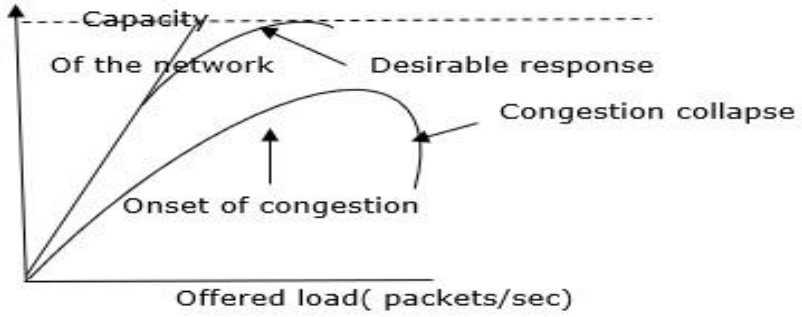
EXPERIMENT 7

OBJECTIVE:

Implement Leaky Bucket Algorithm:

When too many packets are present in the network it causes packet delay and loss of packet which degrades the performance of the system. This situation is called congestion.

The **network layer** and **transport layer** share the responsibility for handling congestions. One of the most effective ways to control congestion is trying to reduce the load that transport layer is placing on the network. To maintain this, the network and transport layers have to work together.

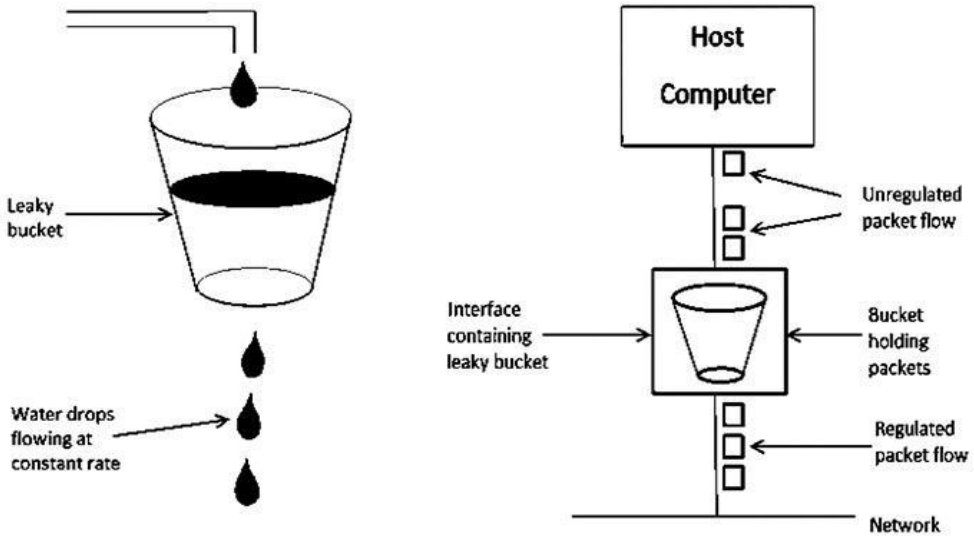


With too much traffic, performance drops sharply. There are two types of Congestion control algorithms, which are as follows −

1. Leaky Bucket Algorithm
2. Token Bucket Algorithm

Leaky Bucket Algorithm

Let see the working condition of Leaky Bucket Algorithm – Leaky Bucket Algorithm mainly controls the total amount and the rate of the traffic sent to the network.



Leaky Bucket Algorithm mainly controls the total amount and the rate of the traffic sent to the network.

Step 1 − Let us imagine a bucket with a small hole at the bottom where the rate at which water is poured into the bucket is not constant and can vary but it leaks from the bucket at a constant rate.

Step 2 − So (up to water is present in the bucket), the rate at which the water leaks does not depend on the rate at which the water is input to the bucket.

Step 3 − If the bucket is full, additional water that enters into the bucket that spills over the sides and is lost.

Step 4 − Thus the same concept applied to packets in the network. Consider that data is coming from the source at variable speeds. Suppose that a source sends data at 10 Mbps for 4 seconds. Then there is no data for 3 seconds. The source again transmits data at a rate of 8 Mbps for 2 seconds. Thus, in a time span of 8 seconds, 68 Mb data has been transmitted.

That's why if a leaky bucket algorithm is used, the data flow would be 8 Mbps for 9 seconds. Thus, the constant flow is maintained.

Here's a step-by-step explanation of the Leaky Bucket algorithm with output at each step:

Step 1: Initialization

Initialize the bucket size (BS) to 10.

Initialize the leak rate (LR) to 1.

Initialize the current bucket level (BL) to 0.

Initialize the packet size (PS) to 4.

Initialize the number of packets (NP) to 4.

Output:

Bucket Size (BS): 10

Leak Rate (LR): 1

Current Bucket Level (BL): 0

Packet Size (PS): 4

Number of Packets (NP): 4

Step 2: First Packet Arrival

- A packet of size 4 arrives.
- Check if the packet can be accommodated in the bucket: BL + PS <= BS (0 + 4 <= 10).
- Since the packet can be accommodated, update the bucket level: BL = BL + PS (0 + 4 = 4).
- Print the current bucket level and the bucket size.

Output:

Buffer size= 4 out of bucket size= 10

Step 3: Leaking

- Leak packets from the bucket at a rate of 1 packet per time unit.
- Update the bucket level: BL = BL - LR (4 - 1 = 3).
- Since the bucket level cannot be negative, set it to 0 if it is less than 0.

Output:

Bucket level after leaking: 3

Step 4: Second Packet Arrival

- A packet of size 4 arrives.
- Check if the packet can be accommodated in the bucket: BL + PS <= BS (3 + 4 <= 10).
- Since the packet can be accommodated, update the bucket level: BL = BL + PS (3 + 4 = 7).
- Print the current bucket level and the bucket size.

Output:

Buffer size= 7 out of bucket size= 10

Step 5: Leaking

- Leak packets from the bucket at a rate of 1 packet per time unit.
- Update the bucket level: BL = BL - LR (7 - 1 = 6).
- Since the bucket level cannot be negative, set it to 0 if it is less than 0.

Output:

Bucket level after leaking: 6

Step 6: Third Packet Arrival

- A packet of size 4 arrives.
- Check if the packet can be accommodated in the bucket: BL + PS <= BS (6 + 4 <= 10).
- Since the packet can be accommodated, update the bucket level: BL = BL + PS (6 + 4 = 10).
- Print the current bucket level and the bucket size.

Output:

Buffer size= 10 out of bucket size= 10

Step 7: Leaking

- Leak packets from the bucket at a rate of 1 packet per time unit.
- Update the bucket level: BL = BL - LR (10 - 1 = 9).
- Since the bucket level cannot be negative, set it to 0 if it is less than 0.

Output:

Bucket level after leaking: 9

Step 8: Fourth Packet Arrival

- A packet of size 4 arrives.
- Check if the packet can be accommodated in the bucket: BL + PS <= BS (9 + 4 <= 10).
- Since the packet cannot be accommodated, print a packet loss message.

Output:

Packet loss = 4

Step 9: Leaking

- Leak packets from the bucket at a rate of 1 packet per time unit.
- Update the bucket level: BL = BL - LR (9 - 1 = 8).
- Since the bucket level cannot be negative, set it to 0 if it is less than 0.

Output:

Bucket level after leaking: 8

The final output of the Leaky Bucket algorithm is:

Buffer size= 4 out of bucket size= 10

Buffer size= 7 out of bucket size= 10

Buffer size= 10 out of bucket size= 10

Packet loss = 4

Buffer size= 9 out of bucket size= 10

This output shows the buffer size and bucket size after each packet arrival and leaking. The Leaky Bucket algorithm has finished processing all packets, and the final buffer size is 8, which is less than the bucket size of 10. The algorithm has successfully policed the network traffic by controlling the rate at which packets are sent.

```
#include <stdio.h>
#include <stdlib.h>

#define BUCKET_SIZE 30
#define OUTGOING_RATE 3

int main() {
int incoming, bucket_content = 0, n, i;

printf("Enter the number of packets: ");
scanf("%d", &n);

for(i = 0; i < n; i++) {
printf("Enter the number of incoming packets at time %d: ", i + 1);
scanf("%d", &incoming);

if(incoming + bucket_content> BUCKET_SIZE) {
printf("Bucket overflow! Dropping %d packets.\n", (incoming + bucket_content) - BUCKET_SIZE);
bucket_content = BUCKET_SIZE;
        } else {
bucket_content += incoming;
        }

printf("Packets in bucket: %d\n", bucket_content);
bucket_content -= OUTGOING_RATE;
if(bucket_content< 0) {
bucket_content = 0;
        }
printf("After sending %d packets, remaining in bucket: %d\n", OUTGOING_RATE, bucket_content);
    }

    // Empty the bucket at the end
while(bucket_content> 0) {
printf("After sending %d packets, remaining in bucket: %d\n", OUTGOING_RATE, bucket_content);
bucket_content -= OUTGOING_RATE;
if(bucket_content< 0) {
bucket_content = 0;
        }
    }

printf("Bucket is now empty.\n");
return 0;
}
```

OUTPUT

```
Enter the number of packets: 10
Enter the number of incoming packets at time 1: 12
Packets in bucket: 12
After sending 3 packets, remaining in bucket: 9
Enter the number of incoming packets at time 2: 12
Packets in bucket: 21
After sending 3 packets, remaining in bucket: 18
Enter the number of incoming packets at time 3: 00
Packets in bucket: 18
After sending 3 packets, remaining in bucket: 15
Enter the number of incoming packets at time 4:
0
Packets in bucket: 15
After sending 3 packets, remaining in bucket: 12
Enter the number of incoming packets at time 5: 0
Packets in bucket: 12
After sending 3 packets, remaining in bucket: 9
Enter the number of incoming packets at time 6: 0
Packets in bucket: 9
After sending 3 packets, remaining in bucket: 6
Enter the number of incoming packets at time 7: 0
Packets in bucket: 6
After sending 3 packets, remaining in bucket: 3
Enter the number of incoming packets at time 8: 2
Packets in bucket: 5
After sending 3 packets, remaining in bucket: 2
Enter the number of incoming packets at time 9: 2
Packets in bucket: 4
After sending 3 packets, remaining in bucket: 1
Enter the number of incoming packets at time 10: 2
Packets in bucket: 3
After sending 3 packets, remaining in bucket: 0
Bucket is now empty.
```