CS621 Network Programming - Spring 2023 - Project 2 Specification
A Base Class to Simulate Differentiated Services

Deadline: May 12th, 2023 at 11:59PM

To be done individually or in teams of up to three members.

Project Outcomes

1. Work effectively with a network simulator.
2. Implement a base class for differentiated services in ns-3.
3. Implement Strict Priority Queueing (SPQ) and Deficit Round Robin (DRR), using the base class.
4. Validate and verify your SPQ and DRR implementations.

Overview

The purpose of this project is to implement a selection of differentiated services. Because there is a large amount of shared structure amongst differentiated services, it is a practical exercise to start by creating a base class. The design for the base class is provided to you. Your assignment is to implement it.

Upon successfully implementing the base class, you will implement two quality-of-service (QoS) mechanisms: SPQ and DRR. Refer to lectures on QoS and the corresponding reading assignments for a detailed discussion of these services. To conclude the project, you will validate and verify your implemented services. This will require creating and running various ns-3 simulation runs, which you may want to write scripts to automate them.

Components

(1) DiffServ: A Base Class to Simulate Differentiated Services

DiffServ class provides basic functionalities required to simulate differentiated services:

- Classification - The classify() function utilizes the filter aspect to sort the traffic packets into appropriate traffic queues.
- Scheduling - The schedule() function follows the designed QoS algorithm to schedule which traffic queue to be served at the time.

For network queues, you may override DoEnqueue() and DoDequeue() functions to meet implementation requirements for various QoS algorithms.

Your base class will be a subclass of `ns-3::Queue`, as such you should start this project out by learning in detail the `ns-3::Queue` API reference and the closely related `ns-3::DropTailQueue` API reference.

Your base class should follow the requirements in the UML diagram in Figure 1. Here is a short description detailing the role of some of the classes in the figure and their members:

- DiffServ
  - m_mode - The QueueMode specifies whether service is in byte mode or packet mode.
  - q_class - The vector (array) of TrafficClass pointers. setMode()/getMode() - the accessor and modifier for private variable m_mode.
  - Schedule() - Returns a packet to transmit.

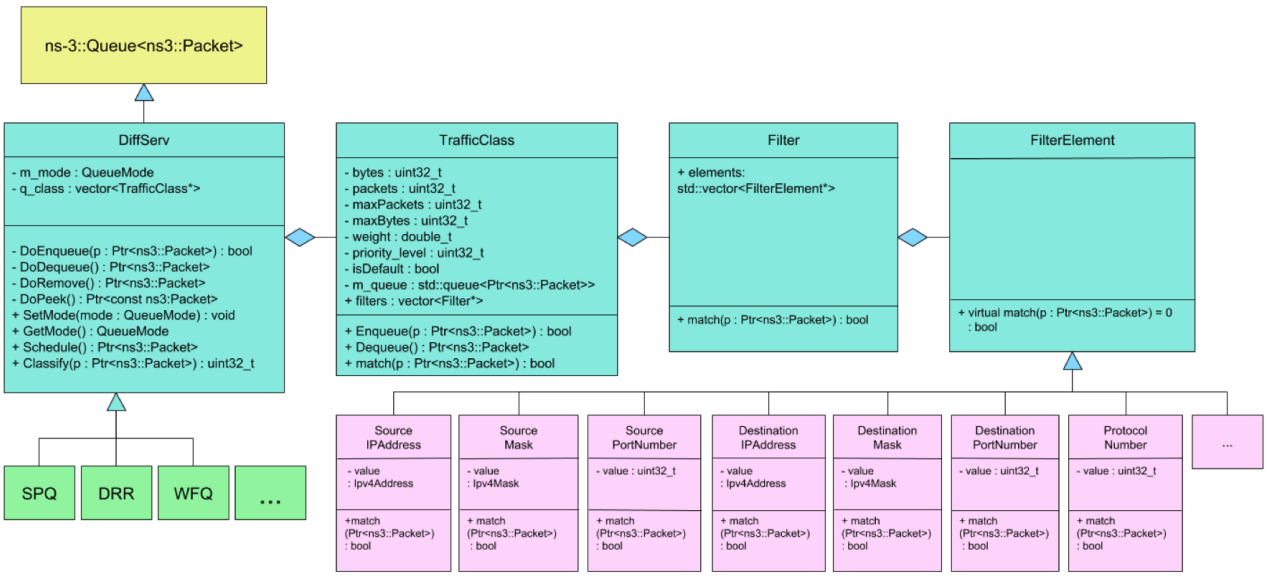

Figure 1: Base Class Design

  - Classify(p) - Takes a packet and returns an integer.

- TrafficClass - Defines the characteristics of a queue.
  - bytes - the byte count in the queue.
  - packets - the packet count in the queue.
  - maxPackets - the maximum number of packets allowed in the queue.
  - maxBytes - the maximum number of bytes allowed in the queue.
  - weight - applicable if the QoS mechanism uses weights.
  - priority_level - applicable if the QoS mechanism uses priority levels.
  - filters - A collection of conditions as Filters, any of which should match in order to trigger match.
- Filter - A collection of conditions as FilterElements, all of which should match in order to trigger match. Refer to Figure 2 for an example.
  - elements - the array of pointers to FilterElement.
- FilterElement - a base class for a primitive condition to match. You should write one subclass for every seven boxes in the design diagram.

(2) Implementing SPQ and DRR with DiffServ

Using your implemented DiffServ class, your task is to implement two QoS mechanisms: Strict Priority Queueing (SPQ) and Deficit Round Robin (DRR). Either implementation take the corresponding configuration filename as a CLI input:

- SPQ's configuration file includes the number of queues and their respective priority levels.
- DRR's configuration file includes the number of queues and the corresponding quantum values for each queue.

Extra Credit: Implement a parser for the configuration file for SPQ so that the configuration file is a sequence of CLI commands for Cisco Catalyst 3750 switch.

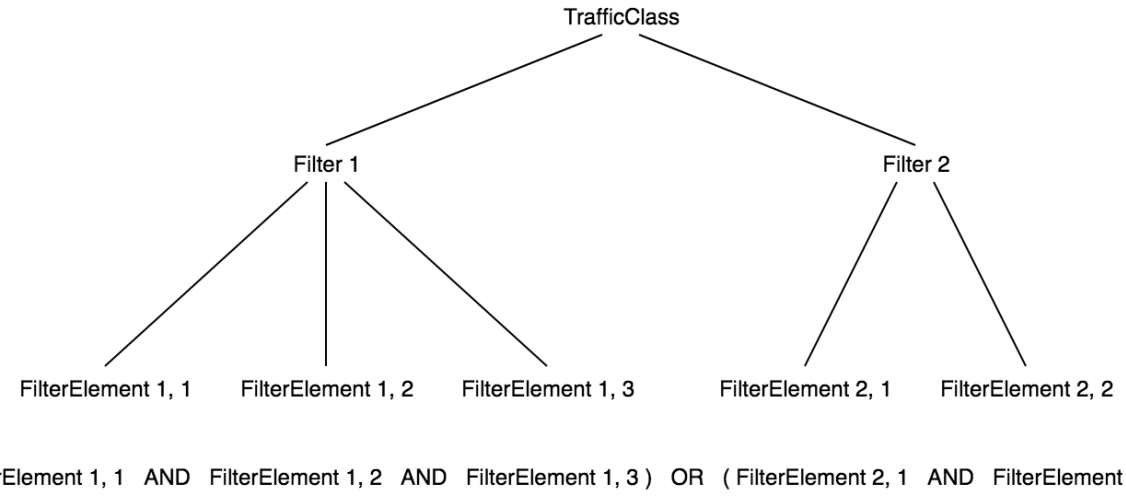

Figure 2: An example constructing complex boolean expressions using the DiffServ class.

(3) Simulation Verification and Validation

As the last component of this project, you are to create ns-3 simulation runs to demonstrate the correctness of your implementations above.

For both SPQ and DRR, create a 3-node topology (set data rates to 40-10 Mbps), where the middle node is a QoS-enabled router, and the two outer nodes are end-hosts running two or more instances of a client-server bulk data transfer application.

- SPQ: to validate your SPQ implementation, you are to use two client-server bulk data transfer applications A and B. Configure the router on the path so that A's traffic is prioritized over B. Application A should begin sometime after application B starts. In this scenario, your router has two queues with two priority levels.
- DRR: to validate your DRR implementation, you are to create three queues so that their corresponding quanta follow the 3:2:1 ratio. Furthermore, you use three bulk data applications so that the classifier places each in a separate queue. In this scenario, all three applications start at the same time.

For both simulation scenarios, your demonstration should include one throughput vs. time plot comprising all applications' traffic. An example of throughput vs. time plot is provided in Figure 3.

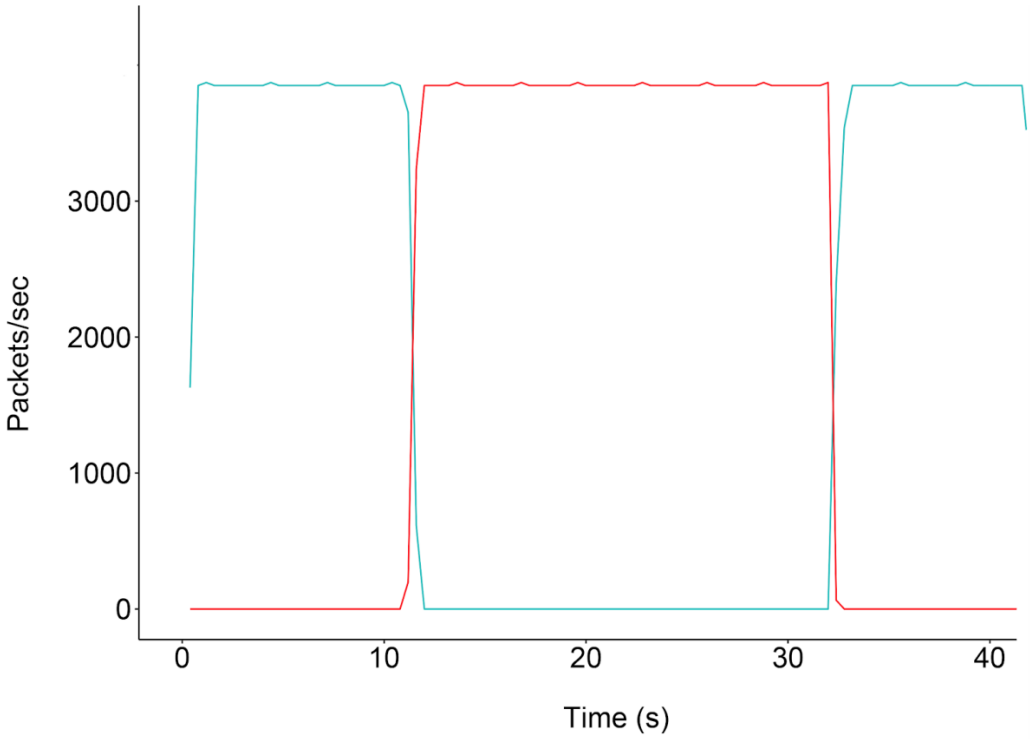

Figure 3: In this scenario, the sender starts sending only low priority traffic at time 0 sec (i.e., only application A is running). At time 12 sec, the sender sends both high and low priority traffic (i.e., both applications A and B are running). Once the application with high priority traffic begins, low priority traffic bandwidth allocation is severely reduced. Once high priority traffic ends, then low priority traffic regains the total available bandwidth allocation.