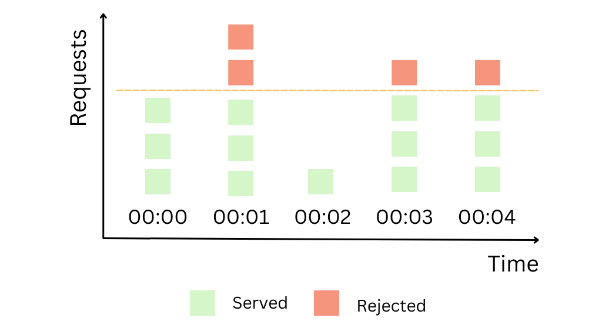
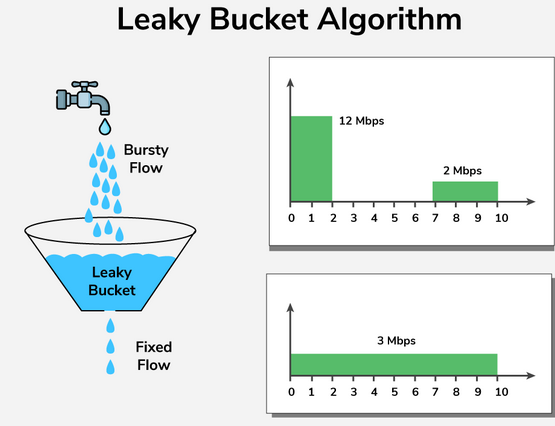
Question 4

1. Rate limiting: Controlling frequency of requests to prevent spams:
   1. Fixed Window Rate Limiter - Fixed Window Counter rate limiting operates by dividing time into fixed-size windows and tracking the number of requests from each user or client within each window. If a user exceeds the maximum allowed number of requests for a window, subsequent requests are blocked until the next window starts. This is the simplest approach.



Approach to build:

* Creates a defaultdict (hash map) of each new user and a request counter
* Current time of request is converted into a window number (time based, eg: window size = 1 second)
* For a new user at a new time, window counter is reset
* If over the limit for all users, block any incoming request
  1. Leaky Bucket Rate Limiter – The approach of using a leaky bucket is where the bucket size is constant and has a leak that allows it to shrink in size progressively. New incoming requests are accumulated into the bucket, and if this one is full, requests are rejected.
  2. 

Approach to build:

* Each user gets a water level (threshold)
* A Leak\_rate (defined in the problem statement) determines how much water has passed using requests
* If below 0, block request
* Water level resets with every time frame (5 seconds)
* Alternatively, if water is added (new request) and a threshold is reached, prevent any further requests.

1. Demonstrated in efficient\_rate\_limiter.py

Efficient data structure used – defaultdict

1. Tradeoffs:

Fixed window:

Simple and efficient, but causes traffic spikes

Low memory footprint, but sharp counter reset at window boundary

Good for MVP, but does not consider previous window requests

Leaky bucket:

Works great for traffic, but relies on timestamp (can be event time or window time) and must be configured

Natural recovery, but less precise

Better for backend servers, but time drift may reduce accuracy