CSCI-1680
Transport Layer I

Rodrigo Fonseca

Based partly on lecture notes by David Mazières, Phil Levis, John Jannotti
Today

• **Transport Layer**
  – UDP
  – TCP Intro
    • Connection Establishment
From Lec 2: OSI Reference Model

Application Protocol

Transport Protocol

Link-Layer Protocol

Network Protocol

One or more nodes within the network
Transport Layer

- Transport protocols sit on top of the network layer (IP)
- Can provide:
  - Application-level multiplexing ("ports")
  - Error detection, reliability, etc.

- Problem solved: communication among processes
  - Application-level multiplexing ("ports")
  - Error detection, reliability, etc.
UDP – User Datagram Protocol

- Unreliable, unordered datagram service
- Adds multiplexing, checksum
- End points identified by *ports*
  - Scope is an IP address (interface)
- Checksum aids in error detection
UDP Checksum

- **Uses the same algorithm as the IP checksum**
  - Set Checksum field to 0
  - Sum all 16-bit words, adding any carry bits to the LSB
  - Flip bits to get checksum (except 0xffff->0xffff)
  - To check: sum whole packet, including sum, should get 0xffff

- **How many errors?**
  - Catches any 1-bit error
  - Not all 2-bit errors

- **Optional in IPv4: not checked if value is 0**
Pseudo Header

0 7 8 15 16 23 24 31
+-------------------------------+
| source address               |
+-------------------------------+
| destination address          |
+-------------------------------+
| zero  |protocol| UDP length |
+-------------------------------+

- **UDP Checksum is computer over *pseudo-header* prepended to the UDP header**
  - For IPv4: IP Source, IP Dest, Protocol (=17), plus UDP length

- **What does this give us?**

- **What is a problem with this?**
  - Is UDP a layer on top of IP?
Next Problem: Reliability

- Review: reliability on the link layer

<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>Keeping the pipe full</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

• Need for connection establishment, tear down
  – Analogy: dialing a number versus a direct line

• Varying RTTs
  – Both across connections and *during* a connection
  – Why do they vary? What do they influence?
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
  – Congestion Control (next class)
TCP – Transmission Control Protocol

- **Service model:** “reliable, connection oriented, full duplex ordered byte stream”
  - Endpoints: <IP Address, Port>
- **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
  – We talked about this, don’t always need everything!

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

• Alternative idea:
  – Provide reliability, flow control, etc, link-by-link
  – Does it work?
Not the only options…

<table>
<thead>
<tr>
<th></th>
<th>UDP</th>
<th>TCP</th>
<th>SCTP</th>
<th>DCCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplexing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In-order</td>
<td></td>
<td></td>
<td></td>
<td><strong>optional</strong></td>
</tr>
<tr>
<td>Message</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stream</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flow Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Congestion Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple Streams</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple Paths</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*MPTCP adds multiple streams and multiple paths
This table is not exhaustive!
Why not provide (*) on the network layer?

• **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

* may be security, reliability, ordering guarantees, …
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

• Lesson: weigh the costs and benefits at each layer
  – Also: the end also varies from case to case
TCP Header

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Source Port | Destination Port

Sequence Number

Acknowledgment Number

Data | U|A|P|R|S|F|
Offset | Reserved | R|C|S|S|Y|I|
| | G|K|H|T|N|N|

Window

Checksum

Urgent Pointer

Options

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 numbers
- If no one is listening on port: server sends RST
- If server is overloaded: ignore SYN
- If no SYN-ACK: retry, timeout
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**

  ![Connection Termination Diagram](attachment:image.png)
Summary of TCP States

Unsynchronized

Synchronized

Connection Establishment

Active open/SYN

Close

Send/SYN

Unsynchronized

Synchronized

Connection Establishment

Active close: Can still receive

Passive close: Can still send!

Timeout after two segment lifetimes
TIME_WAIT

• Why do you have to wait for 2MSL in TIME_WAIT?
  – What if last ack is severely delayed, AND
  – Same port pair is immediately reused for a new connection?

• Solution: active closer goes into TIME_WAIT
  – Waits for 2MSL (Maximum Segment Lifetime)

• Can be problematic for active servers
  – OS has too many sockets in TIME_WAIT, can accept less connections
    • Hack: send RST and delete socket, SO_LINGER = 0
  – OS won’t let you re-start server because port in use
    • SO_REUSEADDR lets you rebind
From: The TIME−WAIT state in TCP and Its Effect on Busy Servers, Faber and Touch Infocom 1999
Next class

• Sending data over TCP