Deber #3

1

- Disclaimer: My current computer is owned by my employer. No matter what I tried, I couldn't manage to make Wireshark work on it. I'm out of time to try something different. I'm sorry about the first exercise.

2 Based on the following python implementation:

```python
import heapq

def calculate_distances(graph, starting_vertex):
    distances = {vertex: float('infinity') for vertex in graph}
    distances[starting_vertex] = 0

    pq = [(0, starting_vertex)]
    while len(pq) > 0:
        current_distance, current_vertex = heapq.heappop(pq)

        if current_distance > distances[current_vertex]:
            continue

        for neighbor, weight in graph[current_vertex].items():
            distance = current_distance + weight

            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(pq, (distance, neighbor))

    return distances

graph = {
    'A': {'B': 5, 'D': 2},
    'B': {'A': 5, 'D': 2, 'E': 2, 'C': 4},
    'C': {'B': 4, 'E': 1},
    'D': {'A': 2, 'B': 2, 'E': 5},
    'E': {'D': 5, 'B': 2, 'C': 1}
}

distances_from_A = calculate_distances(graph, 'A')
distances_from_B = calculate_distances(graph, 'B')
distances_from_E = calculate_distances(graph, 'E')

print("Shortest distances from node A:")
for node, distance in distances_from_A.items():
    print(f"Node {node}: {distance}")

print("\nShortest distances from node B:")
for node, distance in distances_from_B.items():
    print(f"Node {node}: {distance}")

print("\nShortest distances from node E:")
for node, distance in distances_from_E.items():
    print(f"Node {node}: {distance}")
```

```
Shortest distances from node A:
Node A: 0
Node B: 4
Node C: 7
Node D: 2
Node E: 6

Shortest distances from node B:
Node A: 4
Node B: 0
Node C: 3
Node D: 2
Node E: 2

Shortest distances from node E:
Node A: 6
Node B: 2
Node C: 1
Node D: 4
Node E: 0
```

3

- Con TCP no hay manera exacta de determinar si un paquete llegó al primer intento o se tuvo que volver a enviar porque se perdió. Si el recibidor responde con un eco apenas le llega el paquete, eso podría ayudar a saber si es que se consiguió. Muchas implementaciones derivadas del método Berkeley miden timeouts con una granularidad de 0.5 segundos y RTTs para un solo enlace sin pérdida serían hasta dos ordenes de magnitud más pequeños pero su implementación es realmente compleja.

4

a) Empieza cwnd = 1; con el ACK: cwnd = cwnd + 1/cwnd
con timeout: cwnd = min(1, cwnd/2)

b) Basado en la asunción proporcionada, el algoritmo operaría así:

Empieza cwnd = 1; Por cada ACK cwnd = cwnd + 1/cwnd
Por cada timeout cwnd = cwnd + cwnd/2

C) la congestion window size (cwnd) trabaja en función del RTT cuando se pierden los paquetes: 9, 25, 30, 38 y 50. S: el timeout = 1RTT

RTT	1	2	3	4	9 se pierde	5	6	7	8	9
enviado	1	2-3	4-6	7-10	cwnd reduce a 2	9-10	11-13	14-17	18-22	23-25

25 se pierde	10	11	30 se pierde	12	13	14	38 se pierde
cwnd reduce a 3	25-27	28-31	cwnd reduce a 2	30-31	32-34	35-38	cwnd reduce a 2

15	16	17	18	50 se pierde	19
38-39	40-42	43-46	47-50	cwnd reduce a 2	50

Plot: