Spring
99

# Assignment #3

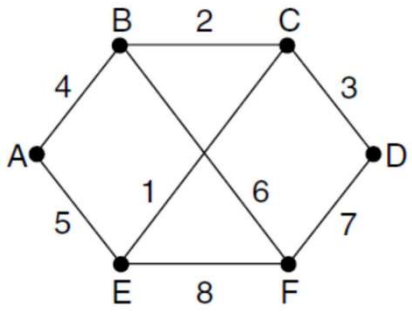
COMPUTER NETWORKS

SAHAND KHOSHDEL – ST ID: 810196607

**TABLE OF CONTENTS**

## Q1) *DVR (Distance Vector Routing) Algorithm*

### a) Running Distance Vector Routing Algorithm for node A (as Source) (A's Routing Table):




| Node (Destination) / Iteration(State) | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| Initial state (0) | - | (-1,∞) | (-1,∞) | (-1,∞) | (-1,∞) | (-1,∞) |
| (1) | - | (B,4) | (-1,∞) | (-1,∞) | (E,5) | (-1,∞) |
| (2) | - | (B,4) | (B,6) / (E,6) | (-1,∞) | (E,5) | (B,10) |
| (3) | - | **(B,4)** | **(B,6) / (E,6)** | **(B,9) / (E,9)** | **(E,5)** | **(B,10)** |

*Figure 1.1 – Running DVR Algorithm to build "A" 's Routing Table*

e

- The DVR Algorithm converged after 3 iterations (4 states including initial) and th last row of the table above shows A's routing table values.

| Destination | Assigned pair |
|---|---|
| A | - |
| B | **(B,4)** |
| C | **(B,6) / (E,6)** |
| D | **(B,9) / (E,9)** |
| E | **(E,5)** |
| F | **(B,10)** |

*Figure 1.2 – "A" 's Routing Table*

- ***Describing the steps (iterations) to convergence:***

*State 0 (Initial):*
"A" doesn't have any information from any other nodes in the network (so we assign the pair "(-1, ∞)" to it contractually.

*State 1:*
"A" is informed of the distance to its neighbors in the first iteration, (C, D, F) remain unknown in "A" 's routing table as they are 2 hops away.

*State 2:*
"A" is informed of the distance to its second order neighbors. (C, E) obtain values but D remains unknown as its 3 hops away.

(The route to C can be starts with either B or E, as both lead to the same amount of cost(distance) and hop count. (Congestion control Algorithms may send packets in both directions in order to control network traffic)

*State 3: (Convergence)*
"A" has been informed of all the nodes in the network and now can decide which route is the best to any destination using "Vector Routing Algorithm"

## b) Running Distance Vector Routing Algorithm for all nodes towards "A" (as Destination) (the columns dedicated to "A" in each routing table):

| Node (Source) / Iteration(State) | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| Initial state (0) | - | (-1,∞) | (-1,∞) | (-1,∞) | (-1,∞) | (-1,∞) |
| (1) | - | (A,4) | (-1,∞) | (-1,∞) | (A,5) | (-1,∞) |
| (2) | - | (A,4) | (B,6) / (E,6) | (-1,∞) | (A,5) | (B,10) |
| (3) | - | **(A,4)** | **(B,6) / (E,6)** | **(C,9)** | **(A,5)** | **(B,10)** |

*Figure 1.3- Running DVR algorithm to find cells dedicated to "A" (as destination) in each routing table*

- *Describing the steps (iterations) to convergence:*

*State 0 (Initial):*
None of the nodes have any information about "A" and routes towards it. All assign the pair "(-1, ∞)" as a consequence.

*State 1:*
"B", "E" (neighbors of "A") obtain their distance to "A". They reach their neighbor via itself as they are only 1 hop away. Other's path remains unknown as they are farther away.

*State 2:*
"C", "F" figure out that their route towards "A" goes through E/B and B correspondingly. "D" 's path remains unknown as it is 3 hops away.

*State 3:*
All nodes have been informed of their route towards "A" and how much it costs.

- **The last row of the table (Figure1.2) summarizes all the routes to the destination "A" via links in between.**

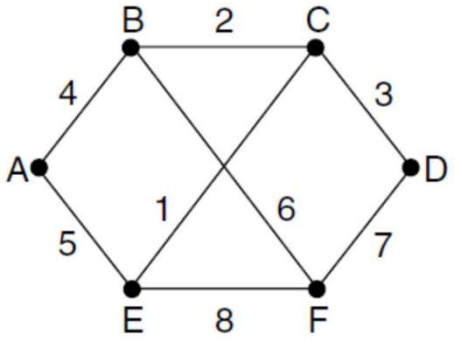
*For Example:*

For **sending a packet from the node "D" to "A"** the table tells us we should go to "C" at the first place (**Cost = 9**; Hop count = 1). Next we should visit either B or E according to C's column in the table. Let's assume we choose "B" as the costs are equal (Hop count = 2). At the last hop **we reach our destination** "A" as "B" is a neighbor **(Hop count = 3)**

## Q2) ***Dijkstra's Algorithm:***

- **Dijkstra's Algorithm** is part of a bigger procedure called **"link state routing"**:

- We will follow "link state routing" steps and run Dijksta's Algorithm in the last step that eventually defines the best route among any source and destination.




- *Step 1. Discovering Neighbors:*

Each node sends a "Hello" packet to all its neighbors and receives the ACK for it including the receiver's name (IP address). It also can measure the delay, bandwidth and other properties within this round-trip if needed. (If the Network Operator doesn't know it yet.)

- *Step 2. Setting Link Costs:*

The cost to reach neighbors can be set automatically or by the Network Operator. The Network Operator (Usually one of the Routers or a user behind one of the devices) assigns costs to the links according to their propagation delay, bandwidth, etc. (The metric that has the more importance for the Operator has more effect on calculating the cost.

- *Step 3. Building the "link state packets" for the network:*

Each node creates a link state packet that includes the information of its neighbors (name and distance(cost)). The purpose of link state algorithm and eventually "Dijkstra" is to spread the information each node has obtained across the entire network (via sending link state packets to neighbors in each iteration)

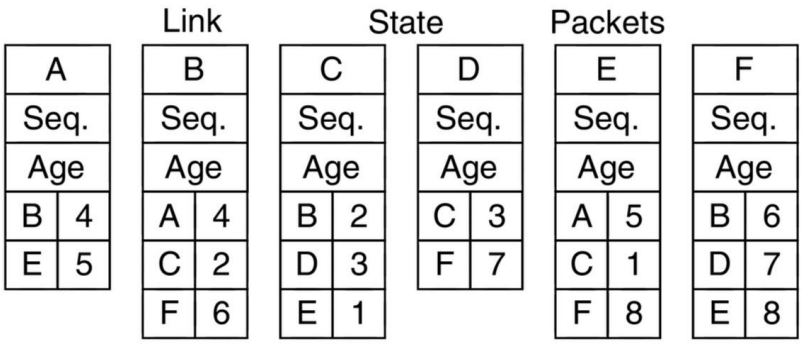



*Figure 2.1- Link State Packets Generated by each node*

- **Seq** is a parameter for preventing duplicate frames reaching destination router multiple times and prioritizing older data over recent data.
- **Age** is a measurement parameter indicating hop count (Hop count = Initial Age – Current Age). It also prevents data from flowing in the network for a long time (If the Age becomes 0, the router will discard the link state packet). Each router decrements Age number by one by every hop count.

- *Step 4. Running Dijkstra's Algorithm:*

| Permanently Labeled | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | - | **(B,4)** | (-1,∞) | (-1,∞) | (E,5) | (-1,∞) |
| A,B | | - | (B,6) | (-1,∞) | (E,5) | (B,10) |
| A,B,E | | - | (B,6) | (-1,∞) | - | (B,10) |
| A,B,E,C | | - | - | (C,9) | - | (B,10) |
| A,B,E,C,D | | - | - | - | - | (B,10) |
| A,B,E,C,D,F | | - | - | - | - | - |

*Figure 2.2- Dijkstra's Optimal Path Algorithm*

- In each iteration nodes send their "link state packets" to their neighbors.
  The node which has the least distance from the source, will be permanently labeled (the pair is confirmed as the shortest path to that particular and sent on the routing table) and no more mentioned in the next iterations. Assuming we have M nodes in the network, after M-1 iterations (M states) the algorithm converges and all nodes have been labeled.

- (M=6 in our case, converges after 5 iterations that is more compared to DVR which was 3 but doesn't have DVR issues)
- **The pairs which are permanently labeled are in fact the cells of "A" 's routing table.**

## Q3) *Reverse Path Forwarding Algorithm*

- The "Reverse Path Forwarding" algorithm uses "flooding" technique that is a common technique in broadcasting**. In the flooding technique each incoming packet to a router is forwarded on every outgoing link of the router except the one it has arrived on**. Flooding has issues like generating lots of duplicate packets that can be solved using to flooding control parameters called hop count (Age), Sequence number. With this technique we are sure each router has received the broadcast packet.

- If the broadcast packet arrives on **a link that's not on the preferred(optimal) path from the source** (in other words if the link is **not included in the sink tree of the network graph**), it will be considered **duplicate and discarded** immediately.

- **Before** the **"Reverse Path Algorithm"** is performed, the **routers need to know** how to reach all destinations (Need to know **network topology + optimal path**) which is why link state routing and the source(A)'s routing table is needed and why this problem is referred to the answer of Problem #2.

- According to the routing table of A, B and E are directly reached by A and nodes C, F are reachable via B, and finally D is reached via (B-C). so **the sink tree (optimal path for broadcasting)** will be as shown in the figure below:

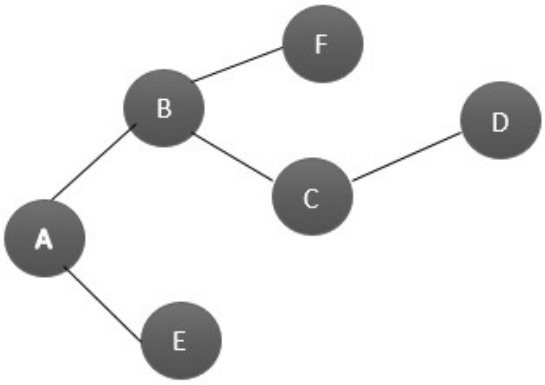



*Figure 3.1- Sink Tree of Reverse path forwarding*

- The final purpose of all routing algorithms is to reach this sink tree that shows the best path for any packet transfer.

- Reverse Path routing algorithm terminates non-optimal paths after using a non-optimal link. But some algorithms such as Spanning Tree algorithm that define the route before broadcasting don't send packets on non-optimal paths at all (number of used links are optimized).

## Q4) *Traffic shaping: Token bucket, Leaky Bucket:*

**There was a contradiction between the data burst times reported in the graph and the time reported in the problem.**

.

Assuming time values in the graph leads to congestion (as $t_2 = 200$ (the time where first data burst is expected to be fully transferred) is after the second data burst causing complexity and not letting us to transmit with a higher data rate (because the bucket is empty and the data rate remains at 10MBps. Therefore, we assume that the burst time intervals are 10msec long according to the problem's text

- Token bucket and Leaky bucket techniques are traffic shaping methods used in congestion control.

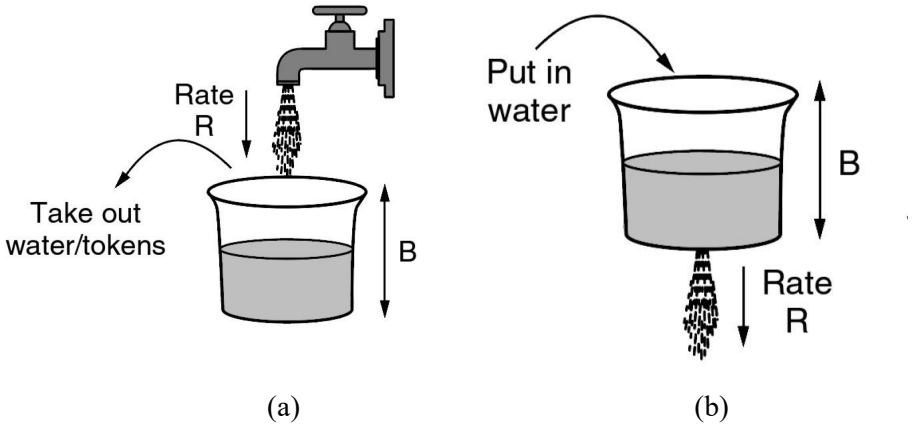



*Figure 4.1- Traffic shaping; Token Bucket (a), Leaky Bucket (b)*

- Token bucket is a method where we can give users which use the net much less than others higher data rates for a limited amount of time, This process is modeled with constantly giving them tokens(water) when they are idle that let them use the net with a higher data rate than usual when needed, until finally the bucket becomes empty and the output rate is limited to the token entry rate ($R_{in} = R_{out}$),whenever the users access is over, the token entry rate ($R_{in}$) will fill the bucket again in the absence of $R_{out}$

### *a,b)*

- B (bucket capacity) = maximum token size = 1MB
- $R_{in}$ (bucket entry rate) = 10MBps
- $R_{burst}$ (received burst data rate from transport layer) = 100 MBps, in 2 bursts, each lasting for 10 msec (40-50,140-150)
- $R_{out}$: The data rate which the network layer transmits following token-bucket.

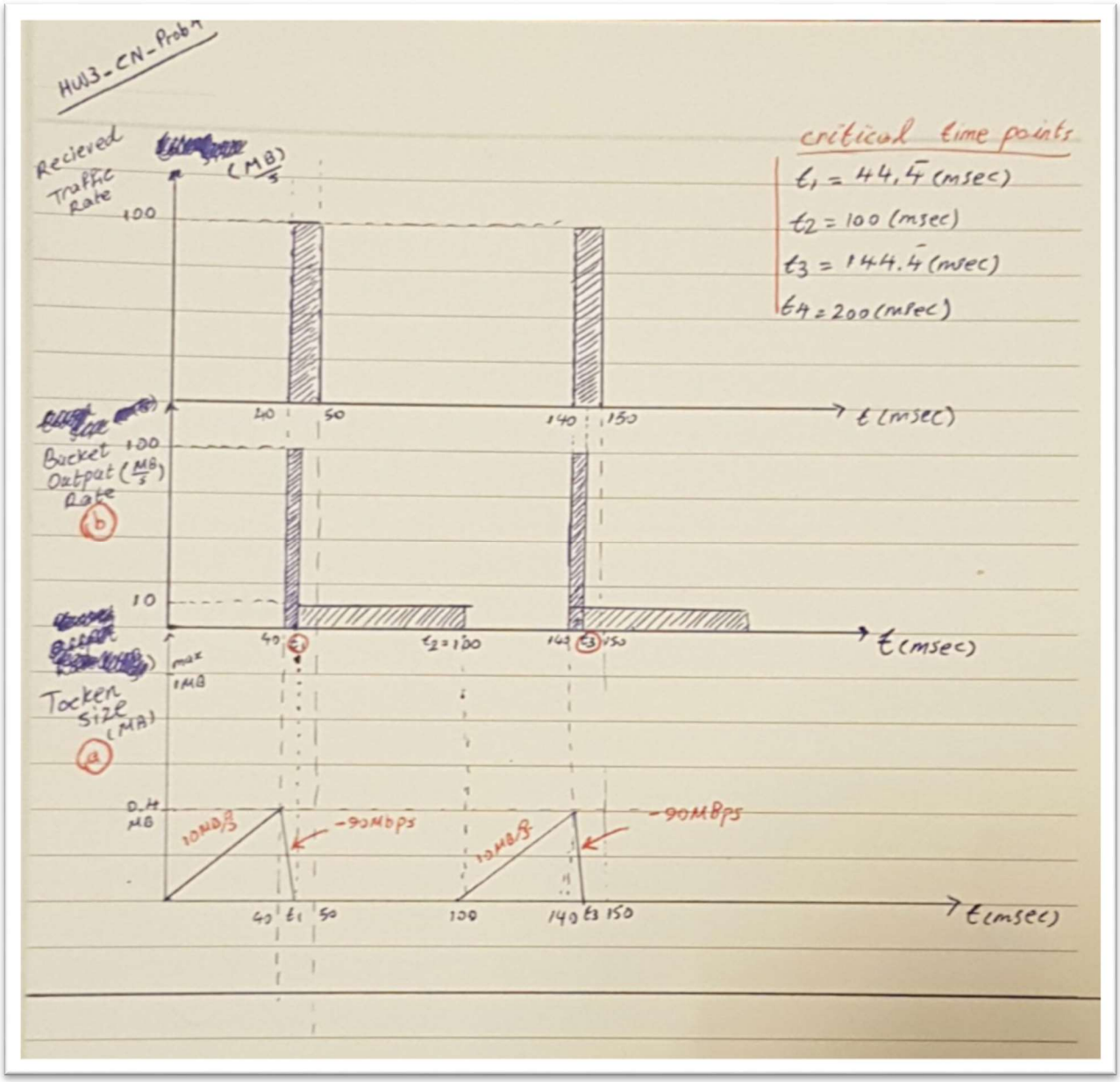



*Figure 4.1 – Token size, Output Rate Graphs for part a,b*

- In the first 40 msec that network layer hasn't received any data from the upper layer, tokens are added to the bucket with a rate of $R_{in}$ = 10MBps. (At (t = 40 msec),when the user has a total token size of 0.4 MB (bucket not full yet), the user's network layer start to receive a 2MB ($20msec \times 100\ MBps$)burst of data.

- The time which the higher data rate (100MBps) can be used for data transfer (the bucket is becoming empty) is derived from the following equation: ($\Delta t_1$)

$$R_{net} \times \Delta t_1 = Current\ Bucket(Token)size = 0.4MB$$
$$R_{net} = R_{out} - R_{in} = 90(MBps)$$
$$\boldsymbol{\Delta t_1 = 4.44(msec)}$$
$$\boldsymbol{t_1 = 40 + \Delta t_1 = 44.44(msec)}$$

- The amount of data expected to be transferred in the interval ($t_1$: 50), will be transferred at a data rate of $R_{out} = R_{in}$ = 10MBps, until all the burst data is fully transferred at (t =$t_2$)

$$R_{burst} \times (50 - t_1) = Rout \times \Delta t_2$$
$$100(MBps) \times (50 - 44.44)(msec) = 10MBps \times \Delta t_2$$
$$\boldsymbol{\Delta t_2 = 55.55\ (msec)}$$
$$\boldsymbol{t_2} = 44.44 + 55.55 = \boldsymbol{100\ (msec)}$$

- After (t = $t_2$), the bucket starts to fill (tokens add up in the bucket because there's nothing to transfer yet). (in the interval (100:140) msec, 0.4 MB token fill's half of the bucket. ($50msec \times 10MBps$)

- Once again at (t = 140 msec), the second burst starts and the token size in the bucket reduces with the rate $R_{net}$. The corresponding time ($t_3$) which the bucket becomes completely empty and transfer rate drops down to $R_{in}$ = 10 MBps, is calculated in the following equations:

$$R_{net} \times \Delta t_3 = Current\ Bucket(Token)size = 0.4MB$$
$$R_{net} = R_{out} - R_{in} = 90(MBps)$$
$$\boldsymbol{\Delta t_3 = 4.44(msec)}$$
$$\boldsymbol{t_3 = 140 + \Delta t_3 = 144.44(msec)}$$

- The amount of data expected to be transferred in the interval ($t_3$: 150), will be transferred at a data rate of $R_{out} = R_{in}$ = 10MBps, until all the burst data is fully transferred at (t =$t_4$)

$$R_{burst} \times (150 - t_1) = Rout \times \Delta t_4$$
$$100(MBps) \times (150 - 144.44)(msec) = 10MBps \times \Delta t_4$$
$$\boldsymbol{\Delta t_4 = 55.55\ (msec)}$$
$$\boldsymbol{t_4} = 144.44 + 55.55 = \boldsymbol{200\ (msec)}$$

## *c)*

- B (bucket capacity) = maximum token size = 1.5MB
- $R_{in}$ (bucket entry rate) = 15MBps
- $R_{burst}$ (received burst data rate from transport layer) = 100 MBps, in periodic bursts, each lasting for 10 msec (40-50,140-150)
- $R_{out}$: The data rate which the network layer transmits following token-bucket.

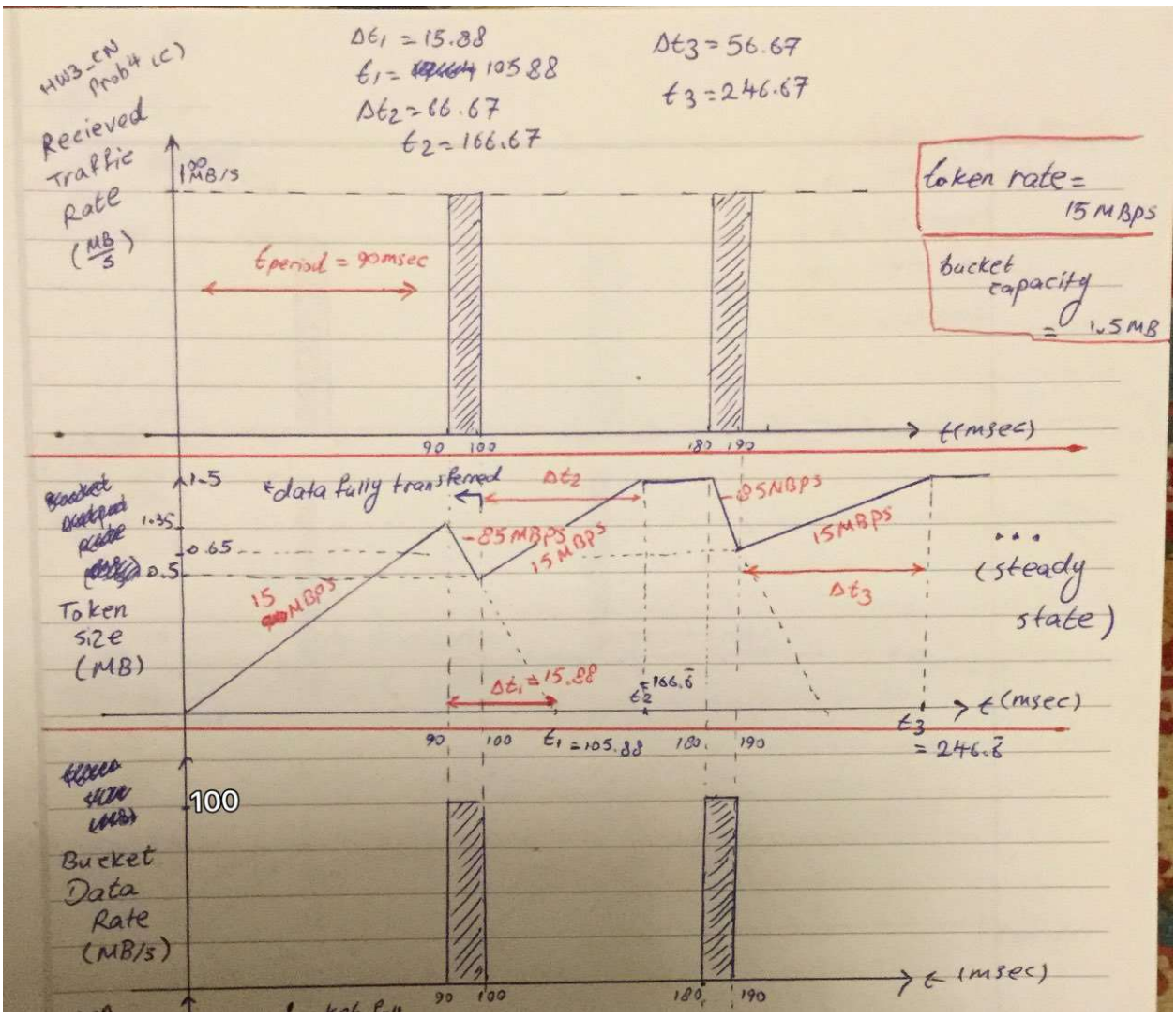



*Figure 4.2 – Token size, Output Rate Graphs for part c*

- First we should figure out whether the bucket reaches max token size (full bucket) or not:

- $R_{out} = R_{in}$ = 15MBps;
- $Bucket\ Capacity = 1.5MB$

$$R_{out} \times 90(msec) = (\boldsymbol{Token\ size\ at\ t = 90}) = 15\ (\text{MBps}) \times 90(msec) = \mathbf{1.35\ MB < 1.5}$$

$$\rightarrow The\ bucket\ reaches\ 90\%\ of\ its\ capacity\ before\ data\ stream\ is\ ready\ for\ transfer$$

- While the 10 msec data burst is received, the network layer will provide a transfer data rate of $R_{net}$ = 100MBps - !5MBps = 85MBps, as the tokens are constantly entering the bucket, Thus the token size will decrease to 0.5MB (whole data is transferred successfully)

$$R_{net} \times (100 - 90) = Token\ size_{t=100} - 1.35$$

$$Token\ size_{t=100} = 1.35 - 85 \times 0.01 = 0.5$$

- After the first data burst, The bucket will be filled with a rate of $R_{out} = R_{in}$ = 15MBps as there is no data that spends the tokens. The bucket will be full at (t = 166.67 msec).
    - $Rout$ = 15MBps

$$(Bucket\ capacity - Token\ size_{t=100}) = Rout \times \Delta t_2$$
$$(1.5 - 0.5)(MB) = 15MBps \times \Delta t_2$$
$$\boldsymbol{\Delta t_2 = 66.67\ (msec)}$$
$$\boldsymbol{t_2} = 100 + 66.67 = \boldsymbol{166.67\ (msec)}$$

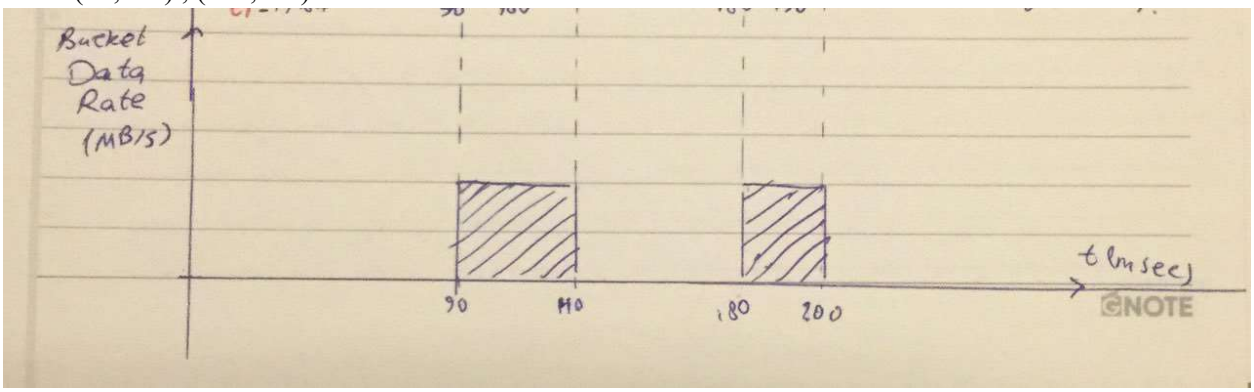
- After (t = $t_2$), The bucket will remain full until the next burst of data arrives at t = 180ms.
- In the second data burst the network layer will provide a transfer data rate of $R_{net}$ = 100MBps - !5MBps = 85MBps, as the tokens are constantly entering the bucket, Thus the token size will decrease to 0.65MB (whole data is transferred successfully)

$$R_{net} \times \Delta t_3 = Current\ Bucket(Token) size = 0.4MB$$
$$R_{net} = R_{out} - R_{in} = 90(MBps)$$
$$\boldsymbol{\Delta t_3 = 56.67(msec)}$$
$$\boldsymbol{t_3 = 190 + \Delta t_3 = 246.67(msec)}$$

- The next time intervals will repeat the pattern of the interval (180: 270) periodically ($t_{period}$ = 90msec)

## *d) Leaky Bucket underneath a Token bucket:*

- Using leaky bucket limits the output data rate to 50MBps. This means each data burst will last twice (90,110) , (180,200)




## Q5) *IP address Corresponding to a port:*

- Subnet Masks are used to separate IP addresses in and out a LAN (subnet). The bits of the IP address which their corresponding subnet mask bit is in the second can be different within a LAN (subnet). But the bits which their corresponding subnet mask bit is in the first part are the same within LAN (subnet) devices.

| Subnet Mask (decimal) | Subnet Mask (binary) | Forwarding Port | IP address range |
|---|---|---|---|
| 172.10.0.0/20 | 1010 1100. 0000 1010.0000 0000.0000 0000 | Part 1 | 172.10.0.0 – 172.10.15.0 |
| 172.10.16.0/19 | 1010 1100. 0000 1010.0001 0000.0000 0000 | Port 2 | 172.10.16.0 – 172.10.31.255 |
| 172.10.32.0/22 | 1010 1100. 0000 1010.0010 0000.0000 0000 | Port 3 | 172.10.32.0 – 172.10.35.255 |