Lab 5: QoS Algorithms Leaky vs Token Bucket Using Server-Client Architecture

Objective:

To implement a video streaming system using server-client architecture and analyze the Impact of QoS algorithms (Leaky Bucket and Token Bucket) on streaming performance, especially under varying network conditions.

Problem Statement:

In real-time applications such as video conferencing and live streaming, delivering continuous, high-quality video under varying network conditions is a significant challenge. Efficient video compression and transmission must be balanced with the need for low latency and minimal packet loss. This lab aims to simulate and analyze a complete video streaming pipeline using a server-client model, integrating key Quality of Service (QoS) techniques. Students will implement both Leaky Bucket and Token Bucket algorithms to control data flow and assess their effectiveness in maintaining stream performance. Through hands-on experimentation, the lab encourages a deeper understanding of how transport protocols, encoding settings, and traffic shaping algorithms impact real-time media delivery.

Part 1: Build the Video Streaming System

Task 1: Server Setup

- Load a video file (`sample.mp4` or raw YUV)
- Use a real-time encode (e.g., ffmpeg, OpenCV, or GStreamer) with a codec like **H.264, VP9, or H.265**.
- Stream the encoded video over **TCP or UDP**.
- Format the stream (e.g., **MPEG-TS**, **WebRTC**, or **raw frames**) for network transmission.

Task 2: Client Setup

- Connect to the server using the same protocol.
- Decode the video stream.
- Display the video using a rendering tool (`OpenCV`, SDL, or GStreamer).

Task 3: Initial Testing

- Confirm client-server connectivity.
- Check real-time playback and measure initial latency.

Part 2: Implement QoS Algorithms

Task 4: Leaky Bucket Algorithm

Server (leaky_bucket_server.py)

- Accept packets from the client.
- Leak packets at a constant rate.
- Drop packets if the bucket is full.

Client (leaky_bucket_client.py)

- Send variable-size packets at random intervals.
- Display server responses (processed/discarded).

Task 5: Token Bucket Algorithm

Server (token_bucket_server.py)

- Maintain a refillable token bucket.
- Allow bursty packet handling if tokens are available.
- Reject packets when tokens are insufficient.

Client (token_bucket_client.py)

- Send requests simulating variable load.
- Log server responses.

Part 3: QoS Integration for Streaming

Task 6: Integrate QoS into Streaming System

- Modify the video streaming server-client to use either the **Leaky Bucket** or **Token Bucket** for traffic shaping.
- Apply the algorithm to manage frame delivery rates.
- Test and compare:
    - Stream consistency under bursty network conditions.
    - Responsiveness and latency.
    - Packet/frame loss behavior.

Experimentation & Analysis

1. Protocol Behavior (TCP vs. UDP)

- Compare reliability, latency, and stream smoothness.
- Discuss scenarios best suited for each.

2. Latency Analysis

- Measure end-to-end delay.
- Identify key contributors: network, buffer, encoding/decoding.

3. Encoding Efficiency

- Test various codecs (H.264, VP9, H.265).
- Vary bitrate, resolution, and framerate.
- Log CPU usage, quality, and bandwidth.

4. QoS Algorithm Evaluation

- Simulate packet loss and bursts.
- Observe how Leaky vs. Token Bucket affects playback.
- Identify which works better for:
    - Real-time conferencing (low-latency).
    - Buffered video streaming (high-quality).

Submission Format:

- Package all deliverables into a single compressed file named `E19XXX_Lab05.zip`, where `<XXX>` is your e no.
- Include the following:
  - `video_server.py`, `video_client.py:` Original streaming setup.
  - `leaky_bucket_server.py, leaky_bucket_client.py`: QoS - Leaky Bucket.
  - `token_bucket_server.py, token_bucket_client.py`: QoS - Token Bucket.
  - `video_qos_server.py, video_qos_client.py`: Streaming system integrated with the best QoS mechanism.
  - `E19XXX_Lab05.pdf`: Detailed explanation and analysis.
  - `README.md: Run instructions and dependencies (e.g., opencv-python, ffmpeg-python, etc.).`
  - **Media Evidence**: Screenshots or screen recordings demonstrating playback.

Assessment Criteria:

1. **Correctness**: The server and client scripts should perform as per the described functionality.
2. **Robustness**: The implementation should handle edge cases (e.g., invalid packet sizes, client disconnects).
3. **Code Quality**: The code should be well-documented and use proper naming conventions.
4. **Output**: The output should clearly show the flow of requests, processing, and responses.

Notes:

- Use threading for simulating the refill and leak processes.
- Test the server and client scripts locally using `localhost` and appropriate ports.
- Use random intervals and sizes for client requests to simulate real-world scenarios.

Bonus:

- Figure out which works best for real-time streaming vs video streaming
- Update previous lab server clients to incorporate the best-matching QoS

Resources:

- Purdue raw YUV test videos: https://engineering.purdue.edu/~reibman/ece634/Videos/video_files.htm
- https://www.geeksforgeeks.org/leaky-bucket-algorithm/
- https://www.geeksforgeeks.org/token-bucket-algorithm/