Transport: Basics

G54ACC – IP and Up

Lecture 3

Recap

- UDP connectionless, unreliable
- TCP *connection oriented*, providing *reliability*
- Provides *flow control* and *congestion control*

- Congestion control will be covered in the next lecture

Contents

Contents

- Overview
  - Transport layer
  - Port numbers
- User Datagram Protocol
- Transmission Control Protocol

The Transport Layer

- Process-to-Process communication
  - Cf. Host-to-Host (or interface-to-interface) from IP
  - I.e., Multiplexing within host
- UDP, RFC768
  - Best effort, unordered, datagrams
- TCP, RFC793
  - Reliable, ordered, byte-stream
  - ...with respect to the *process* not the host

Port Numbers

- Provide multiplexing of host’s network connection between processes
  - (src IP, dst IP, src pt, dst pt, proto) 5-tuple uniquely identifies a *flow* in the Internet
    - At a given moment in time anyway
  - Destination port often identifies service
- 16 bit number space
  - [0, 1023]: *well-known*, usually require root privs.
  - [1024, 49151]: *registered*, often out-of-date
  - [49152, 65535]: dynamic/private/*ephemeral*

Contents

- Overview
- User Datagram Protocol
  - Header format
  - Pseudo-headers
- Transmission Control Protocol

User Datagram Protocol

- About the simplest possible transport protocol
  - Provides simple datagrams over IP's packets
  - Adds a *datagram checksum* and *port numbers*
- Checksum gives processes some reliability
  - Covers *pseudo-header* and data
  - If you receive data, it was (probably) intended for you and was (probably) transmitted correctly
- Ports enable multiple processes per host to use IP networking simultaneously

UDP Headers

- UDP Header
  - Port numbers, length
  - Checksum
  - Protects against bit errors
- UDP Pseudo-Header
  - Prefixed to UDP header
  - IP addresses, IP protocol, length, zero checksum
  - Adds protection against misdelivery to checksum

```
 0      7 8     15 16    23 24    31
+--------+--------+--------+--------+
|  Source Port    | Destination Port|
+--------+--------+--------+--------+
|     Length      |    Checksum     |
+--------+--------+--------+--------+
|          data octets              ...
+-----------------+--------+--------+
```

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

Contents

- Overview
- User Datagram Protocol
- Transmission Control Protocol
  - Header format
  - State machine: setup, teardown
  - Priority, precedence
  - Reliability, timeouts
  - Host considerations
  - Flow control

Transmission Control Protocol

- Reliable...
  - It will retransmit data if it gets lost
- ...ordered...
  - Data guaranteed to arrive in the order sent
- ...byte-stream...
  - No need for application to segment data
  - However, it is common to insert delimiters
- **Key problem**: Network is *asynchronous*
  - No timing guarantees on, or indication of, delivery
  - We will use timers to infer loss – but how to set them?

A General Principle

- “TCP implementations will follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.” RFC 793
  - Followed in most of the good Internet protocols
  - Although consider the trade-off between robustness and enforcing upgrade/evolution

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 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|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                            ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Connections: State Machine

- Quite complex!
  - Why? ☺
- Transitions via:
  - (Sockets) API call
  - Packet Rx/Tx
  - Timer expiry (timeout)
- Typically three phases:
  - Connecting
  - Established
  - Closing
- ...but corner-cases exist!
  - E.g., simultaneous open/close

```
                              +---------+ ---------\      active OPEN
                              |  CLOSED |            \    -----------
                              +---------+<---------\   \   create TCB
                                |     ^              \   \  snd SYN
                   passive OPEN |     |   CLOSE        \   \
                   ------------ |     | ----------       \   \
                    create TCB  |     | delete TCB         \   \
                                V     |                      \   \
                              +---------+            CLOSE    |    \
                              |  LISTEN |          ---------- |     |
                              +---------+          delete TCB |     |
                   rcv SYN      |     |     SEND              |     |
                  -----------   |     |    -------            |     V
 +---------+      snd SYN,ACK  /       \   snd SYN          +---------+
 |         |<-----------------           ------------------>|         |
 |   SYN   |                    rcv SYN                     |   SYN   |
 |   RCVD  |<-----------------------------------------------|   SENT  |
 |         |                    snd ACK                     |         |
 |         |------------------           -------------------|         |
 +---------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +---------+
   |           --------------   |     |   -----------
   |                  x         |     |     snd ACK
   |                            V     V
   |  CLOSE                   +---------+
   | -------                  |  ESTAB  |
   | snd FIN                  +---------+
   |                   CLOSE    |     |    rcv FIN
   V                  -------   |     |    -------
 +---------+          snd FIN  /       \   snd ACK          +---------+
 |  FIN    |<-----------------           ------------------>|  CLOSE  |
 | WAIT-1  |------------------                              |   WAIT  |
 +---------+          rcv FIN  \                            +---------+
   | rcv ACK of FIN   -------   |                            CLOSE  |
   | --------------   snd ACK   |                           ------- |
   V        x                   V                           snd FIN V
 +---------+                  +---------+                   +---------+
 |FINWAIT-2|                  | CLOSING |                   | LAST-ACK|
 +---------+                  +---------+                   +---------+
   |                rcv ACK of FIN |                 rcv ACK of FIN |
   |  rcv FIN       -------------- |    Timeout=2MSL -------------- |
   |  -------              x       V    ------------        x       V
    \ snd ACK                 +---------+delete TCB         +---------+
     ------------------------>|TIME WAIT|------------------>| CLOSED  |
                              +---------+                   +---------+
```

Connections: Setup

- First step: setup a connection
  - Repeat: *the network is asynchronous*!
- 3-way handshake:
  - [A] SYN: “I wish to connect using `seqno=x`.”
  - [B] SYN/ACK: “Sure, `ackno=x+1`; likewise with `seqno=y`.”
  - [A] ACK: “Acknowledged, `ackno=y+1`.”
- Minimum required for both parties to agree a connection has been setup
  - Can send data with the final ACK
- Connection is *bidirectional*: data can flow both ways

Connections: Teardown

- Last step: tearing down the connection
  - Politely; can always just RESET
  - *Bidirectional* so teardown both directions
  - Messages can overlap!
- Two-way, but in both directions:
  - [1] FIN: “Stop sending me stuff.”
  - [2] ACK: “Ok.”
  - ...
  - [2] FIN: “Stop sending me stuff.”
  - [1] ACK: “Ok.”

[1] FIN: “Stop sending me stuff.”
[2] FIN/ACK: “Ok; likewise.”
[1] ACK: “Ok.”

- Summary: network asynchrony, bidirectional comms., and local resource management make this tricky

Priority and Precedence

- TCP presents an *ordered bytestream*
  - The stack can hold on to data before Tx
  - Not all data in a stream is equal
  - Mechanisms to deal with these problems
- PuSH
  - Process says to stack “Tx queued data *now*!”
  - In practice “promptly”
  - Exists on the wire, so Rx can notice it too
- URGent
  - Tx indicates “this future data is urgent!”
  - Rx can hurry processing to reach the indicated point

Contents

- Overview
- User Datagram Protocol
- Transmission Control Protocol
  - Header format
  - State machine: setup, teardown
  - Priority, precedence
  - Reliability, timeouts
  - Host considerations
  - Flow control

Reliability

- Basically: retransmit when data lost
- Three problems:
  - How to detect loss?
  - How to avoid loss due to host congestion?
  - How to avoid loss due to network congestion?
- Detect loss via (byte-stream) sequence numbers
  - Sender sends seqno
  - Receiver sends ackno
  - Sender can detect how far receiver is through stream

Timeouts

- How long should we wait to retransmit? RTO(RTT)
- RTT estimation: exponential average
  - $RTT_{sample} = time_{ackRx} - time_{packetTx}$
  - $RTT_{estimated} = \alpha.RTT_{estimated} + (1-\alpha).RTT_{sample}$
  - $\alpha = 7/8$
- RTO computation: standard deviation rather expensive
  - Jacobsen/Karels: mean deviation
    - $difference = RTT_{sample} - RTT_{estimated}$
    - $devation \mathrel{+}= \delta.(|difference| - deviation)$
    - $RTO = \mu.RTT_{estimated} + \phi.deviation$
    - $\delta = ¼, \mu = 1, \phi = 4$
  - Karn/Partridge: retransmits give ambiguous measurements
    - Avoid using retransmitted segments for RTT_{sample}
    - Exponential backoff of RTO: each expiry, RTO *= 2 (max. 60s)

Flow Control

- Rx host needs to put data somewhere
  - It's likely to be busy in an OS scheduling sense
  - Buffer management required for connection
- Rx advertised window (awnd)
  - Advertise unused buffer space in ACK
- Enables Tx to avoid sending so much that Rx will be forced to drop it
  - ...wasting network (and Tx host) resource
  - Avoiding *network* induced loss is *congestion control*

Connections: Hosts

- Managed via the Transmission Control Block
- Contains state used to manage the connection
  - In reality, `struct` is 134 lines in OpenBSD!
  - Adds connection state, timers, buffer queue, congestion management details, retransmit details, cached templates, window scaling, path-MTU discovery, &c.

```
Send Sequence Variables
 SND.UNA - send unacknowledged
 SND.NXT - send next
 SND.WND - send window
 SND.UP  - send urgent pointer
 SND.WL1 - segment sequence number used for
           last window update
 SND.WL2 - segment acknowledgment number used
           for last window update
 ISS     - initial send sequence number

Receive Sequence Variables
 RCV.NXT - receive next
 RCV.WND - receive window
 RCV.UP  - receive urgent pointer
 IRS     - initial receive sequence number
```

Summary

- Transport gives per-process demultiplexing
- UDP: connectionless, unreliable; *vs*.
- TCP: connection-oriented, sequenced, reliable
  - Question: why does IP have header checksum, UDP a pseudo-header checksum, and TCP a data checksum?
- Providing reliability in an asynchronous network is *hard*!
  - If you want it to be a generally usable feature
  - In a wide variety of situations
  - *Congestion control next...*