A Weighted Fair Queueing (WFQ) Server

1 Introduction

Weighted Fair Queueing (WFQ) is an advanced packet scheduling algorithm designed to allocate bandwidth equitably among multiple traffic flows based on assigned weights. The server is non-preemptive and work-conserving. In this assignment, you will implement a WFQ-based server using UDP socket programming in Python.

The server will manage multiple client flows, each identified by a unique UDP client port number, and schedule packets according to their designated flow weights. Additionally, it will enforce a buffer policy to handle congestion effectively. Key parameters such as server capacity, buffer size, and flow weights will be configured to ensure fair resource allocation.

A client script is provided to generate traffic for the server. Your implementation must accurately handle incoming packets from multiple clients (operating on port numbers 5001, 5002, and 5003), compute their virtual finish times, and queue or serve them based on these times. By the end of this assignment, you will have developed a functional WFQ-based server capable of distributing bandwidth fairly across diverse client flows.

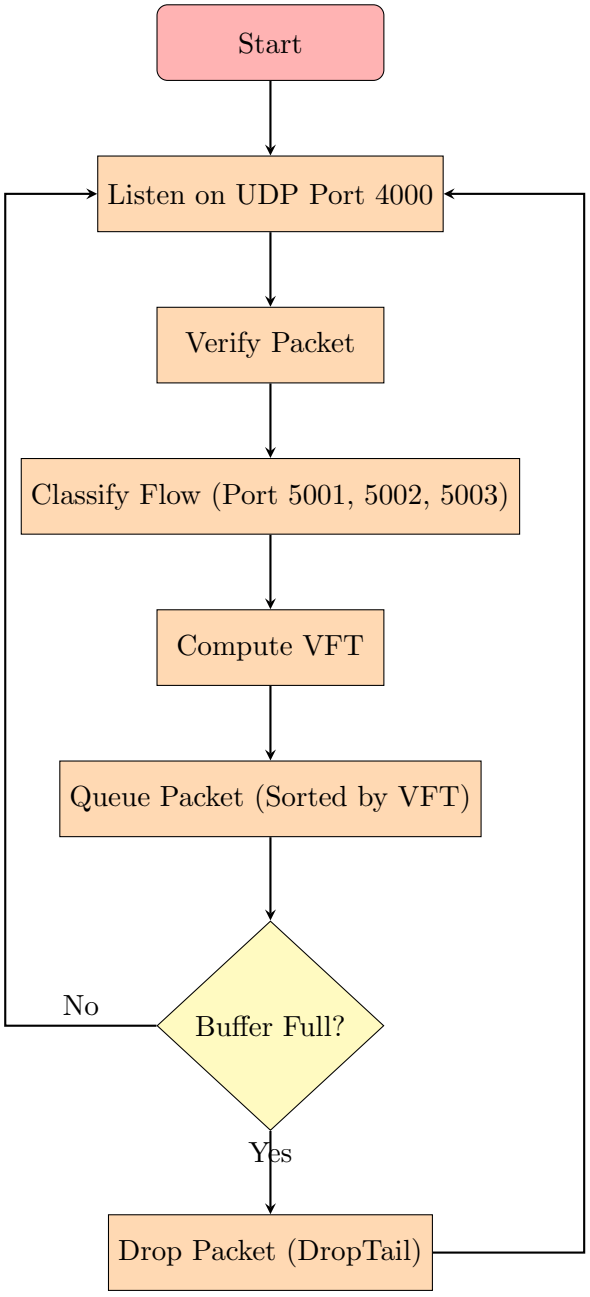
1.1 System Setup

The system consists of a WFQ server and multiple clients (max three). Each client sends packets to the server at regular intervals, and the server processes them using a WFQ server with a capacity of C packets per second, flow weights $\{W_i\}$, and a buffer size of B packets.

2 System Requirements

The goal is to implement a WFQ server that processes packets from multiple clients fairly based on their assigned weights. The server should do the following:

- **UDP Socket Communication and Flow Identification**: Listen on a specified UDP port (4000) for incoming packets from clients. Identify packets of different flows based on the client UDP port numbers (5001, 5002, 5003).



- **Virtual Finish Time**: Assign a Virtual Finish Time (VFT) to each packet based on its flow's weight and the system's virtual time:

$$VFT_i = \max(VT, VFT_{last}) + \frac{1}{C \times W_i}$$

where

  - VFT_i is the virtual finish time of the incoming packet.
  - VT is the current system virtual time.
  - VFT_{last} is the last virtual finish time of a packet currently in the system for the corresponding flow.
  - C is the server processing capacity (packets per second).
  - W_i is the weight assigned to the flow.

- **Buffer and Queue**: Queue the packet in the buffer based on its virtual finish time (packets are arranged in increasing order of VFT in a single queue).
- **Enforce a Buffer Policy**: If the queue exceeds the buffer size (B packets), drop the packet with the highest VFT (DropTail).
- **Packet Serving**: Serve packets at a given capacity (e.g., C packets per second) in order of their VFT, ensuring fairness across flows, and echo them back to the respective clients.
- **Virtual Time Management**: Update the system's virtual time based on the VFT of the packet currently being served. If no packet is being served, reset the virtual time to the arrival time of the next packet.
- The server should function correctly with the provided client script, which sends packets at regular intervals and computes throughput metrics based on the echoed packets.

2.1 Configuration

The server should be configurable with the following parameters:

- **Capacity**: Maximum packets processed per second.
- **Buffer Size**: Maximum number of packets in the queue.
- **Flow Weights**: A function mapping client port numbers to their respective weights.

2.2 Client Script

A client script is provided to test the server. The client:

- Sends packets to the server at regular intervals.
- Receives echoed packets from the server.
- Computes and prints long-term and short-term throughput.

3 Submission Instructions

Students must submit:

- A Python script implementing the WFQ server.
- A short one-page (PDF) report discussing the implementation details, and how to configure and run the server. The report shall include a table reporting the performance in a variety of traffic/network scenarios.

Table 1: WFQ Server Configuration

Capacity C (in Pkts/sec)	Flow Weights $\{W\}$	Buffer Size B (in Pkts)	Client Rate (in Pkts/sec)	Flow Throughput(s) (in Pkts/sec)
10	1:1:1	10	10:10:10	
10	1:1:1	10	10:20:40	
10	1:2:4	10	10:10:10	
20	1:2:4	100	10:10:10	

4 Reference

Section 4.3: Scheduling from Chapter 4: Stream Sessions: Deterministic Network Analysis from *Communication Networking: An Analytical Approach*, Anurag Kumar, D Manjunath and Joy Kuri, Morgan Kaufmann Publishers.