Computer-Networks-Leaky-Bucket-Tips

What is leaky bucket algorithm?

The Leaky Bucket Algorithm is a method used to control the rate at which data is sent over a network. It's designed to prevent data overload on the network. Imagine a bucket with a hole in the bottom that leaks out water at a constant rate. Here's how it works:

- The bucket has a specific size, which represents the maximum amount of data that can be buffered at any given time.
- Data entering the network is likened to pouring water into the bucket.

- The outgoing data rate is equivalent to the leak at the bottom of the bucket. Data continuously flows out of the bucket at this rate.

Basic Algorithm for Remembering

1. Read Bucket size, Outgoing rate, number of packets
2. Loop through each packet, Loop until number of packet = 0
    1. Read the incoming packet size
    2. Check if the incoming packet can be accomodated in the buffer
        1. If possible, then add the incoming packet to `store` variable
        2. else set the store = bucket size
    3. Simulate the outgoing data by store = store - outgoing
    4. Check if store variable is less than 0
        1. If its less than 0 then set it to 0 so it doesnt go negative
    5. Decrement number of packets by 1

Algorithm in detail

1. Read Bucket size, Outgoing rate, number of packets

```
    int incoming, outgoing, buck_size, n, store = 0; // Declare variables for incoming packet size, outgoing rate, bucket size, number of packets, and current buffer size

    // Prompt the user to enter bucket size, outgoing rate, and number of packets
    printf("Enter the bucket size (Bytes): ");
    scanf("%d", &buck_size);

    printf("Enter the outgoing rate (Bytes per second): ");
    scanf("%d", &outgoing);

    printf("Enter the number of packets: ");
    scanf("%d", &n);
    printf("-----------\n"); // Divider for clarity
```

2. Loop through each packet

```
while(n != 0) {
```

2.1 Read the incoming packet size

```
printf("Enter the incoming packet size (Bytes): ");
scanf("%d", &incoming);
```

2.2 Check if the incoming packet can be accomodated in the buffer

```
if(incoming <= (buck_size - store)) {
            // If yes, add the packet to the buffer
            store += incoming;
            printf("Current Bucket buffer size %d out of %d\n", store,
buck_size);
        } else {
            // If not, drop the excess data and update buffer size
            printf("Dropped %d bytes of data from Packet\n", incoming -
(buck_size - store));
            printf("Current Bucket buffer size %d out of %d\n", buck_size -
store, buck_size);
            store = buck_size;
        }
```

- Here incoming = incoming packet size
- Store variable is initially zero, after that it is filled step by step until the bucket is full
- Buck_size is the bucket size
- If the capacity is not enough
    - Excess data is removed and the store will be equal to bucket size

2.3 Simulate outgoing data

Store is decremented by the outgoing variable

```
store = store - outgoing;
```

2.4 Negative value check and decrementing packet size

```
// Ensure buffer size doesn't go negative
if (store < 0) {
        store = 0;
}

// Display the remaining buffer size after outgoing data
printf("After outgoing %d bytes left out of %d in buffer\n", store,
buck_size);
printf("----------------------------\n"); // Divider for clarity
n--; // Decrement the number of packets
```

2.5 Ending the loop

```
}

return 0;
```