HW2 Report

Tanvi Aggarwal
SBU ID: 114353100

Part A: Programming - Analysis of assignment2.pcap file

Code overview analysis_pcap_tcp.py:

- **TCPPacket**
  This is a data class for the fields required to store in a TCP packet - source port, destination port, source ip, destination ip, sequence number, ack number, window, header length, payload size, syn flag, ack flag, fin flag, maximum segment size, scaling factor and timestamp.
- **MyTCPParser**
  - **get_all_packets** - this function takes in a pcap object and returns all parsed objects parsed using parse_packet_bytes
  - **parse_packet_bytes** - this function takes the raw bytes and extracts all relevant fields and creates a TCPPacket object. Details about the indices used to parse fields are documented in the code.
- **Analyser**
  - **load_data** - This loads the pcap data and stores it for further analysis
  - **count_flows_initiated** - For a given sender and receiver IP pair, this function computes the number of flows initiated by the sender. This computation is done based on the SYN, SYN-ACK, FIN packets for unique ports. Details in the computation details section below.
  - **get_all_flow_packets** - For a given sender and receiver port pair, this function filters the packets sent by the sender, sent by the receiver and all packets for that flow. This computation is done based on the packets for given ports. Details in the computation details section below.
  - **get_flow_info** - For a given flow (identified by source and destination port pair), this function calls the respective functions to perform analysis (transaction, throughput, loss rate, RTT) and returns the results.
  - **get_transaction_info** - Given the packets sent and received for a given flow, this function extracts sequence number, acknowledgement number and the receive window information for first n transactions after the 3 way handshake is complete. Details in the computation details section below.
  - **compute_throughput** - Given all the packets sent for a flow, this function computes the throughput (the total bytes transferred/time elapsed). Details in the computation details section below.
  - **compute_loss** - Given all the packets sent for a flow, this function computes the loss rate (packets not received by the receiver). Details in the computation details section below.
  - **estimate_RTT** - This function computes the average RTT based on the timestamps of sent and received packets for a given flow. Details in the computation details section below.
- **Main driver code**

Read the pcap file using dpkt library and load the data into the Analyser. Then we compute the number of flows initiated and get the required information for each flow. The results are printed using a utility function print_flow_info.

Computation details:

1. Number of flows
    - We can look at the number of packets with SYN flag from the sender and the number of packets with SYN-ACK flags from the receiver. This tells us the ports of the connections/flows initiated by the sender and acknowledged by the receiver.
    - We also verify the count of packets with FIN flags sent by the sender to verify that any connection on the same port was closed.
    - We also return the unique ports used for each flow for further analysis.
2. Transaction analysis
    - get_all_flow_packets - identify the packets with source and destination port as specified to get the packets sent/received.
    - Given the sent and received packets, we iterate over the sent packets and identify the first n packets with an ACK after the 3 way handshake. We find the corresponding packets at the receiver end (ACK number from sender = SEQ number from receiver for a transaction pair).
    - We also use the scaling factor to compute the window size (2^14).
3. Throughput
    - To get the throughput at the receiver, we look at the packets sent by the sender for throughput assuming that this is the amount of data that the receiver would have processed (sender was able to transmit data at this rate).
    - To calculate the total size of data transferred for all packets we sum up the following for each of the packets - frame + IP header + TCP header + TCP payload size.
    - We also calculate the time elapsed between the first and last packet transmission.
    - Throughput = data transmitted/total time elapsed
    - This is reported in MegaBytes per second.
4. Loss rate
    - This is computed by checking the number of packets retransmitted by the sender (duplicate sequence numbers) divided by the total number of packets sent.
    - We look at the packets sent by the sender for this, assuming that any retransmitted packet was likely re-sent because the receiver did not receive it initially (duplicate ACKs or timeout).
5. Average RTT
    - We compute the round trip time based on the response packet corresponding to the packet sent by the sender.
    - This can be matched by checking the sequence number of the sent packet with the acknowledgment number of the received packet (which would be sequence number of sent packet + size of data in sent packet).
    - Then we compute the RTT for each transaction and compute the average over all of them to get the average RTT. I have ignored the retransmitted packets because there is ambiguity in the time taken (similar to Karn’s algorithm).

- Then we can calculate the theoretical throughput using the formula taught in class
  = sqrt(1.5) * MSS / (Avg RTT * sqrt(loss rate))

Data:

1. Number of flows initiated is 3, on source ports 43498, 43500, 43502 and destination port 80.
2. Flow analysis:

   Flow 1: between sender on port 43498 and receiver on port 80

   a. Transaction analysis

      Transaction 1

      Sender: Sequence number = 705669103, ACK number = 1921750144, window size = 49152

      Receiver: Sequence number = 1921750144, ACK number = 705669127, window size = 49152

      Transaction 2

      Sender: Sequence number = 705669127, ACK number = 1921750144, window size = 49152

      Receiver: Sequence number = 1921750144, ACK number = 705669127, window size = 49152

      Observation:

      - ACK number from sender = SEQ number from receiver for a transaction pair
      - Window size is calculated by multiplying with the scaling factor = 2^14.

   b. Throughput

      5.2514 MBps

   c. Loss rate

      Number of packets lost = 3

      Loss rate = 0.00043 (Number of packets lost/Total number of packets)

   d. Average RTT

      Empirical average RTT = 0.07352s

      Theoretical throughput = 1.1729669709 MBps

      Observation:

      - Actual throughput is more than the theoretical throughput possibly because of some assumptions made while computing the loss rate and average RTTs.

   Flow 2: between sender on port 43500 and receiver on port 80

   a. Transaction analysis

      Transaction 1

      Sender: Sequence number = 3636173852, ACK number = 2335809728, window size = 49152

      Receiver: Sequence number = 2335809728, ACK number = 3636173876, window size = 49152

Transaction 2
Sender: Sequence number = 3636173876, ACK number = 2335809728, window size = 49152
Receiver: Sequence number = 2335809728, ACK number = 3636173876, window size = 49152

Observation:

- ACK number from sender = SEQ number from receiver for a transaction pair
- Window size is calculated by multiplying with the scaling factor = 2^14.

b. Throughput
1.2854 MBps

c. Loss rate
Number of packets lost = 94
Loss rate = 0.013299 (Number of packets lost/Total number of packets)

d. Average RTT
Empirical average RTT = 0.07388s
Theoretical throughput = 0.2098606764 MBps

Observation:

- Actual throughput is more than the theoretical throughput possibly because of some assumptions made while computing the loss rate and average RTTs.

Flow 3: between sender on port 43502 and receiver on port 80

a. Transaction analysis
Transaction 1
Sender: Sequence number = 2558634630, ACK number = 3429921723, window size = 49152
Receiver: Sequence number = 3429921723, ACK number = 2558634654, window size = 49152

Transaction 2
Sender: Sequence number = 2558634654, ACK number = 3429921723, window size = 49152
Receiver: Sequence number = 3429921723, ACK number = 2558634654, window size = 49152

Observation:

- ACK number from sender = SEQ number from receiver for a transaction pair
- Window size is calculated by multiplying with the scaling factor = 2^14.

b. Throughput
1.4815 MBps

c. Loss rate
Number of packets lost = 0
Loss rate = 0.0 (Number of packets lost/Total number of packets)

d. Average RTT
Empirical average RTT = 0.07311s
Theoretical throughput = INF MBps

Observation:

- Since loss rate is zero we see that the theoretical throughput is infinity, but that is not possible practically as there is generally some loss associated.

Part B: Congestion control - Analysis of assignment2.pcap file

Code overview analysis_pcap_tcp.py:

- **TCPPacket**
  This is a data class for the fields required to store in a TCP packet - source port, destination port, source ip, destination ip, sequence number, ack number, window, header length, payload size, syn flag, ack flag, fin flag, maximum segment size, scaling factor and timestamp.
- **MyTCPParser**
  - **get_all_packets** - this function takes in a pcap object and returns all parsed objects parsed using parse_packet_bytes
  - **parse_packet_bytes** - this function takes the raw bytes and extracts all relevant fields and creates a TCPPacket object. Details about the indices used to parse fields are documented in the code.
- **Analyser**
  - **load_data** - This loads the pcap data and stores it for further analysis
  - **count_flows_initiated** - For a given sender and receiver IP pair, this function computes the number of flows initiated by the sender. This computation is done based on the SYN, SYN-ACK, FIN packets for unique ports. Details in the computation details section below.
  - **get_all_flow_packets** - For a given sender and receiver port pair, this function filters the packets sent by the sender, sent by the receiver and all packets for that flow. This computation is done based on the packets for given ports. Details in the computation details section below.
  - **get_flow_info** - For a given flow (identified by source and destination port pair), this function calls the respective functions to perform analysis (compression window sizes, retransmissions) and returns the results.
  - **compute_compression_window_sizes** - this function takes in the packets sent and received and computes the total amount of data sent between two ACKs from the receiver.
  - **count_retransmissions** - this function takes in the packets sent and received and checks the packets which were retransmitted because of a triple duplicate ACK and the total number of retransmissions.
- **Main driver code**
  Read the pcap file using dpkt library and load the data into the Analyser. Then we compute the number of flows initiated and get the required information for each flow. The results are printed using a utility function print_flow_info.

Calculation details:

1. Congestion window
    - Since the congestion window is adjusted whenever we get an ACK (roughly every RTT), we can count the number of bytes sent before an ACK is received.
    - We start counting after the 3 way handshake is complete. We ignore consecutive ACKs from the receiver since no data was sent in between and we are interested in how much data is sent by the sender before the first ACK.
    - Then we compute the growth rate as the increase in window size between two windows.
2. Retransmissions
    - We assume that retransmissions happen either because of the triple duplicate ACKs or because of timeouts.
    - We can get the sequence numbers for which retransmissions were made and check if the receiver had sent duplicate ACKs for that packet. The duplicate ACKs will have the same ACK number as the sequence number of the sent packet (since receiver is still waiting for that packet from sender).
    - Retransmissions due to timeout = Total retransmission - retransmissions due to triple duplicate ACKs.

Data:

Flow 1: between sender on port 43498 and receiver on port 80

a. Congestion windows
   window sizes are: [11584, 13032, 14480, 15928, 17376, 18824, 20272, 26064, 27512, 33304]
   growth rates: [1.12, 1.11, 1.1, 1.09, 1.08, 1.08, 1.29, 1.06, 1.21]

b. Retransmissions
   Retransmissions due to triple duplicate ACKs = 2
   Retransmissions due to timeout = 1 (Total retransmissions - Triple duplicate ACKs)

Flow 2: between sender on port 43500 and receiver on port 80

c. Congestion windows
   window sizes are: [11584, 13032, 15928, 17376, 17376, 18824, 26064, 27512, 33304, 33304]
   growth rates: [1.12, 1.22, 1.09, 1.0, 1.08, 1.38, 1.06, 1.21, 1.0]

d. Retransmissions
   Retransmissions due to triple duplicate ACKs = 28
   Retransmissions due to timeout = 66 (Total retransmissions - Triple duplicate ACKs)

Flow 3: between sender on port 43502 and receiver on port 80

e. Congestion windows
   window sizes are: [11584, 13032, 26064, 27512, 24616, 24616, 27512, 27512, 27512, 27512]
   growth rates: [1.12, 2.0, 1.06, 0.89, 1.0, 1.12, 1.0, 1.0, 1.0]

f. Retransmissions

Retransmissions due to triple duplicate ACKs = 0
Retransmissions due to timeout = 0 (Total retransmissions - Triple duplicate ACKs)

Analysis:

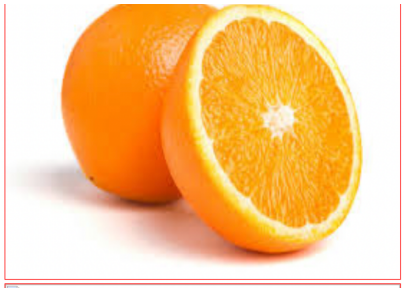
1. The congestion window sizes increase multiplicatively in most cases but not by a factor of 2. The factors by which congestion window sizes increase vary.
2. In flow 1 and 2 we don't observe a decrease and in flow 3 as well (for the most part) suggesting that there are no losses in the first 10 windows as such.
3. The initial congestion window size comes out to approximately 11584 bytes as determined based on the packets sent by the sender before receiving an ACK from the receiver. We could have calculated the packets from the receiver and ACK from the sender but since the data was captured at the sender, I decided to go with the first approach.
4. The retransmissions are computed based on the packets sent by the sender which have duplicate sequence numbers. For a given sequence number, if we find triple duplicate ACKs, it is counted as a retransmission due to Triple Duplicate ACK. Otherwise we assume that the retransmission was due to timeout.

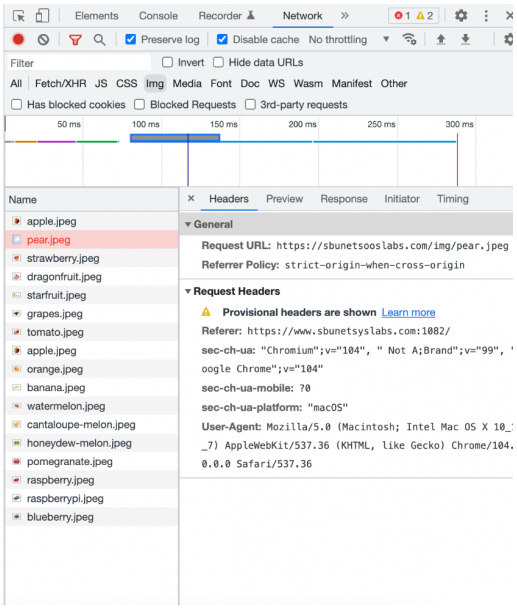
Part C: HTTP Analysis - Generating http_1080.pcap, http_1081.pcap, http_1082.pcap file

Commands used:

1. sudo tcpdump -i en0 -n port 1080 -w http_1080.pcap
2. sudo tcpdump -i en0 -n port 1081 -w http_1081.pcap
3. sudo tcpdump -i en0 -n port 1082 -w http_1082.pcap

Note: For all three experiments, one of the objects did not load successfully (pear.jpeg) even after multiple attempts. This was because the request URL embedded in the page was not resolved (possibly a typo - "sbunetsooslabs.com" instead of "sbunetsyslabs.com"). Screenshot attached below from experiment 3, i.e. port 1082.





Code overview analysis_pcap_http.py:

- **TCPPacket**
  This is a data class for the fields required to store in a TCP packet - source port, destination port, source ip, destination ip, sequence number, ack number, window, header length, payload size, syn flag, ack flag, fin flag, maximum segment size, scaling factor and timestamp.
- **MyTCPParser**
  - **get_all_packets** - this function takes in a pcap object and returns all parsed objects parsed using parse_packet_bytes
  - **parse_packet_bytes** - this function takes the raw bytes and extracts all relevant fields and creates a TCPPacket object. Details about the indices used to parse fields are documented in the code.
- **Analyser**
  - **load_data** - This loads the pcap data and stores it for further analysis
  - **count_flows_initiated** - For a given sender and receiver IP pair, this function computes the number of flows initiated by the sender. This computation is done based on the SYN, SYN-ACK, FIN packets for unique ports. Details in the computation details section below.
  - **get_all_flow_packets** - For a given sender and receiver port pair, this function filters the packets sent by the sender, sent by the receiver and all packets for that flow. This computation is done based on the packets for given ports. Details in the computation details section below.
  - **get_flow_http_info/reorder packets** - For a given flow (identified by source and destination port pair), this function reorders the HTTP request/response packets. Details in the computation details section below.
  - **compute_statistics** - this function takes in all the packets sent and received for all flows and computes the total amount of data sent, total number of packets sent and total transmission time. Details in the computation details section below.
- **Main driver code**
  Read the pcap file using dpkt library and load the data into the Analyser. Then we compute the number of flows initiated and get the required information for each flow. The results are printed using a utility function print_http_info.

Calculation details:

1. Number of flows
   - We can look at the number of packets with SYN flag from the sender and the number of packets with SYN-ACK flags from the receiver. This tells us the ports of the connections/flows initiated by the sender and acknowledged by the receiver.
   - We also verify the count of packets with FIN flags sent by the sender to verify that any connection on the same port was closed.
   - We also return the unique ports used for each flow for further analysis.
2. Reordering HTTP packets
   - Based on the packets sent and received for each flow, we can find the HTTP request/response packets.

- From the packets sent by sender for a flow, we look for the payload with “GET” keyword in them. These are the HTTP get requests.
- Then we can look for the HTTP response packets in the received packets by matching the acknowledgement number of sent request with the sequence number of the received packet.
- After the first response packet, we identify subsequent response packets based on the size of the payload of the response packet, until there is a FIN packet or the payload is empty.

3. Statistics
   - Time taken to load = time elapsed between the last and first packet timestamps for the experiment pcap file
   - Total number of packets transmitted = number of packets in the dump
   - Total amount of data transmitted = sum of size of all packets

Data:

1. For http_1080.pcap, the number of flows initiated is 17. Reconstructed HTTP request/response from http_1080.pcap into `part_c/part_c_output.txt`. Check the file for details.

2. HTTP versions - We can find out the number of flows initiated using the logic described above. This gives us an indication of the HTTP protocol that might be used.
   a. http_1080.pcap likely uses HTTP/1.0 since it opens 17 connections and we see that each connection is used for fetching a single request/response (from analysis in part c1).
   b. http_1081.pcap likely uses HTTP/1.1 since it opens 6 connections. HTTP/1.1 has persistent, parallel connections and the default max number of parallel connections is 6 for Google chrome browser.
   c. http_1082.pcap likely uses HTTP/2 since it opens 1 connection. HTTP/2 uses a single TCP connection and multiplexes multiple request/response objects into a single pipe.

3. Time statistics

   Data:

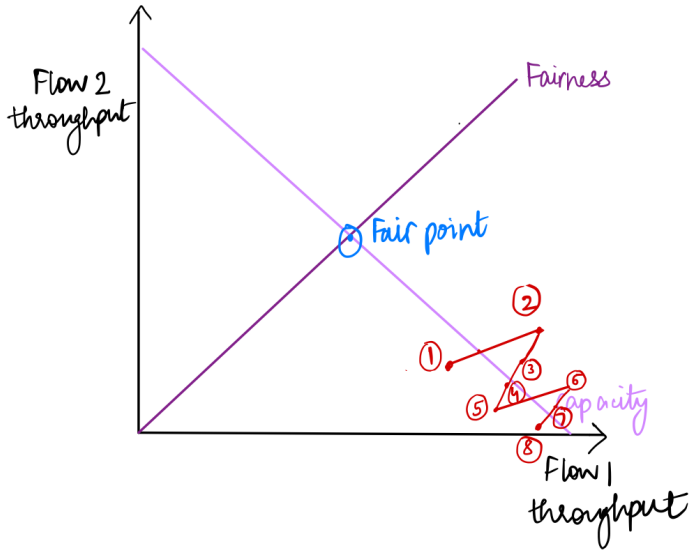
   a. <u>http_1080.pcap</u>
      Number of flows initiated = 17
      Time taken to load = 0.39524s
      Total packets transferred = 1854
      Total size of data transferred = 2296576
   b. <u>http_1081.pcap</u>
      Number of flows initiated = 6
      Time taken to load = 5.29041s
      Total packets transferred = 1762
      Total size of data transferred = 2280976
   c. <u>http_1082.pcap</u>
      Number of flows initiated = 1
      Time taken to load = 5.34571s
      Total packets transferred = 1613
      Total size of data transferred = 2282850

Analysis:

1. We see that the maximum number of packets are sent in HTTP/1.0 (http_1080.pcap). This makes sense because there is no compression/multiplexing.
2. We see that the least number of packets are sent in HTTP/2 (http_1082.pcap). This makes sense HTTP/2 uses a single TCP connection and multiplexes requests/responses together. Since the page we loaded contains multiple small images, they could be batched together.
3. We see that the maximum number of total bytes are sent in HTTP/1.0 (http_1080.pcap). Both HTTP/1.1 and HTTP/2 send less data. This makes sense because there is no header compression in HTTP/1 and we have to establish more connections.
4. Surprisingly, HTTP/2 sends marginally more data than HTTP/1.1. This could be an experimental anomaly but is tough to analyze because we have encrypted traffic.
5. Page load time is again surprisingly smaller (0.4 seconds) for HTTP/1.0, while HTTP/1.1 and HTTP/2.0 which have a comparable PLT of ~5.3 seconds. Since both HTTP/1.1 and HTTP/2.0 have a secure connection, we expect an overhead for exchanging TLS keys. This could be the reason for higher PLT.

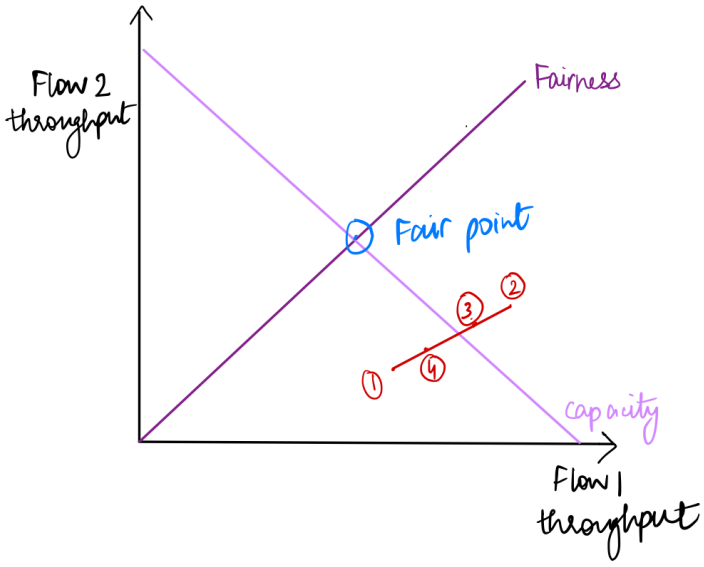
Part D: Fairness

Case 1: Multiplicative increase additive decrease (MIAD)



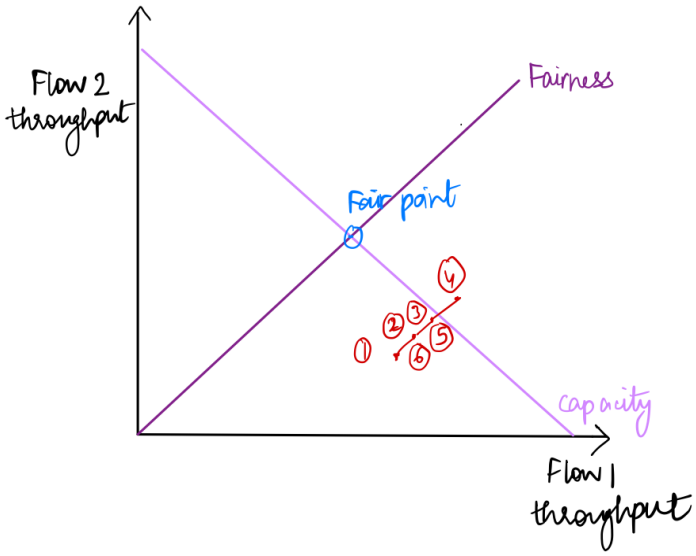
The sequence of points is shown from 1 to 8. What we observe is that the capacity is reached quickly with multiplicative increase (x, y) to (x*k, y*k). However, the decrease is additive and that happens slowly. Moreover it goes further away from the fair point with each iteration (x*k, y*k) to (x*k-c, y*k-c). Thus, MIAD is not fair.

Case 2: Multiplicative increase multiplicative decrease



The sequence of points is shown from 1 to 4. What we observe is that the capacity is reached quickly with multiplicative increase (x, y) to (x*k, y*k). Since the decrease is multiplicative, it happens quickly and that follows the same path (x*k, y*k) to (x*k/c, y*k/c). Thus, we observe that the ratio of utilization of flow 1 and flow 2 always remains the same, so it will not reach the fair point unless we start from there. Thus, MIMD is also not fair.

Case 3: Additive increase additive decrease



The sequence of points is shown from 1 to 6. What we observe is that the capacity is reached slowly with additive increase (x, y) to (x+c1, y+c1). Since the decrease is also additive, it is slow and follows the same path (x+c1, y+c1) to (x+c1-c2, y+c1-c2). Like MIMD, we observe that the ratio of utilization of flow 1 and flow 2 always remains the same, so it will not reach the fair point unless we start from there. Thus, AIAD is also not fair.