break, start when we get back
Summer/UTA hiring: Expect a message from me today/tomorrow
Today

Light overview of the transport layer and TCP
– Why we need TCP
– What components are involved
– What you will do in the project
• Transport protocols sit on top of network layer
• Problem solved: communication among processes
  – Application-level multiplexing (“ports”)
  – Error detection, reliability, etc.
From Lec 2: OSI Reference Model

One or more nodes within the network

Application Protocol

Transport Protocol

Network Protocol

Link-Layer Protocol
User Datagram Protocol

- Unreliable datagram service
- Adds multiplexing (via ports) and nothing else
- Checksum is pretty useless
Next Problem: Reliability

We talked briefly about link-layer reliability:

<table>
<thead>
<tr>
<th>Problem</th>
<th>Mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dropped Packets</td>
<td>Acknowledgments + Timeout</td>
</tr>
<tr>
<td>Duplicate Packets</td>
<td>Sequence Numbers</td>
</tr>
<tr>
<td>Packets out of order</td>
<td>Receiver Window</td>
</tr>
<tr>
<td>Maximizing throughput</td>
<td>Sliding Window (Pipelining)</td>
</tr>
</tbody>
</table>

- Single link: things were easy… 😊
Transport Layer Reliability

• Extra difficulties
  – Multiple hosts
  – Multiple hops
  – Multiple potential paths

• What does this mean?
  – Multiple opportunities for failure
  – Hosts have different resources
  – Varying RTTs
Extra Difficulties (cont.)

• Out of order packets
  – Not only because of drops/retransmissions
  – Can get very old packets (up to 120s), must not get confused

• Unknown resources at other end
  – Must be able to discover receiver buffer: flow control

• Unknown resources in the network
  – Should not overload the network
  – But should use as much as safely possible to maximize throughput
TCP – Transmission Control Protocol

- Service model: “reliable, connection oriented, full duplex ordered byte stream”
- Flow control: If one end stops reading, writes at other eventually stop/fail
- Congestion control: Keeps sender from overloading the network
TCP

• Specification

• Was born coupled with IP, later factored out

• End-to-end protocol
  – Minimal assumptions on the network
  – All mechanisms run on the end points

• What if you had link-layer reliability instead?
Why not provide X on the network layer?

$X = \text{Reliability, security, message ordering...}$

- **Cost**
  - These functionalities are not free: don’t burden those who don’t need them
- **Conflicting**
  - Timeliness and in-order delivery, for example
- **Insufficient**
  - Example: reliability
End-to-end argument

• Functions placed at lower levels of a system may be redundant or of little value
  – They may need to be performed at a higher layer anyway
• But they may be justified for performance reasons
  – Or just because they provide most of what is needed
  – Example: retransmissions
• Takeaway: weigh the costs and benefits at each layer
TCP Header

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|       Source Port       |       Destination Port       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                          Sequence Number                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                          Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                          Window                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                            Data                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                         |U|A|P|R|S|F|                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Offset| Reserved |R|C|S|S|Y|I|                          Window |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                            |G|K|H|T|N|N|                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                          Checksum                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                            Options                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                          Urgent Pointer                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                          Padding                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
|                          data                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
Header Fields

- **Ports**: multiplexing
- **Sequence number**
  - Correspond to bytes, not packets!
- **Acknowledgment Number**
  - Next expected sequence number
- **Window**: willing to receive
  - Lets receiver limit SWS (even to 0) for flow control
- **Data Offset**: # of 4 byte (header + option bytes)
- **Flags, Checksum, Urgent Pointer**
Header Flags

• URG: whether there is urgent data
• ACK: ack no. valid (all but first segment)
• PSH: push data to the application immediately
• RST: reset connection
• SYN: synchronize, establishes connection
• FIN: close connection
Establishing a Connection

- Three-way handshake
  - Two sides agree on respective initial sequence nums
- If no one is listening on port: server sends RST
- If server is overloaded: ignore SYN
- If no SYN-ACK: retry, timeout

Active participant (client)
- SYN, SequenceNum = x
- SYN + ACK, SequenceNum = y
- ACK, Acknowledgment = x + 1

Passive participant (server)
- listen(), accept()
- connect()
- accept() returns
Connection Termination

- FIN bit says no more data to send
  - Caused by close or shutdown
  - Both sides must send FIN to close a connection
- Typical close
Summary of TCP States

- **Passive close:** Can still send!
- **Active close:** Can still receive

Connection Establishment:
- **CLOSED**
- **LISTEN**
- **SYN_RCVD**
- **SYN_SENT**
- **ESTABLISHED**
- **FIN_WAIT_1**
- **FIN_WAIT_2**
- **CLOSING**
- **TIME_WAIT**
- **CLOSE_WAIT**
- **LAST_ACK**
- **CLOSED**

Active open:
- /SYN

Passive open:
- Close
Next class

• Sending data over TCP