Base Class to Simulate Differentiated Services

Aditya Kunatharaju
University of San Francisco
avkunatharaju@dons.usfca.edu

1 INTRODUCTION

This a project to implement a selection of differentiated services. Two quality-of-service (**QoS**) mechanisms have been implemented: Strict Priority Queue (**SPQ**) and Deficit Round Robin (**DRR**). The network topology and simulation was build and implemented using the ns-3 library.

Built in 3 parts:

(1) Base Class. DiffServ, TrafficClass, Filter and FilterElements classes built.
(2) SPQ and DRR. These derived from the base DiffServ.
(3) Simulation Validation. Network topology and running a 3 node network with SPQ or DRR.

The project serves to abstract away the large amount of common componenets among the SPQ and DRR differentiated services. Simulation validation required creating multiple nodes and UDPClient and UDPServer applications in ns-3.

2 DESIGN

There are two main design components to this project. First, the design of the base DiffServ class and all the classes it needs. Second, implementing SPQ and DRR as dervied classes of DiffServ, by overriding the relevant methods to apply the QoS.

2.1 Base Class

The design of the base classes followed this UML:

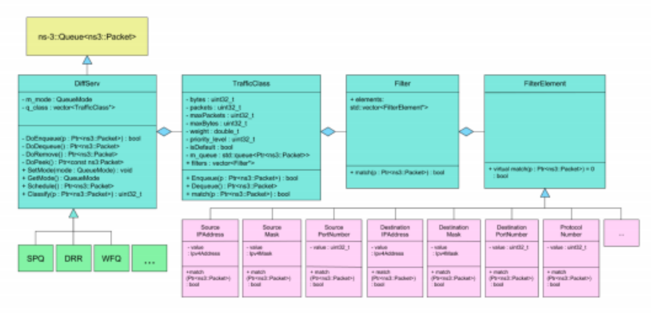
ns-3::Queue<ns3::Packet>
DiffServ
- m_mode : QueueMode
- q_class : vector<TrafficClass*>
- DoEnqueue(p : Ptr<ns3::Packet>) : bool
- DoDequeue() : Ptr<ns3::Packet>
- DoRemove() : Ptr<ns3::Packet>
- DoPeek() : Ptr<const ns3::Packet>
+ SetMode(mode : QueueMode) : void
+ GetMode() : QueueMode
+ Schedule() : 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
Source Mask
Source PortNumber
Destination IPAddress
Destination Mask
Destination PortNumber
Protocol Number
...

Figure 1: Base Class Design

Here is a brief description of DiffServ and TrafficClass methods.

DiffServ:

q_class: The vector of TrafficClass pointers.
Schedule(): Returns a copy of the packet to transmit.
Classify(packet): Takes a packet and return the index of the qclass it should be sent/enqueued to.
DoEnqueue(packet): Calls Classify(packet) and enqueues the packet to the appropriate q_class/TrafficClass.
DoDequeue(): Calls Schedule() to get a copy of the next packet to be dequeued. Calls Classify() on this packet to get the index of the TrafficClass in q_class to call TrafficClass.Dequeue() on.
AddTrafficClass(tc): Adds the TrafficClass tc to the q_class.

TrafficClass:

set_maxPackets(num) / get_maxPackets(): Set / get the maxPackets that can be stored in the m_queue.
Enqueue(packet): Enqueues a packet to the TrafficClass m_queue.
Dequeue(): Dequeues the packet at the front of the m_queue.
set_weight(w) / get_weight(): Set / get the weight of the TrafficClass. Used in DRR.
Match(packet): Call Match(packet) on all the filters in Filters vector and return True if any one matches.

Here is how Match() in TrafficClass further calls Match on the FilterElements in the TrafficClass's Filters.

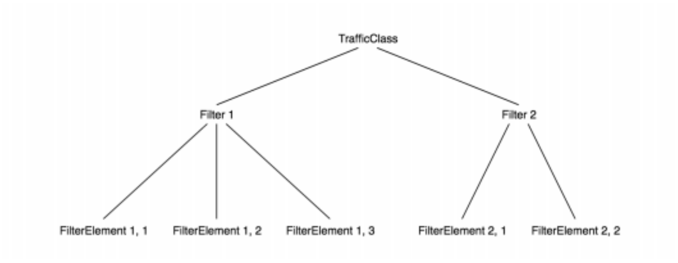

Figure 2: Matching

2.2 QoS Classes

This project implemented two QoS: SPQ and DRR.

SPQ has queues with different priorities; packets get enqueued in the approrpiate queue depending on certain characteristics. Queues are dequeued in order of priority; a lower priority queue is only dequeued if the queues with higher priorities are empty.

DRR attempts to be more fair than SPQ and allowing low priority queues to intermittently send packets. Each flow or queue is assigned a "deficit counter" that determines the amount of bandwidth it can use in each scheduling round. The deficit counter starts with an initial value, and if the next packet in the queue is smaller than the deficit, the packet gets dequeued and the deficit decremented by packet size. The implementation and pseudocode for the DRR Schedule(), Enqueue() and Dequeue() was obtained from Shareer and Verghese and Wikipedia.[3][1]

Here's an overview of the DiffServ methods that were overridden in SPQ and DRR.

SPQ

Schedule(): Checks the TrafficClass's in order of priority and peeks the packet at the front of the first-non-empty TrafficClass.

DRR

Schedule(): Checks the activeList for the TrafficClasses that are not empty. Gets the front() of the activeList and peeks the TrafficClass at that index in the q_class vector.
DoEnqueue(packet): Calls Classify(packet) to get the TrafficClass packet should be enqueued to. Checks activeList if TrafficClass is present, if not, adds the TrafficClass to the activeList.
DoDequeue(): Calls Schedule() to get a copy of the next packet to be dequeued. Calls Classify() on this packet to get the index of the TrafficClass in q_class to call TrafficClass.Dequeues packet from the TrafficClass if the relevant deficit is greater than the size of the packet, else increments the relevant deficit by the weight of the TrafficClass.

3 DESIGN ISSUES AND LIMITATIONS

Our project implementation assumes that the primary goal of the project is to implement, test and validate the SPQ and DRR QoS' as differentiated services derived from an abstract base class, and not the robustness or adaptability of the program. While some robustness has built in-to the program, certain liberties have been taken in that regard to progress towards and reach the main goal of the project.

3.1 ns-3

- **Scalability:** Our implementation is built on a three node network topology. Could be re-worked to include more nodes but that is not an option currently.
- **NS-3 Learning Curve:** Due to a lack of instructive documentation for NS-3, the scope of the project might have been limited.
- **Multi-threading Support:** The program currently doesn't provide multi-threading support, i.e., elements such as the size of the various vectors are not locked or made thread-safe.

3.2 Project Implementation

- **SPQ Queues:** Our implementation assumes that TrafficClasses are added to the SPQ q_class in order of priority.
- **Filter Matching:** Our implementation matches packets queues to TrafficClasses on only the destination ports.
- **Hardcoding:** maxPackets for TrafficClass, number of packets sent by the UDP clients, start and stop time for node applications, data-rate and delay for the network links have been hardcoded.
- **Config File:** The config.json files need to certain parameters and values for the program to run properly. Config.json files were parsed with the rapidjson[2] library.
- **Default Queue:** If a packet does not match any TrafficClass, the last TrafficClass in the q_class is defaulted to as the classification.

4 IMPROVEMENTS

The project can be improved making it more robust. This could be done by defaulting to values in case parameters are not included in the config.json files. The hardcoding mentioned in the Design Issues and Limitations section can be changed to variables instantiated by the config file.

5 SIMULATION VALIDATION

5.1 SPQ

Here is how the packet flow looks from the router node to the server node. The high-priority packets are in blue, and low priority in orange. Flow initially starts with low priority packets being sent, however, once high-priority packets are available, only high-priority packets get sent. Low-priority flow only resumes after all high-priority packets have been sent.

Thus, SPQ is validated.

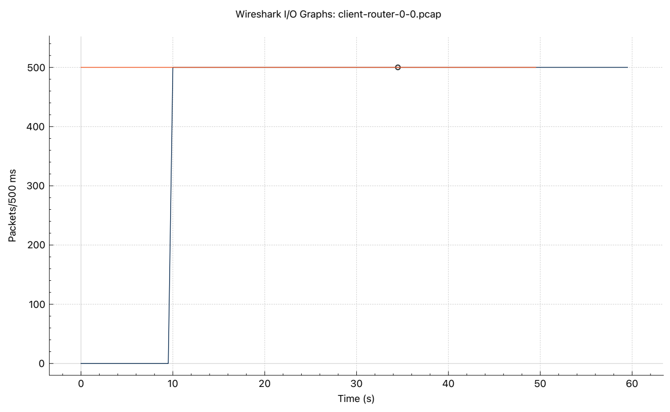

Figure 3: Pre-SPQ Flow Graph

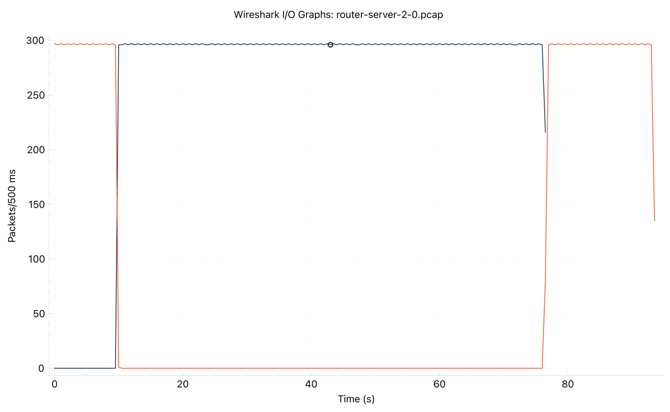

Figure 4: Post-SPQ Flow Graph

5.2 DRR

From the figure below, we see that all flows start at the same time. However, the bandwidth or number of packets sent per second is in order of the weights associated with the TrafficClasses. The weights ratio for blue, orange, green is 3 2:1 respectively, we see that the available bandwidth is split in these ratios.

Thus, DRR is validated.

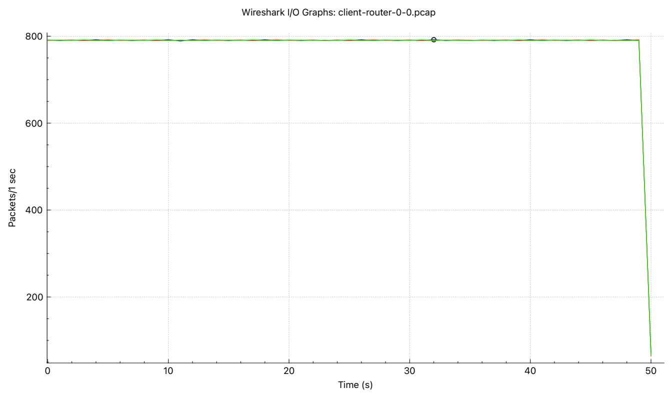

Figure 5: Pre-DRR Flow Graph

latex-sigcomm18-master/figures/router-server_drr.png

Figure 6: Post-DRR Flow Graph

6 API USAGE

Key elements a user needs to know to use are:

- **Filter.AddFilterElement(filterElement)**. Adds a FilterElement to the vector in Filter.
- **TrafficClass.AddFilter(filter)**. Adds a Filter to the TrafficClass filters vector.
- **TrafficClass.set_maxPackets(num)**. Sets the max_Packets for that TrafficClass to num.
- **TrafficClass.set_weight(w)**. Sets the weight of the TrafficClass to w.
- **DiffServ.Get_QClass()**. Gets the DiffServ q_class vector.

- **DiffServ.InitializeDeficit(num)**. Pushes num to the deficitVector. deficitVector[i] acts as the initial deficit value for the TrafficClass q_class[i].

7 ALTERNATIVE DESIGN

Implementing the improvements mentions in the Improvements section and addressing the points brought up in the Design Issues and Limitations should make the project implementation stronger.

REFERENCES

[1] [n. d.]. ([n. d.]).
[2] [n. d.]. rapidjson. ([n. d.]). Available at https://rapidjson.org.
[3] 2023. Deficit Round Robin. (2023). https://en.wikipedia.org/wiki/Deficit_round_robin