Computer Systems Tutorial

November 24^{th} 2023

Preliminaries

There were some definitions of bandwidth and delay in the lecture but we will have to be a bit more specific in the exercises.

Bandwidth (digital) It is the data rate of a channel, a quantity measured in bits/sec, i.e. tells how many bits can arrive at a unit of time. That data rate is the end result of using the **analog** bandwidth (measured in Hertz) of a physical channel for digital transmission. Two ways that a digital bandwidth can be increased is either by increasing the analog bandwidth or by sending more than one bit "at once". The formula that relates analog and digital bandwidths is the following (assuming that we are in a noiseless environment)

$$B_{\text{digital}} = 2B_{\text{analog}} log_2 V \text{ bits/sec}$$

where V is the number of discrete levels in the signal (roughly how many bits arrive at once).

Propagation speed s_{prop} This is the speed a signal can travel through a medium, e.g. a wire (Ethernet) or air (WiFi).

Propagation delay This is hence the distance divided by propagation speed. Roughly, it tells how soon a single bit will arrive at the destination (or more than a single bit depending on the encoding).

Bandwidth delay (Transmission delay) It is the time needed for a sender to get the packet onto the wire (note that it is not the time when the receiver receives the packet). This is simply the packet size divided by the bandwidth.

An important difference between bandwidth delay and propagation delay is that bandwidth delay is proportional to the amount of data sent while propagation delay is not. If we send two packets back-to-back, then the bandwidth delay is doubled but the propagation delay counts only once.

Total one-way transmission delay This is hence propagation delay plus bandwidth delay.

There are also queuing delay and **processing** delay that mean the obvious things.

Exercises

Exercise 1 How many edges are in a complete (each node is connected to any other node) graph with 5 nodes? In a graph with n nodes? Assume that only one edge between any two nodes is allowed.

Exercise 2

a Consider the following configuration

$$A \longrightarrow B$$

- Propagation delay is 40 µsec
- Bandwidth is 1 byte/µsec (1 mB/sec, 8 Mbit/sec)
- Packet size is 200 bytes (200 µsec bandwidth delay)

What is the total transmission delay to send one packet from A to B?

b Consider the following configuration

$$A \longrightarrow C \longrightarrow B$$

- Propagation delay is 40 µsec (for each wire)
- Bandwidth is 1 byte/µsec (1 mB/sec, 8 Mbit/sec)
- Packet size is 200 bytes (200 µsec bandwidth delay)

What is the total transmission delay to send one packet from A to B?

c The same as above, but with data sent as two 100-byte packets.

Exercise 3 Now let us consider the situation when the propagation delay is the most significant component. Suppose we need to send a packet from A to B where the distance between A and B is 10000 km, propagation speed is 200 km/ms. Also, suppose that B has to respond to A with some acknowledgement signal. How many bits of data can we send from A to B before A receives the acknowledgement for the first received bit?

At most non-LAN scales, the delay is typically simplified to the round-trip time, or RTT: the time between sending a packet and receiving a response. Try executing `ping google.com` in your terminal.

Exercise 4 This exercise is intended to encourage you to have a look at how various internet protocols look like. Below we will have a look at HTTP. We will need

- Wireshark traffic sniffer `https://www.wireshark.org/`
- Any HTTP server (with no encryption), for example `https://github.com/zowe/sample-node-api` Note that you need `npm` to run the server.

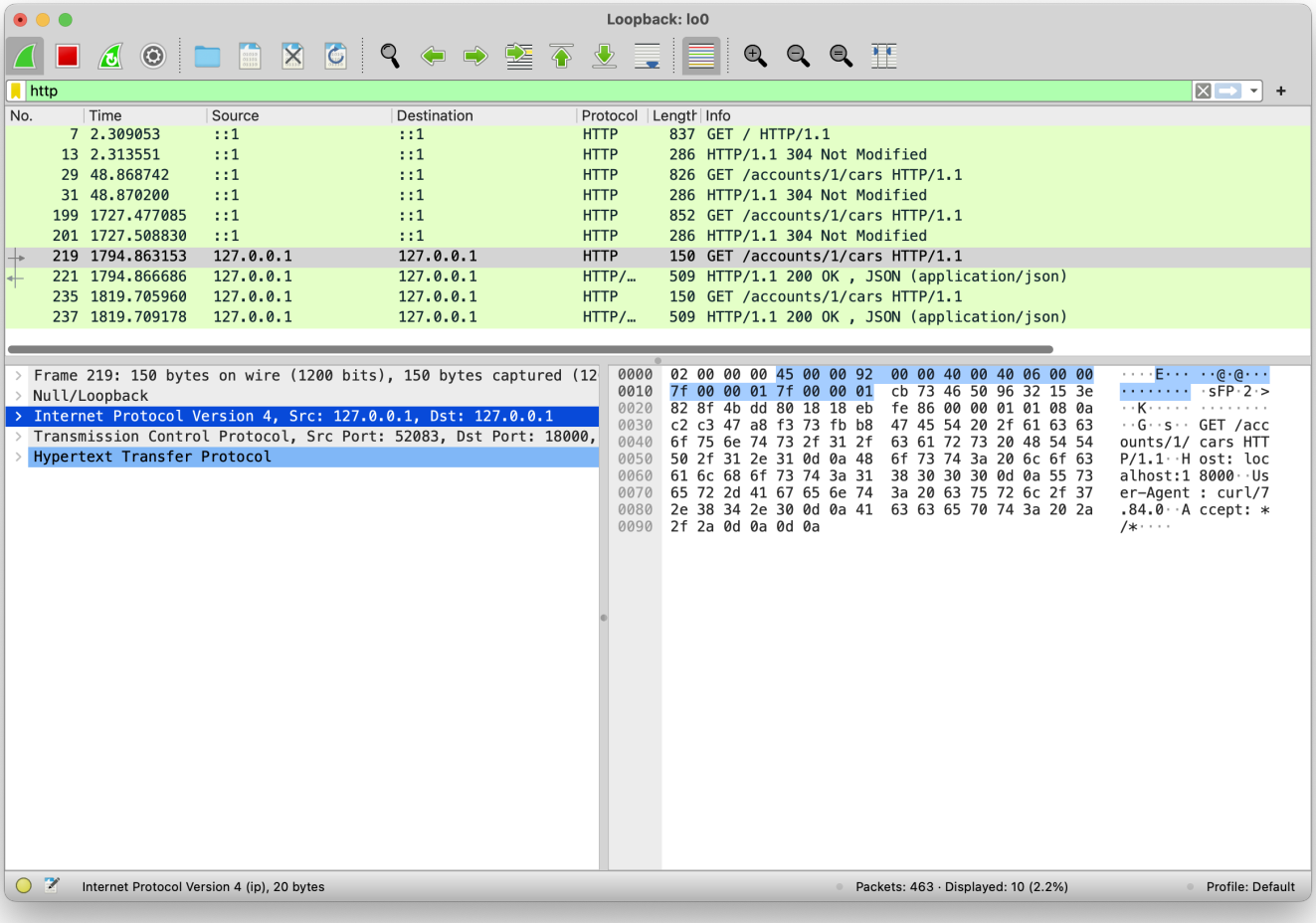

Figure 1: Wireshark example

Clone the repository above, do `npm install;npm start` which will run the server on **localhost:18000**. Start Wireshark and select **loopback** interface for sniffing and start listening on this interface. Then do `curl localhost:18000/accounts/1/cars`. We use `curl` command line utility so that the responses do not get cached. Then stop listening and find the HTTP request that you've sent and HTTP response that you have received. Notice how your HTTP request gets wrapped into TCP and IP packets. The output should look like in Figure 1.

UPD: this will actually work with any web-server. For example, you can make an HTTP request to google by `curl http://www.google.com` and HTTPS request by `curl https://www.google.com`. Compare the corresponding packets in Wireshark (you need to select another interface to listen, e.g. wifi or ethernet).