Congestion control algorithm using Leaky Bucket Algorithm

Problem Statement

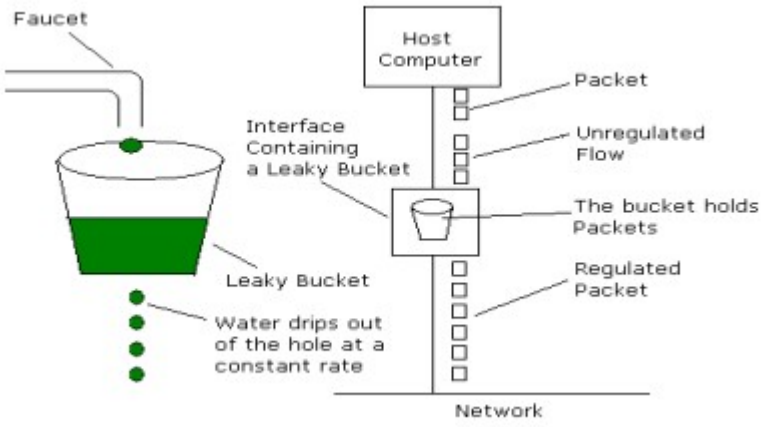
Write a program for congestion control using Leaky bucket algorithm.

Theory

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination. In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back. The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer. Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called traffic shaping. The other method is the leaky bucket algorithm.

Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.



Algorithm:

1. Start
2. Set the bucket size or the buffer size.
3. Set the output rate.
4. Transmit the packets such that there is no overflow.
5. Repeat the process of transmission until all packets are transmitted. (Reject packets where its size is greater than the bucket size)
6. Stop

Program

```
#include<stdio.h>
int main(){
   int incoming, outgoing, buck_size, n, store = 0;
   printf("Enter bucket size, outgoing rate and no of inputs: ");
   scanf("%d %d %d", &buck_size, &outgoing, &n);

   while (n != 0) {
      printf("Enter the incoming packet size : ");
      scanf("%d", &incoming);
      printf("Incoming packet size %d\n", incoming);
      if (incoming <= (buck_size - store)){
         store += incoming;
         printf("Bucket buffer size %d out of %d\n", store, buck_size);
      } else {
         printf("Dropped %d no of packets\n", incoming - (buck_size - store));
         printf("Bucket buffer size %d out of %d\n", store, buck_size);
         store = buck_size;
      }
      store = store - outgoing;
      printf("After outgoing %d packets left out of %d in buffer\n", store, buck_size);
```

```
        n--;
    }
}
```

Output

```
Enter bucket size, outgoing rate and no of inputs: 50 100 3
Enter the incoming packet size : 50
Incoming packet size 50
Bucket buffer size 50 out of 50
After outgoing -50 packets left out of 50 in buffer
Enter the incoming packet size : 100
Incoming packet size 100
Bucket buffer size 50 out of 50
After outgoing -50 packets left out of 50 in buffer
Enter the incoming packet size : 20
Incoming packet size 20
Bucket buffer size -30 out of 50
After outgoing -130 packets left out of 50 in buffer
```