Project 2 Technical Specification
CS621: Network Programming – Spring 2019
Updated: March 4th, 2019

Project Outcomes

1. Build a base class for Differentiated Services in ns-3.
2. Implement Strict Priority Queueing (SPQ) and Deficit Round Robin (DRR), using the base class.
3. Validate and verify your SPQ and DRR implementations.

Overview

The ultimate purpose of this project is to learn about and 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 API for the base class is laid out for you. Your assignment is to implement it.

After completion of the base class, you will implement two differentiated services: SPQ and DRR. Refer to Lecture 10 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 applications and simulations.

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 filter aspect to sort the traffic packets into appropriate traffic queues.
- Scheduling - The schedule function carries out designed Quality-of-Service (QoS) algorithm to schedule which traffic queue to be served at the time.

Two separate functions need to be implemented per QoS algorithm design of the developers choice. For network queues, DoEnqueue() and DoDequeue() functions can be overwritten 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 laid out in the UML diagram in Figure 1.

Here is a short description detailing the role of some of 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.

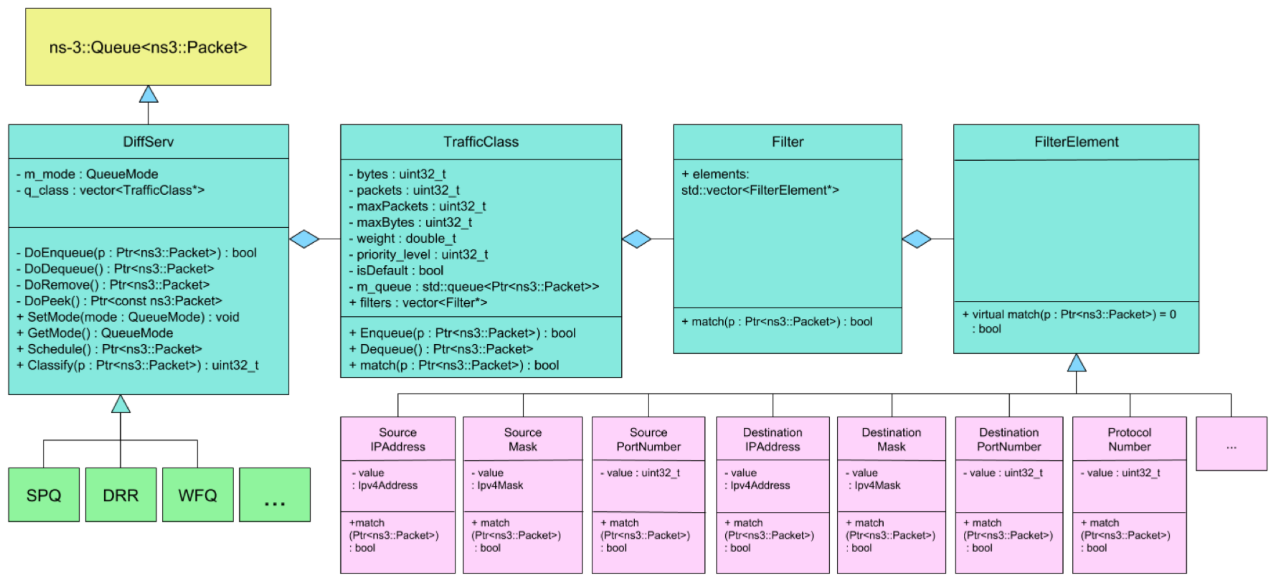
ns-3::Queue<ns3::Packet>
DiffServ
- m_mode : QueueMode
- q_class : vector<TrafficClass*>
- DoEnqueue(p : Ptr<ns3::Packet>) : bool
- DoDequeue() : Ptr<ns3::Packet>
- DoRemove() : Ptr<ns3::Packet>
- DoPeek() : Ptr<const ns3:Packet>
+ SetMode(mode : QueueMode) : void
+ GetMode() : QueueMode
+ Schedule() : Ptr<ns3::Packet>
+ Classify(p : Ptr<ns3::Packet>) : uint32_t
TrafficClass
- bytes : uint32_t
- packets : uint32_t
- maxPackets : uint32_t
- maxBytes : uint32_t
- weight : double_t
- priority_level : uint32_t
- isDefault : bool
- m_queue : std::queue<Ptr<ns3::Packet>>
+ filters : vector<Filter*>
+ Enqueue(p : Ptr<ns3::Packet>) : bool
+ Dequeue() : Ptr<ns3::Packet>
+ match(p : Ptr<ns3::Packet>) : bool
Filter
+ elements:
std::vector<FilterElement*>
+ match(p : Ptr<ns3::Packet>) : bool
FilterElement
+ virtual match(p : Ptr<ns3::Packet>) = 0 : bool
SPQ
DRR
WFQ
...
Source IPAddress
- value : Ipv4Address
+match (Ptr<ns3::Packet>) : bool
Source Mask
- value : Ipv4Mask
+ match (Ptr<ns3::Packet>) : bool
Source PortNumber
- value : uint32_t
+ match (Ptr<ns3::Packet>) : bool
Destination IPAddress
- value : Ipv4Address
+ match (Ptr<ns3::Packet>) : bool
Destination Mask
- value : Ipv4Mask
+ match (Ptr<ns3::Packet>) : bool
Destination PortNumber
- value : uint32_t
+ match (Ptr<ns3::Packet>) : bool
Protocol Number
- value : uint32_t
+ match (Ptr<ns3::Packet>) : bool
...

Figure 1: Base Class Design

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

- TrafficClass - Defines the characteristics of a queue.
  - bytes - the count of bytes in the queue.
  - packets - the count of packets 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 weight.
  - priority_level - applicable if the QoS mechanism uses priority level.
  - 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 on. You should write one subclass for every seven (light purple) boxes in the design diagram. Refer to Figure 2 for an example.

(2) Implementing SPQ and DRR with DiffServ

You will make two separate implementations, one implementing Strict Priority Queueing, and the other implementing DRR, both using your implemented DiffServ class.

Your SPQ implementation should take exactly one command line input, the name of a config file. In that file you should specify the number of queues and a priority level for each queue.

Your DRR implementation should also take only that one input for the config file. It specifies the number of queues and the corresponding quantum value for each queue.

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

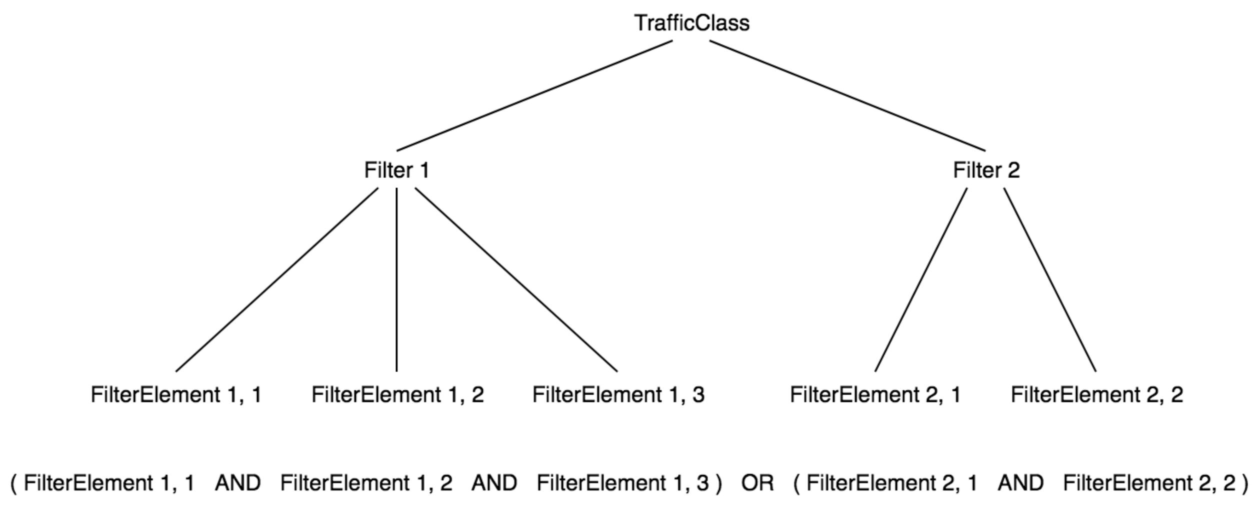

Figure 2: An example of how users of your base class will be able to match any boolean condition given the right set of Filters and FilterElements.

(3) Simulation Verification and Validation

As the last part of your project, you are to create ns-3 applications and simulations to demonstrate the correctness of your implementations above. For both SPQ and DRR, create a 3-node topology (set data-rates to 4-1 Mps), where the middle node is a QoS-enabled router and two outer nodes are end-hosts running simple client/server bulk data transfer applications.

To validate SPQ, you should create two client server applications A and B. Configure the router on the path so that, A traffic is prioritized over B. Application A should begin sometime after application B starts. In this case, your router has two queues.

For your DRR verification, you should initialize three queues with quanta ratio 3:2:1. Then, start three client-server bulk data transfer applications, all starting at the same time.

For both simulations your demonstration should include a plot of throughput vs. time for all applications. An example of throughput vs. time plot is provided in Figure 3.

Submission Requirements and Deadlines

You final submissions will be graded on design, implementation, code quality, coding style, and documentation quality. You will submit two documents: (1) Technical usage manual, and (2) final report. You should follow the ns-3 coding style. While you are not required to submit a design document, you are strongly encouraged to maintain one in order to deliver your project successfully.

There will be two checkpoints for this project, one on **March 28th** (in-class), and the other some time on **the week of 4/15**. All of your code and technical manual is due on **May 1st**. Commits made after the due date will not be reviewed.

Keep in mind that at the end of the semester, you will present your project on **May 9th** in class, and submit a final report that is due **May 15th, 11:59PM**.

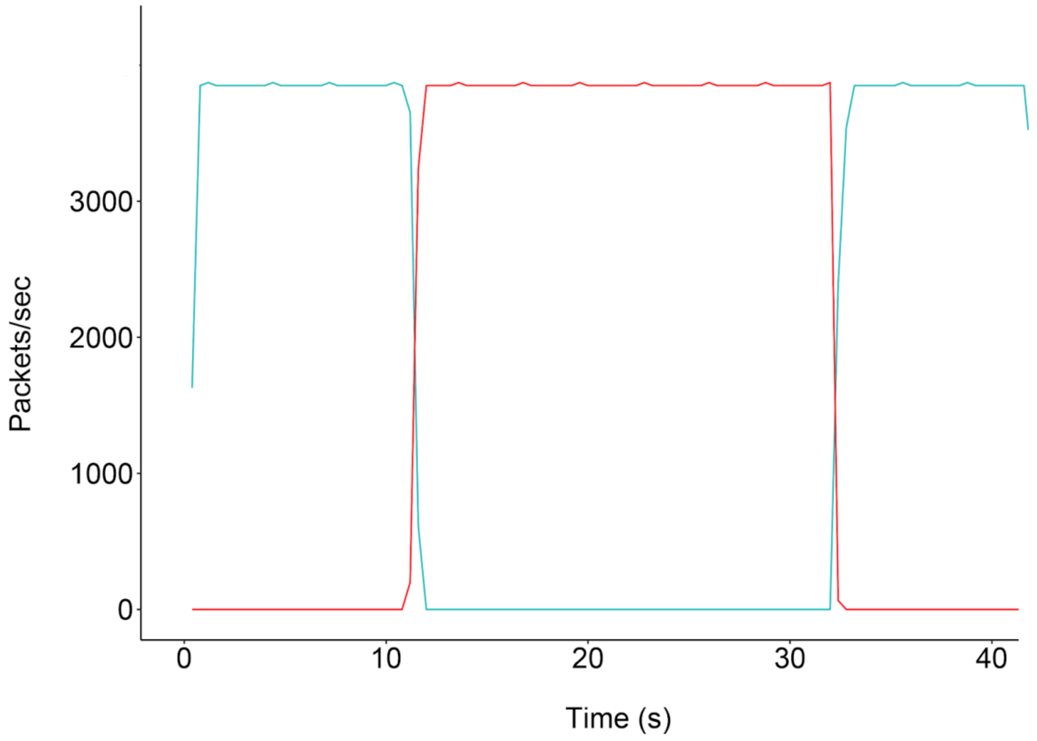

Figure 3: In this scenario, the sender starts sending low priority traffic at time 0s. At time 12s, the sender sends both high priority traffic. Once high priority traffic begins, low priority traffic bandwidth allocation is severely reduced. Once high priority traffic ends, then low priority traffic regains full available bandwidth allocation.