Introduction to the Transport Layer

Overview
- Discuss tasks performed by the transport layer
  - Services provided to the application layer
  - Services expected from the network layer
- Multiplexing and demultiplexing
- Connection management
- Error checking - the checksum

Transport services and protocols
- Provide logical communication, a virtual connection
  ...between application processes running on different hosts

This is not a physical path including routers

Transport Layer Tasks
- The transport layer (TCP) provides reliability over an unreliable network
- What can go wrong?
  - .
  - .
  - .
The Actual Transport Layer

- Basic transport layer services:
  - 
  - 
  - 
  - 
  - 

- Services not available:
  - 
  - 
  - 

- Internet transport protocols:
  - UDP: Connectionless transport
  - TCP: Connection-oriented transport

Constructing Transport Segment

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>other header fields</td>
<td></td>
</tr>
</tbody>
</table>

Multiplexing/demultiplexing

- Multiplexer
  - 
  - 

- Demultiplexer
  - 
  - 

Encapsulation

message segment

datagram

Encapsulation

source application transport network link physical

destination application transport network link physical

Multiplexing/demultiplexing

Multiplexer

Demultiplexer
Multiplexing/demultiplexing

Multiplexing at sender:
- handle data from multiple sockets
- add transport header (later used for demultiplexing)

Demultiplexing at receiver:
- use header info to deliver received segments to correct socket

Connectionless demultiplexing

- UDP socket is bound to the local host port #
- recall: when creating datagram to send into a UDP socket, the socket must specify
  - destination IP address
  - destination port #

- when host receives UDP segment:
  1) check destination port # in segment header
  2) direct UDP segment to socket with that port #

IP datagrams with same dest. port #, but different source IP addresses and/or source port numbers will be directed to same socket at destination

UDP: User Datagram Protocol

- UDP is a “best effort” service. Segments may be:
  - lost
  - delivered out of order

SO why is there a UDP?

-
Connection-oriented demux: example

TCP Socket & Segment

- Create sockets with port & IP addr:
  
  ```java
  Socket clientSocket = new Socket("hostname", 6789);
  Socket connectionSocket = welcomeSocket.accept();
  ```

- TCP: Server host has simultaneous TCP sockets, one for each connection:
  - each socket identified by its own 4-tuple

- TCP segment includes data, source & destination port and IP addresses (+ length & checksum)

Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number

- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple

- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

- demux: receiver uses all four values to direct segment to appropriate socket

TCP Connection-oriented demux

- In what situation will all TCP segments with different source IP addr and port #s nonetheless go to the same port?
  -

- Alternatively: Web servers have different sockets for each connected client
  - non-persistent HTTP will have different socket for each request
TCP Connection Management: Set up

Recall: TCP senders and receivers establish a “connection” before exchanging data segments.

Three way handshake:

Step 1: client host sends TCP SYN segment to server
- “SYN” for “synchronize”
- Specifies (random) initial sequence #
- No data is sent

Step 2: server host receives SYN, replies with SYNACK segment
- Server allocates buffers and variables
- Specifies its own, server initial sequence #

Step 3: client receives SYNACK, replies with ACK segment
- Client allocates buffers and variables
- This packet may contain data

TCP Connection Management: Close

Closing a connection:

How many steps?
What are they?
Error Checking: Checksum

**Goal:** detect “errors” (e.g., flipped bits) in transmitted segment

**Sender:**
- treat segment contents as sequence of 16-bit integers
- checksum: 1’s complement of the sum of (16-bit) segment contents
- sender puts checksum value into UDP checksum field

**Receiver:**
- compute checksum of received segment - including the sender’s checksum 16-bit word in the sum
- If receiver’s sum is all ‘1’s then there were no errors (probably)
- If a bit is 0 then the packet has errors

Internet Checksum Example

**Note**
- When adding numbers, a carryout from the most significant bit needs to be added to the result, for 1’s complement

**Example:** add two 16-bit integers

\[
\begin{array}{cccccccccccccccc}
1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

wraparound \[
\begin{array}{cccccccccccccccc}
1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\
\end{array}
\]

sum \[
\begin{array}{cccccccccccccccc}
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

Checksum example

- Is it possible for a 1-bit error to go undetected?
- Is it possible for a 2-bit error to go undetected?

Summary

- Transport layer services
  - Desired services
  - Actual protocol services
  - What can go wrong?
- Multiplexing and demultiplexing
- Error checking - checksum
  - Transport layer provides end-to-end error checking vs. link layer single link error checking