Lecture 21: Transport Layer

Adam Hawley

April 9, 2019

Contents

1 Introduction

The transport layer provides **end-to-end connectivity** in terms of a **transport protocol** (end-to-end protocol). The underlying network layer usually only provides a **best-effort host-to-host service** (e.g. IP):

- messages are dropped (due to congestion)
- messages are re-ordered
- messages are delivered several times (problem of duplicates)
- messages are limited to some finite size
- messages are delivered after some long delay

Different transport protocols address (some of) these limitations by offering different services:

- Simple (application) demultiplexing service (**User Datagram Protocol**)
- Reliable Byte-Stream Service (**Transmission Control Protocol**)

2 User Datagram Protocol (UDP)

UDP is a simple extension of the host-to-host delivery service to a process-to-process communication service. It lets multiple application processes share the same network. Using an **abstract locator**, a process and a **port** are identified. The network delivers messages to the port and a destination receives messages from the port. UDP uses a 16-bit numbering scheme to identify ports. Processes are identified by a **demultiplexing key** of the form `<port,host>`. However, UDP does not support any other mechanisms such as flow control or reliable/ordered delivery.

2.1 Implementation of UDP

2.1.1 How to learn of ports?

- Clients send initial messages to specific servers via some *well-known* (published) ports, e.g. port 80 for HTTP.
- A well-known port is used to agree on some other port that will be used subsequently.

2.1.2 How are ports implemented?

- Via message queues.

3 Transmission Control Protocol (TCP)

TCP implements a reliable, connection-oriented byte-stream service with guaranteed in-order delivery, flow control and network congestion control. It supports full-duplex connectivity, i.e. pairs of byte streams between two processes, one stream in each direction. It also offers a **demultiplexing** mechanism, similar to UDP.

3.1 The Segment Format of TCP

While TCP allows senders to write bytes *into a connection* and receivers to read bytes *out of a connection*, it transmits byte sequences in reasonably sized packets, called **segments**.

3.2 When to Send A Segment?

- When the **maximum segment size is reached**.
  - Typically set to about the MTU (maximum transmission unit) of the directly connected network, such that the local IP does not need to fragment segments.
- When the process explicity invokes a **push operation** to flush the TCP buffer of unsent bytes.
- When some **periodic timer** fires (and flushes the TCP buffer).

3.3 How Are TCP Connections Uniquely Identified?

Via a demultiplexingkey that is of the form:

- ~<SrcPort, SrcIPAddr, DstPort, DstIPAddr>

But different incarnations of a connection are possible since connections are established and torn down.

4 Connection Establishment & Termination

- Connection Establishment (**asymmetric Activity**):
  1. Server performs a **passive open** (create buffer to read from).
  2. Client does an **active open** to the server.

3. Two sides echange messages to establish the connection.

- Connection Termination (**Symmetric Activity**):

1. Participant closes one direction of the connection.
2. Connection termination messages are exchanged.
3. Other participant might leave the other direction open.

See slide 13 for **TCP State Diagram**.

4.1 Three-Way Handshake Algorithm for Connection Establishment

The goal of the three-way handshake is to agree on some parameters, such as starting sequence number (for each direction).

Notes:

- Due to different incarnations, it is a good idea to start with *random* sequence numbers instad of 0. Otherwise an acknowledgement from a previous incarnation may be taken as an acknowledgement for the current incarnation.
- The acknowledgement field actually identifies the **next expected sequence number**.

5 Flow Control & Congestion Control

Two different factors can limit the rate at which a source sends data:

- The inability of the destination to accept new data.
  - Techniques that address this are referred to as **flow control**.
- The number of packets within the subnet.
  - Techniques that address this are referred to as **congestion control**.

TCP standard includes both techniques. Some protocol stacks also apply such technique at lower layers (e.g. flow control at the data link level, congestion control at the network level.

5.1 Flow Control

The key functionality of a flow control algorithm is that the sender and receiver have finite buffer space and process data at different rates. Rather than waiting for acknowledgement on every message (which would impact performance greatly) we can send several segments at once. However, the segments should not be removed from the buffer until the acknowledgement is received. This is called a sliding window protocol because there is an upper-bound on *un-ACKed* segments, called a window which limits the amount of buffer space required at sender and receiver. A bigger window means more flexibility but also a greater risk that failures remain undected.

The sender knows:

- Which data has been sent and acknowledged
- The receivers buffer size
- Its own buffer size

The sender creates a window stretching from:

- Last data acknowledged (X), to
- X + send buffer size

If the sender sends data beyond this it can't re-send in case of failure. By a similar process the receiver calculates its available window (**AdvertisedWindow**) and informs the sender of it. The sender does not send information that the receiver cannot handle, i.e. buffer.

5.2 Congestion Control

Typical congestion control techniques at the transport layer.

- Rate Control: Source injection rate is explicitly controlled based on feedback from either the network and/or the receiver (e.g. keep increasing injection rate until packets are dropped / ACKs are not received).
- Admission Control/ Traffic Policing: Only allow connections in if the network can handle them and make sure that admitted sessions do not send them at too high of a rate.

6 Retransmission

Retransmission (from sender) of unacknowledged segments after some timeout has expired. This requires a timer but there are two main ways of choosing the value of said timer:

- Fixed value needs knowledge of network behaviour
  - Does not respond to changing network conditions
  - Too small, network flooded with retransmissions
  - Too large and the receiver stalls
- Adaptive timer
  - Difficult to monitor **Re-Transmission Timer (RTT)** with cumulative acknowledgements.
  - Network conditions may change faster than we can adapt.

6.1 Weighted (Moving) Average

$$EstRTT = \alpha \times EstRTT + \beta \times SampleRTT \tag{1}$$

Where:

- $\alpha + \beta = 1$
- $0.8 \leq \alpha \leq 0.9$
- $0.1 \leq \beta \leq 0.2$

The weighted average is calculated with the following steps:

1. Measure `SampleRTT` for each segment/ACK pair.
2. Compute weighted average of RTT (see equation (1))
3. Set timeout based on `EstRTT`
   - $TimeOut = 2 \times EstRTT$