Computer Networks
Homework 3

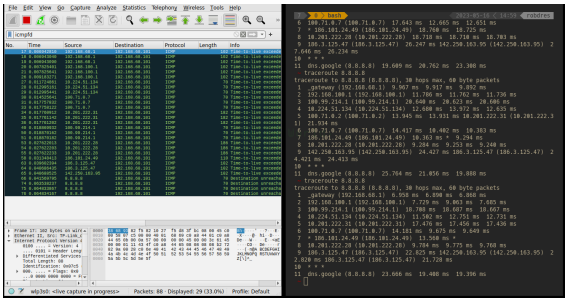
Nombre: Roberto Alvarado
BanneriD: 00206411

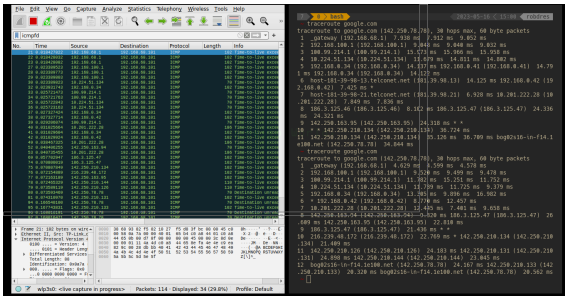
Universidad San Francisco

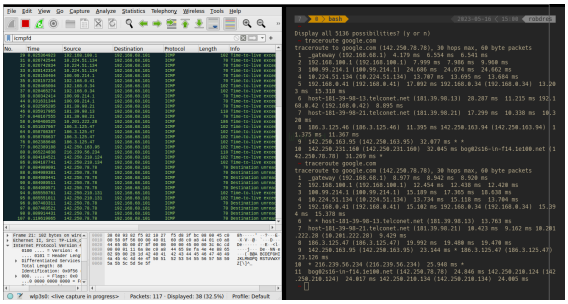
Exercise 1:

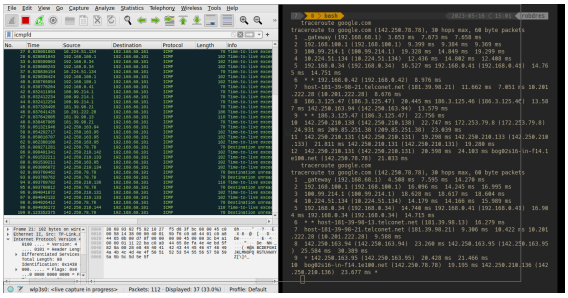
> Read the following Wireshark tutorial, and use it to capture traffic from the following scenarios. Use screenshots to show your results.
>
> - Run 10 traceroute commands against google.com
> - Watch a video from youtube.com. Capture the TCP handshake, and the congestion window.

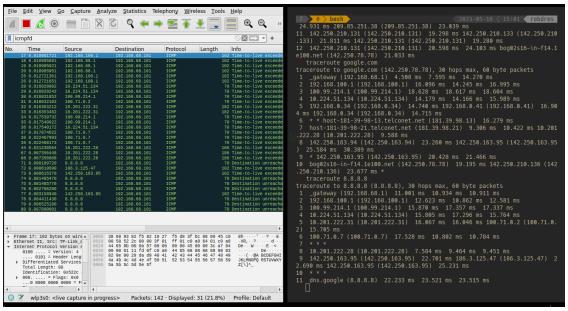
- Results of 10 traceroute commands

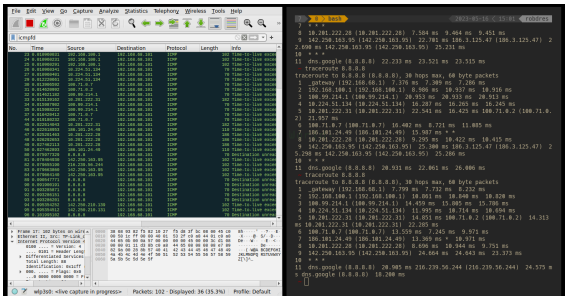


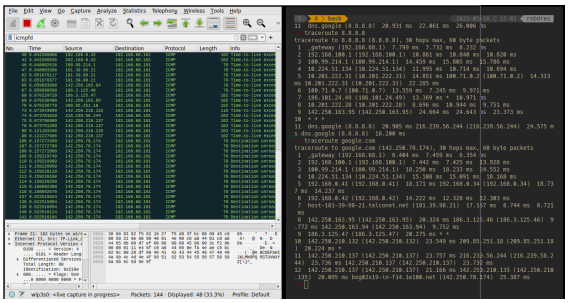


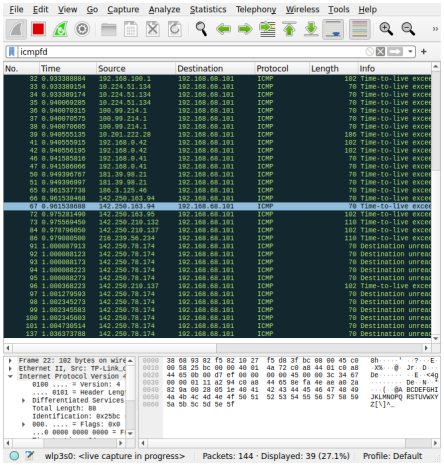


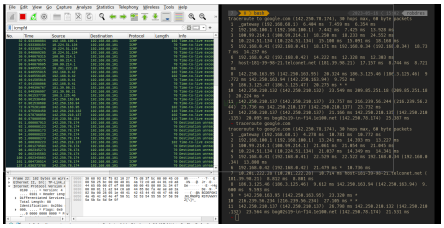


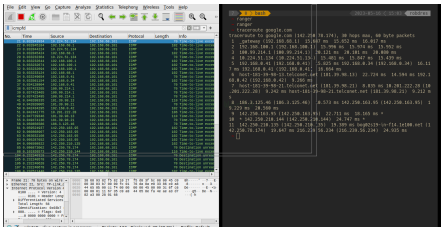






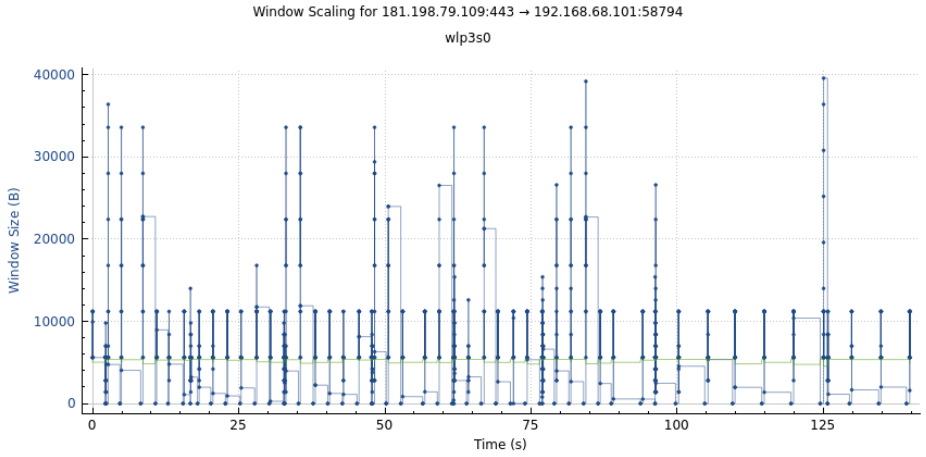


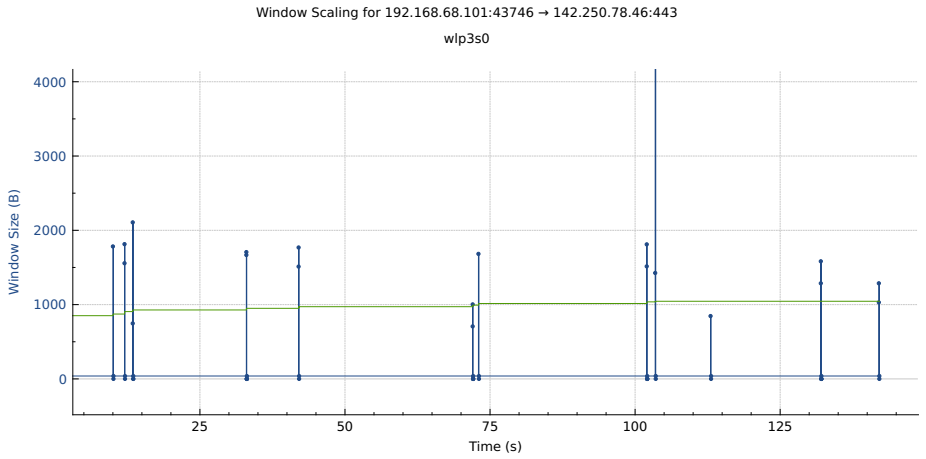




- Now first the handshake And now the results

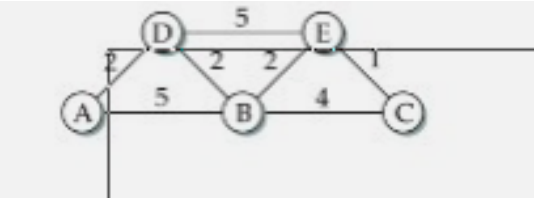
4… 25.1129…	52.88.48.20	192.168.68.101	TCP	66 [TCP ACKed unseen segment] 443 → 60972 [ACK] Seq=1 Ack=2 Win=110
4… 26.9212…	192.168.68.101	100.20.231.226	TCP	74 54134 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval
4… 27.1716…	192.168.68.101	100.20.231.226	TCP	74 54138 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval
4… 27.1755…	100.20.231.226	192.168.68.101	TCP	74 443 → 54134 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1412 SACK
4… 27.1755…	192.168.68.101	100.20.231.226	TCP	66 54134 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2527891601 T





Exercise 2:

Use Dijkstra's to get the routing tables for nodes A, B and E.



Sol.

```
1
2  from queue import PriorityQueue
3  class Graph:
4    def __init__(self, num_of_vertices):
5      self.v = num_of_vertices
6      self.edges = [[-1 for i in range(num_of_vertices)] for j in range(num_of_vertices)]
7      self.visited = []
8    def add_edge(self, u, v, weight):
9      self.edges[u][v] = weight
10     self.edges[v][u] = weight
11
12 def dijkstra(graph, start_vertex):
13   D = {v:float('inf') for v in range(graph.v)}
14   D[start_vertex] = 0
15
16   pq = PriorityQueue()
17   pq.put((0, start_vertex))
18
19   while not pq.empty():
20     (dist, current_vertex) = pq.get()
21     graph.visited.append(current_vertex)
22
23     for neighbor in range(graph.v):
24       if graph.edges[current_vertex][neighbor] != -1:
25         distance = graph.edges[current_vertex][neighbor]
26         if neighbor not in graph.visited:
27           old_cost = D[neighbor]
28           new_cost = D[current_vertex] + distance
29           if new_cost < old_cost:
30             pq.put((new_cost, neighbor))
31             D[neighbor] = new_cost
32   return D
33
34 g = Graph(5)
35 g.add_edge(0,1,5)
36 g.add_edge(0,3,2)
37 g.add_edge(1,3,2)
38 g.add_edge(1,4,2)
39 g.add_edge(1,2,4)
40 g.add_edge(2,4,1)
```

```
41 g.add_edge(3,4,5)
42
43 print(dijkstra(g,0))
```

El resultado es

```
1 {0: 0, 1: 4, 2: 7, 3: 2, 4: 6}
```

Entonces routing tables

Node	Next
B	D
C	D
D	D
E	D

Node	Next
A	D
C	E
D	D
E	E

Node	Next
A	E
B	E
D	E
E	E

Node	Next
A	A
B	B
C	B
E	B

Node	Next
A	B
B	B
C	C
D	B

Exercise 3:

> Suppose a host wants to establish the reliability of a link by sending packets and measuring the percentage that are received; routers, for example, do this. Explain the difficulty of doing this over a TCP connection.

Sol. There is one main reason why using TCP, could be a problem. The way that it manages lost packages or broken packages, as if a package in this protocol is lost, then it will be retransmitted, but the sending host will only receive the response from the destination host, it will not know if the package was retransmitted, if we could add a tag to the package if it was retransmitted.
So at the end if we want to manage this analysis using TCP, as we could be receiving ACK, without knowing the loss, then we could add maybe a TIME tag so that we manage it

Exercise 4:

Consider a simple congestion control algorithm that uses linear increase and multiplicative decrease (no slow start). Assume the congestion window size is in units of packets rather than bytes, and it is one packet initially.

- Give a detailed sketch of this algorithm.
- Assume the delay is latency only, and that when a group of packets is sent, only a single ACK is returned.
- Plot the congestion window as a function of RTT for the situation in which the following packets are lost: 9, 25, 30, 38 and 50. For simplicity, assume a perfect timeout mechanism that detects a lost packet exactly 1 RTT after it is transmitted.

Sol.

- The idea of the algorithm is

```
1  [IF ACK]  cwnd <- cwnd+a
2  [IF TIMEOUT]  cwnd <- cwnd/b
3
```

  Parameters have to be defined has to be define, let $a = 1, b = 2$

- In this case we could use an example on how it would increases, and using the case of the last example

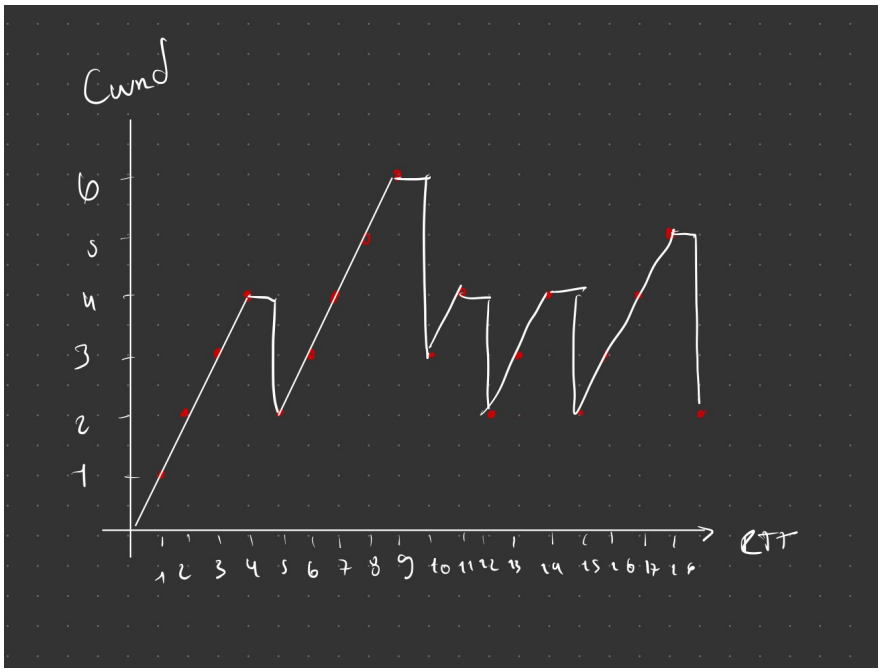


Figure 1

Table 1

RTT STEP	**Last Package Sent**	**# packages sent**
1	1	1
1	3	2
2	6	3
3	10	4
4	9-10	2
5	13	3
6	15	4
7	22	5
8	28	6
9	25-27	3
10	32	4
11	30-31	2
12	34	3
13	38	4
14	38-39	2
15	42	3
16	46	4
17	51	5
18	50-51	2