Leaky bucket

What is a Leaky bucket?

The **leaky bucket** is an algorithm used to control data flow in networks. It allows data packets to pass through at a steady, controlled rate, similar to water leaking slowly from a bucket with a small hole. If data comes in too fast, the bucket "overflows," meaning some data is discarded to prevent network congestion. This ensures a smooth and manageable data transmission.

Explain difference between Leaky Bucket and Token Bucket:

The **Leaky Bucket** and **Token Bucket** are both algorithms used for network traffic management, but they work in different ways:

1. **Leaky Bucket Algorithm**:
    - Data flows out of the bucket (network) at a fixed rate, like water leaking from a bucket with a hole.
    - If incoming data packets exceed the bucket's capacity, the excess packets are discarded, preventing overflow.
    - This algorithm enforces a strict output rate, so data flow remains steady and controlled.
2. **Token Bucket Algorithm**:
    - Tokens (permissions to send data) are generated at a fixed rate and stored in the bucket.
    - Each data packet requires a token to be transmitted, so packets are only sent if tokens are available.
    - If there are enough tokens, the algorithm allows bursts of data, making it more flexible and able to handle sudden surges in traffic without dropping packets, as long as tokens are available.

```
# initial packets in the bucket
storage = 0
```

This line initializes the `storage` variable to `0`, representing the current number of packets in the "bucket" or buffer.

```
# total no. of times bucket content is checked
no_of_queries = 4
```

This variable, `no_of_queries`, defines the total number of times we'll simulate packet arrival and processing. Each iteration will represent a cycle where packets attempt to enter the bucket.

```
# total no. of packets that can be accommodated in the bucket
bucket_size = 10
```

`bucket_size` sets the maximum number of packets the bucket can hold at any given time. If the number of packets exceeds this size, packet loss occurs.

```
# no. of packets that enters the bucket at a time
input_pkt_size = 4
```

`input_pkt_size` represents the number of packets arriving to enter the bucket in each cycle. Here, 4 packets try to enter each time the bucket is checked.

```
# no. of packets that exits the bucket at a timeout
output_pkt_size = 1
```

`output_pkt_size` specifies the number of packets that are processed or "leak" out of the bucket in each cycle. This simulates the rate at which the bucket releases packets.

```
for i in range(0, no_of_queries):
```

This line starts a loop that will run `no_of_queries` times, simulating each time step in which packets try to enter and exit the bucket.

```
   # space left
    size_left = bucket_size - storage
```

This calculates the remaining space in the bucket by subtracting the current `storage` (the number of packets in the bucket) from the `bucket_size`. `size_left` tells us how many packets can still fit in the bucket without causing overflow.

```
   if input_pkt_size <= size_left:
        # update storage
        storage += input_pkt_size
```

This checks if the incoming packets (`input_pkt_size`) can fit within the remaining space (`size_left`). If they can, they are added to `storage`, increasing the packet count in the bucket.

```
   else:
        print("Packet loss =", input_pkt_size)
```

If the incoming packets (`input_pkt_size`) exceed the available space (`size_left`), this `else` block is executed, and it prints a message indicating packet loss, as the incoming packets cannot be accommodated.

```
   print(f"Buffer size= {storage} out of bucket size = {bucket_size}")
```

This line prints the current state of the bucket, showing how many packets (`storage`) are currently in the bucket compared to its maximum capacity (`bucket_size`).

```
   # as packets are sent out into the network, the size of the storage
decreases
    storage -= output_pkt_size
```

Finally, this line decreases `storage` by `output_pkt_size`, simulating the process of packets being sent out or "leaking" from the bucket. This represents the regular rate at which packets exit the bucket in each cycle.

Summary of Code Behavior

In each cycle:

1. Incoming packets attempt to enter the bucket.
2. If there's enough space, they're added to the bucket. If not, packet loss occurs.
3. The bucket's contents are then reduced by a fixed amount, simulating packet exit.
4. The state of the bucket is printed, showing the current buffer size.

This process repeats for `no_of_queries` cycles, providing a simple simulation of the leaky bucket algorithm.

4o