Simulating WFQ/WRR/Priority Queuing using Omnet++

CSC 467 - Switching, Network Traffic and Quality of Service

Che-Chi (Jack) Liu
Dhaimil Modi
April 8, 2019

Table of Contents

1. QoS Understanding

1.1. Quality of service (QoS): Queueing and Scheduling

WRR/WFQ/Priority Queueing is chosen as my QoS topic for this project. So, what is queueing? According to Wikipedia, “Queueing theory is the mathematical study of waiting lines, or queues” [1]. Think of queueing like arriving in line then waiting in line till it is your turn for you to be served, for example, the checkout line at the grocery store. A queue, in computer networking, is a collection of data packets collectively waiting to be transmitted over the network. A queuing point is necessary when there is a rate mismatch, contention for resource, or controlled by QoS by treating different classes of traffic independently. Basic functions for queueing are to **add** packets to the appropriate queue; enqueue, to **drop** packets when the queue buffering (the length of the queue) is full; dequeue, and most importantly to **remove from queue and transmit** packets when so directed by the scheduler and the scheduling discipline. The scheduler controls when a packet can leave its queue to appropriately meet the QoS objectives and it is also responsible for allocating bandwidth to the queues. In addition, a scheduler also has to implement a scheduling discipline for it to serve the queues according to the algorithm.

First, an ideal scheduling discipline needs to be easy to implement since it has to make a decision once every few microseconds. Second, it needs to be fair as it decides how a resource is shared among the packets, it needs to satisfy the max-min fairness where each flow gets no more than what it wants and if there is any excess then it is equally shared among the flows. Third, it provides performance bounds such as bandwidth, delay, delay-jitter, and packet loss in order to guarantee a desired level of service. Lastly, it has to be an ease of admission control as

it controls how much bandwidth is finally given and it decides if a new flow can be allowed and share the resource. In this project, Dhaimil and I will study three different types of scheduling disciplines: **Priority queuing (PQ), weighted fair queuing (WFQ), and weighted round robin (WRR).**

Priority Queuing (PQ)

Priority queueing is a scheduling technique that assigns each queue a priority level and it schedules traffic based on the queue's priority level. Priority queueing always serves packets waiting in the higher-priority queues first. Lower-priority queue's packets can only be served when there are no packets waiting in higher-priority queues. For example, four different priorities are assigned to the four different traffic classes' queue below: **High** => Low bandwidth urgent messages, **Medium** => Real-time, **Normal** => Non-real-time, and **Low** => Best-effort

Figure 1. Priorities of the four different classes. [2]

If the High queue has a packet waiting, the scheduler will service it first. If there is no packet in the High queue, the scheduler will look to service the Medium queue. It will transmit one packet from the Medium queue, and then again look for any packets waiting in the High queue. The

Low queue only gets serviced if there are no packets waiting in High, Medium and Normal queues. The High queue is getting the lowest delay. But one downside about priority queueing is that the traffic of other lower-priority queues may face a complete resource starvation if there are always packets waiting to be served in the higher-priority queues. In other words, service for lower-priority depends on higher-priority traffic. This type of scheduling is very simple and efficient to arbitrate only between a small number of queues (traffic classes) and the priority levels can be assigned according to different traffic classes' delay. Since priority queueing is only effective, efficient, and fair between a small number of classes, **weighted fair queuing (WFQ)** can be used between a large number of classes as it provides some level of fair access to the bandwidth.

Weighted Fair Queueing (WFQ)

Weighted fair queueing is a scheduling technique that consist both packet-based practical implementation of generalized processor sharing (GPS), and a natural generalization of fair queueing (FQ). Instead of giving each flow an equal share of the link's capacity, WFQ allows schedulers to specify, for each flow, which fraction (weight) of the link's capacity will be given. WFQ is also known as packet-by-packet GPS (PGPS) "since it approximates generalized processor sharing to within one packet transmission time, regardless of the arrival patterns" [2]. Since we can't serve infinitesimals, only packets in its entirety, GPS is unimplementable.

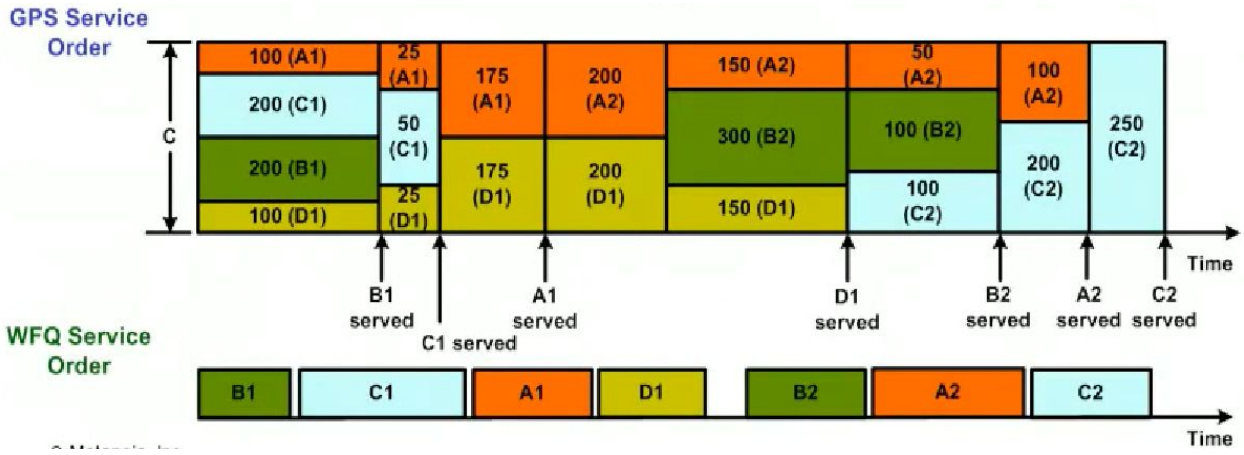

Figure 2. WFQ and GPS relation [3].

As you can see in the diagram above, WFQ serves packets in the GPS service order but based on increasing order of the finishing deadline. In other words, it selects packets based on their finishing time when served bit-by-bit in GPS. For example, B1 is first served in GPS so B1 is the first packet to be served in WFQ. Then C1, A1, D1, B2, A2, and C2. But when there is a tie, the flows are re-ordered randomly.

Weighted Round Robin (WRR)

Weighted round robin is a scheduling technique that address the shortcomings of PQ and FQ. WRR serves the queues according to their weight in a round-robin fashion using cycles. In each cycle, all the queues will have an opportunity to transmit a cell or packet. Since all queues are weighted, queues with larger weights can have multiple transmission opportunities in each cycle.

For example, the **bulk service** WRR transmits packets like the diagram below. Number of packet getting transmitted is based on the weight of the queue, for example, A => 1, B => 2, C => 2, D => 1. As you can see in round 1, one packet is transmitted from queue A, two from queue B, two from queue C, and one from queue D.

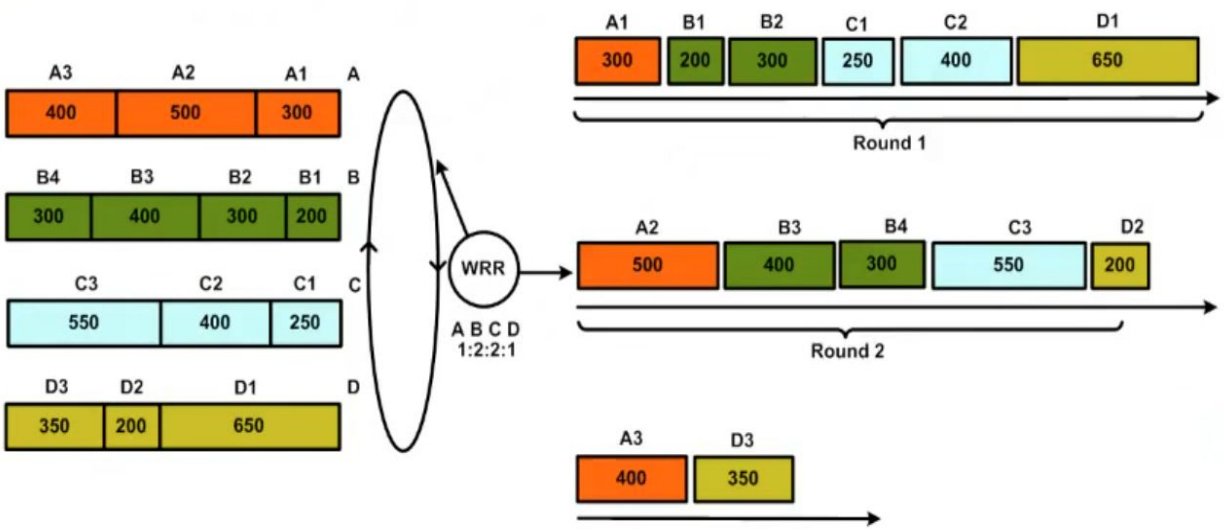

Figure 3. Bulk service WRR Example [3].

Another example, the **smooth service** WRR transmits packets like the diagram below.

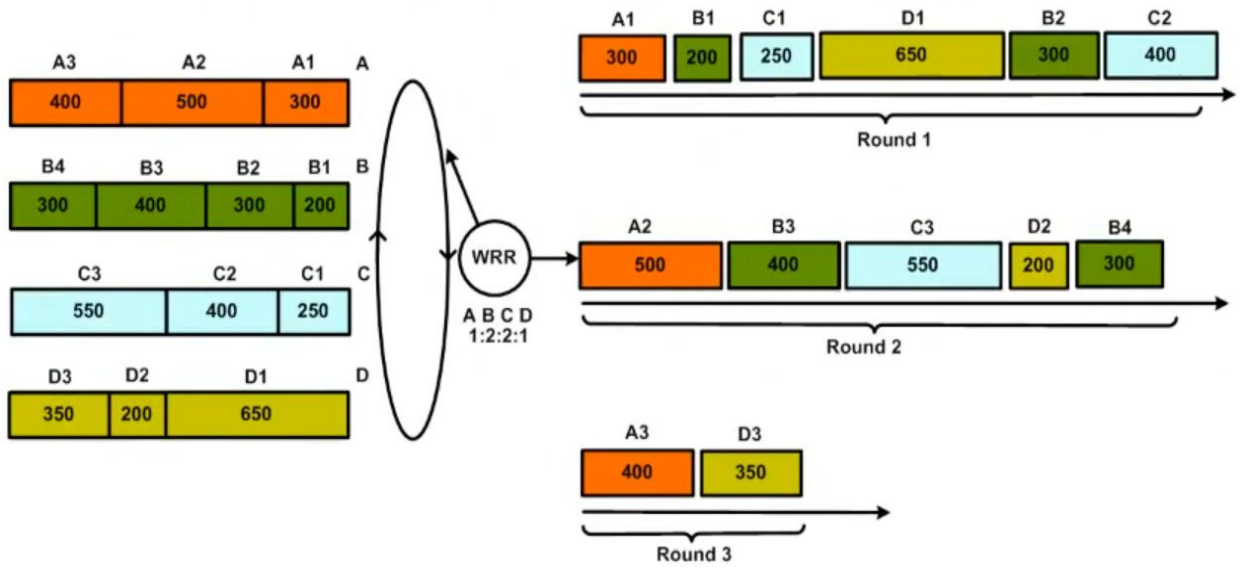

Figure 4. Smooth WRR Example [3].

Instead of transmitting multiple packets from weighted queue all at once during each cycle like bulk service, smooth service only transmits one packet from the weighted queues during each cycle and then it comes back and transmits one packet again from the weighted queues till the weight number of packets has been transmitted. As you can see in round 1, B1 and B2 are not bunched together as was the case in bulk service. If more bunching accumulates, it results in variations in delay and jitter, so smooth service is a way of smoothing out the output stream.

1.2. Quality of service (QoS) Summary

QoS (Quality of Service) is necessary in today's day and age to ensure that all users get exactly the service they require in an efficient and effective manner. For many years, we've had to rely

on the woeful Best Effort (BE) service to power all users utilizing the internet. While this may have been feasible in the past due to having simpler applications that don't need as much service and just the sheer lack of users and applications on the internet, we cannot rely on BE services any longer.

BE tries to deliver data to its intended location but doesn't attempt to retransmit and doesn't deal with packets that have been lost. In essence, nothing is guaranteed so you don't know anything about the delivery. Examples of BE services include simple applications that have been present since the creation of the internet such as emailing services. In contrast, QoS requires a set of rules or guidelines that allows networks to differentiate traffic depending on the class of traffic and simultaneously deal with this traffic accordingly (ie. more resources for higher class of traffic and vice versa). As for users, they simply need to get the service they were guaranteed to ensure effective QoS.

As mentioned already, QoS ensures that traffic (packets in traffic) are differentiated and treated accordingly. The Internet Engineering Task Force (IETF), an organization that regulates internet standards, has come up with a Differentiated Services architecture known as DiffServ, which we will use to simulate a network with different scheduling algorithms (Weighted Round Robin (WRR) and a Priority Queue). DiffServ essentially allows us to classify and manage traffic and manage QoS on today's networks, though it isn't widely used yet. These simulations we have performed will show the contrast of using DiffServ over the default BE and hopefully convey the importance of integrating an effective QoS mechanism, specifically DiffServ, in today's internet entrenched world.

2. Simulation/Emulation Setup

The following section outlines the infrastructure of the network and the schedulers used to simulate all Experiments.

2.1. Network Design

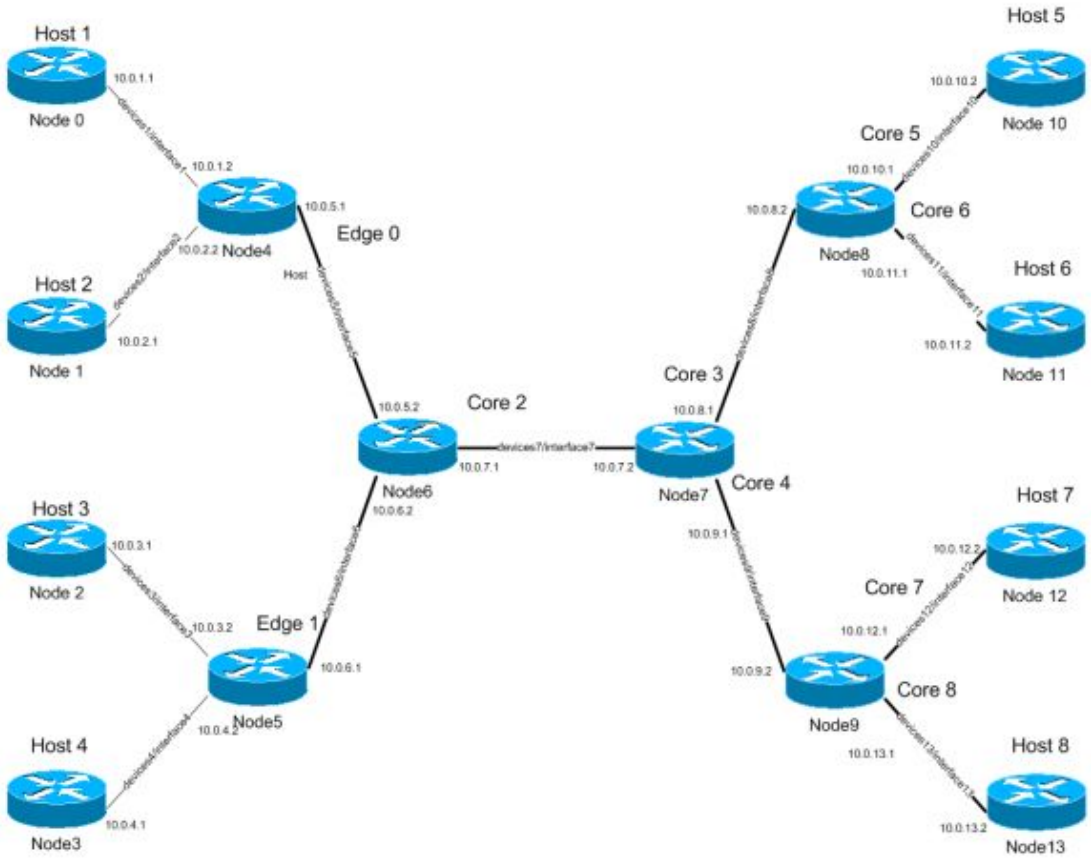

Figure 5. Network infrastructure used for all Experiments. [4]

This network topology was used to replicate an actual network. The two main points of congestion in Figure 5 are at **Edges 0 & 1,** and at **Core 2**. In both, users have to compete for resources to make it through. In the case of Core 2, the competition is greater since it is the only

link that allows users to reach their destination. To elaborate, data is only sent in one direction (Hosts 1-4 send data while Hosts 5-8 receive data).

2.2. Node Design

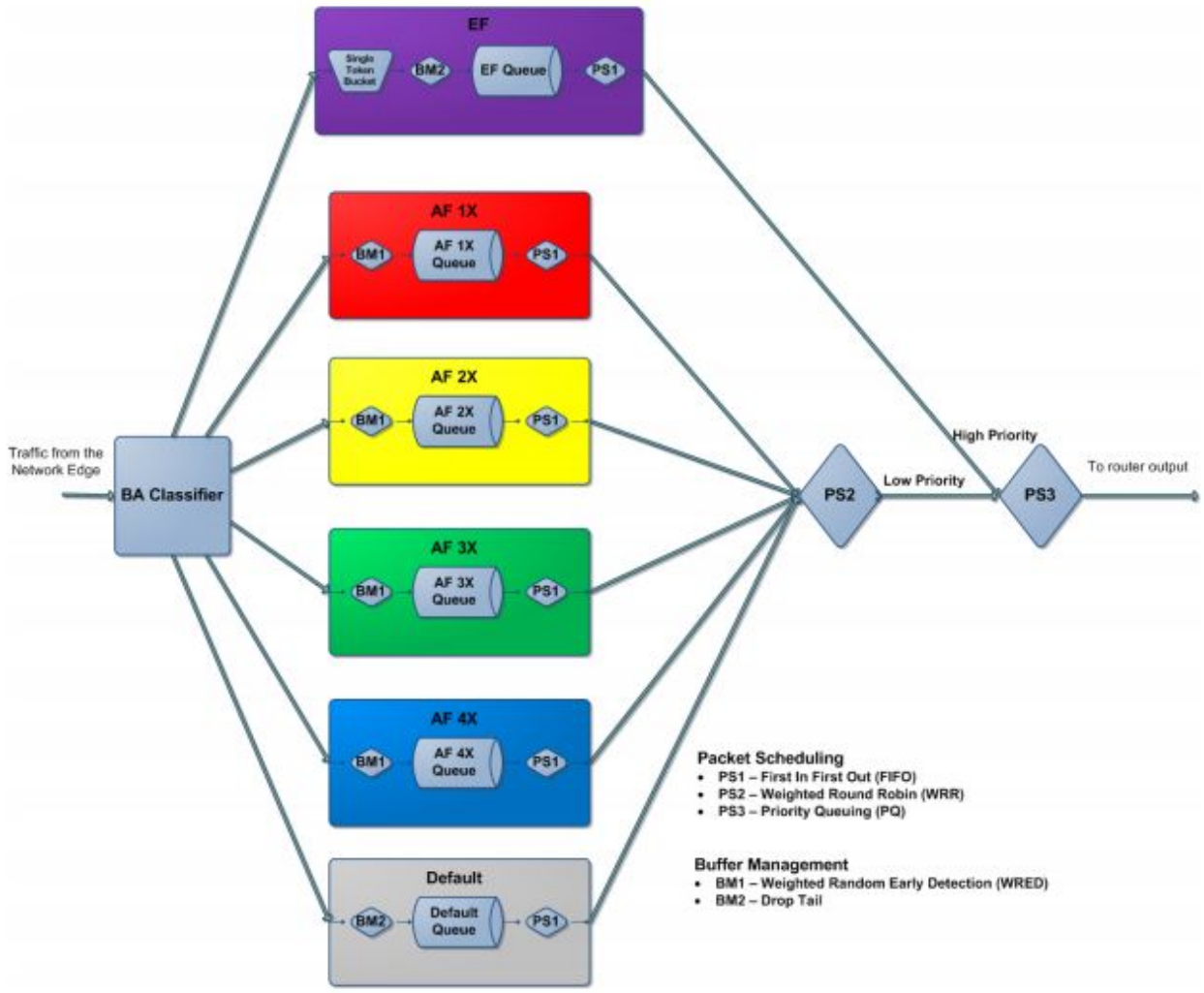

Figure 6. Design of DSQueue1 used for all Experiments.[4]

While we don't use Weighted Random Early Detection (WRED) explicitly, the per-hop-behaviour (PHP) of the AF queues rely on the WRED implementation to perform these experiments which was already present in the inet framework for Omnet++.

2.3. Experiments

All Experiments were carried out using the topology shown in Figure 6. Note that Experiments 1.1 - 1.7 (inclusive) all had a WRR weight of 0 for BE. Additionally, Experiments 1.1 - 1.6 (inclusive) used a buffer size of 100 packets for each internal queue, whereas Experiment 1.7 used a buffer size of 400 packets (equal to combined size of AF queues in the past Experiments) entering a single queue. This was to be fair across all Experiments. The final weights used can be seen below in Figure 7.

Experiment#	Queue Weights (AF1, AF2, AF3,AF4)	SLA to forwarding class mappings (SLA 1, SLA 2, SLA 3, SLA 4)
1.1	10,9,8,7	AF1,AF2,AF3,AF4
1.2	9,8,7,6	Same as above
1.3	8,7,6,5	Same as above
1.4	7,6,5,4	Same as above
1.5	1,1,1,1	Same as above
1.6	1,1,1,n/a	AF1,AF2,AF3,EF
1.7	n/a	Default (queue size = 400)

Figure 7. Weights used for Experiments 1.1 - 1.7. [4]

2.3.1. Experiment 1.1

In Experiment 1.1, we used different WRR weights for AF1, AF2, AF3, and AF4; 10, 9, 8, 7, respectively.

2.3.2. Experiment 1.2

In Experiment 1.2, we used different WRR weights for AF1, AF2, AF3, and AF4; 9, 8, 7, 6, respectively.

2.3.3. Experiment 1.3

In Experiment 1.3, we used different WRR weights for AF1, AF2, AF3, and AF4; 8, 7, 6, 5, respectively.

2.3.4. Experiment 1.4

In Experiment 1.4, we used different WRR weights for AF1, AF2, AF3, and AF4; 7, 6, 5, 4, respectively.

2.3.5. Experiment 1.5

In Experiment 1.5, we used equal WRR weights for AF1, AF2, AF3, and AF4; 1, 1, 1, 1, respectively.

2.3.6. Experiment 1.6

In Experiment 1.6, we used different WRR weights for AF1, AF2, AF3, and EF; 1, 1, 1, 0, respectively. Note that AF4 was not given a weight in this Experiment.

2.3.7. Experiment 1.7

In Experiment 1.7, we used no WRR weights for AF1, AF2, AF3, and AF4; ie. BE network service was used.

2.4. Parameters

```
connections:
    H1.ethg++ <--> ethernetline <--> R0.ethg++;
    H2.ethg++ <--> ethernetline <--> R0.ethg++;
    H3.ethg++ <--> ethernetline <--> R1.ethg++;
    H4.ethg++ <--> ethernetline <--> R1.ethg++;
    H5.ethg++ <--> ethernetline <--> R4.ethg++;
    H6.ethg++ <--> ethernetline <--> R4.ethg++;
    H7.ethg++ <--> ethernetline <--> R5.ethg++;
    H8.ethg++ <--> ethernetline <--> R5.ethg++;

    R0.pppg++ <--> edgeline <--> R2.pppg++;
    R1.pppg++ <--> edgeline <--> R2.pppg++;
    R4.pppg++ <--> coreline <--> R3.pppg++;
    R5.pppg++ <--> coreline <--> R3.pppg++;

    R2.pppg++ <--> coreline <--> R3.pppg++;
```

Figure 8. Networking infrastructure code.

This is the code we used for creating the network. In section 1, we connected Host 1-8 to its edge router. In section 2, we connected the edge routers to their core router. Lastly, we then connected the 2 core routers together. We set a delay of 2 ms for the edge_line and core_line and a delay of 0.1 us for the ethernet_line. Edge_line and core_line both have a data rate of 500 kbps and ethernet_line has a data rate of 100 Mbps. The voice packets sent from Host 1-4 have a size of 172 bytes and the video packets have a size of 500 bytes.

```
connections:
    in --> classifier.in;
    classifier.outs++ --> efQueue.in;
    classifier.outs++ --> af1xQueue.in;
    classifier.outs++ --> af2xQueue.in;
    classifier.outs++ --> af3xQueue.in;
    classifier.outs++ --> af4xQueue.in;
    classifier.defaultOut --> beQueue.in;

    af1xQueue.out --> wrr.in++;
    af2xQueue.out --> wrr.in++;
    af3xQueue.out --> wrr.in++;
    af4xQueue.out --> wrr.in++;
    beQueue.out --> wrr.in++;

    efQueue.out --> priority.in++;
    wrr.out --> priority.in++;

    priority.out --> out;
```

Figure 9. Node hierarchy code.

This is the code we used for creating the queues. In section 1, we connected the network traffic as an input ‘in’ into the BA classifier. We then connected the output of BA classifier to input of each queue. In section 2, we connect the output of all AF queues and the default queue (BE) to the input of WRR scheduler. In section 3, we connected the output of EF queue and the output of WRR scheduler to a priority queue, with the output of EF queue having a higher priority than WRR scheduler. Lastly, the output of the priority queue is connected to the router output ‘out’.

3. Results

The following section shows the output results of all Experiments ran using the parameters above. These results include Average Queue Length and Packet Loss for each queue in router 2 in each Experiment.

3.1. Average Queue Length

Each of the blue bars below shows the Average Queue Length of each queue for the various Experiments performed. Experiment 1.7 did not have a scalar file for Average Queue length since only the BE service was used as shown above in Figure 7. **The order of queues from left to right is AF1, AF2, AF3, AF4, BE, EF.**

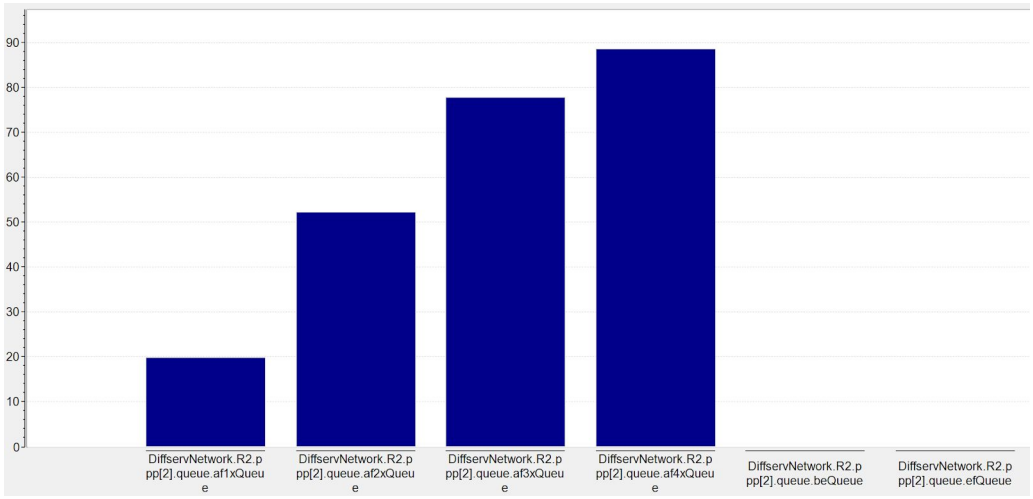

Figure 10. Average Queue Length for each type of queue in Experiment 1.1.

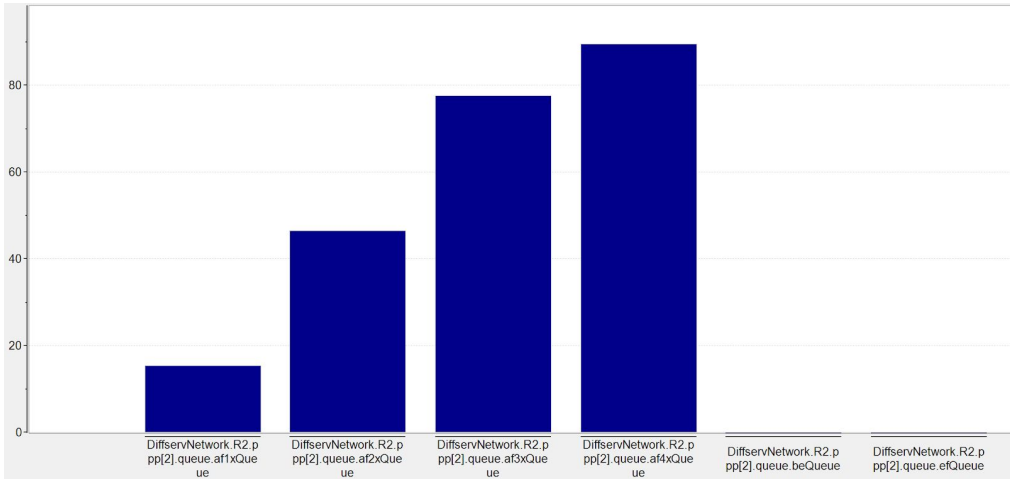

Figure 11. Average Queue Length for each type of queue in Experiment 1.2.

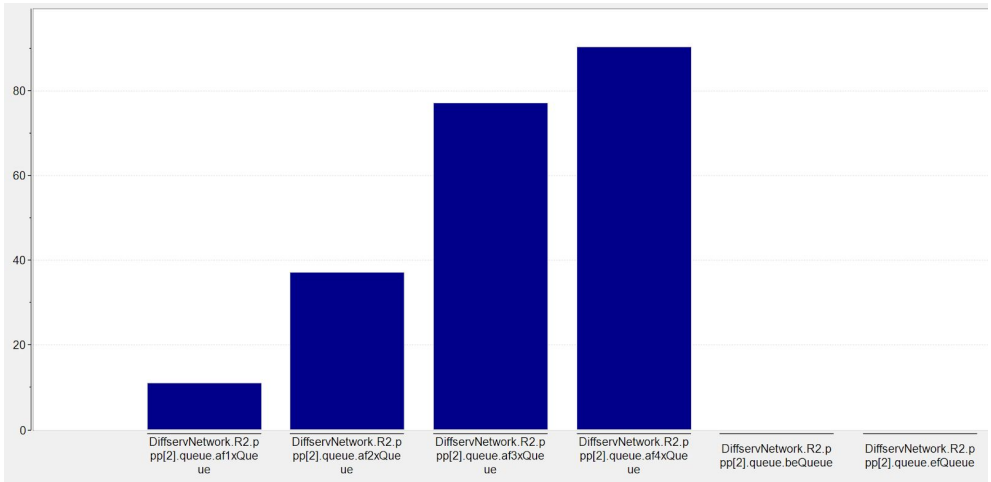

Figure 12. Average Queue Length for each type of queue in Experiment 1.3.

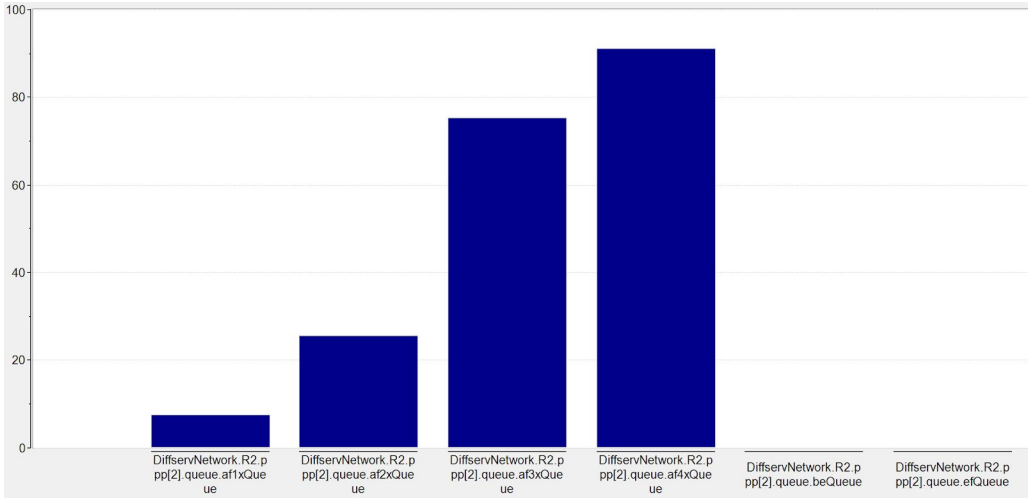

Figure 13. Average Queue Length for each type of queue in Experiment 1.4.

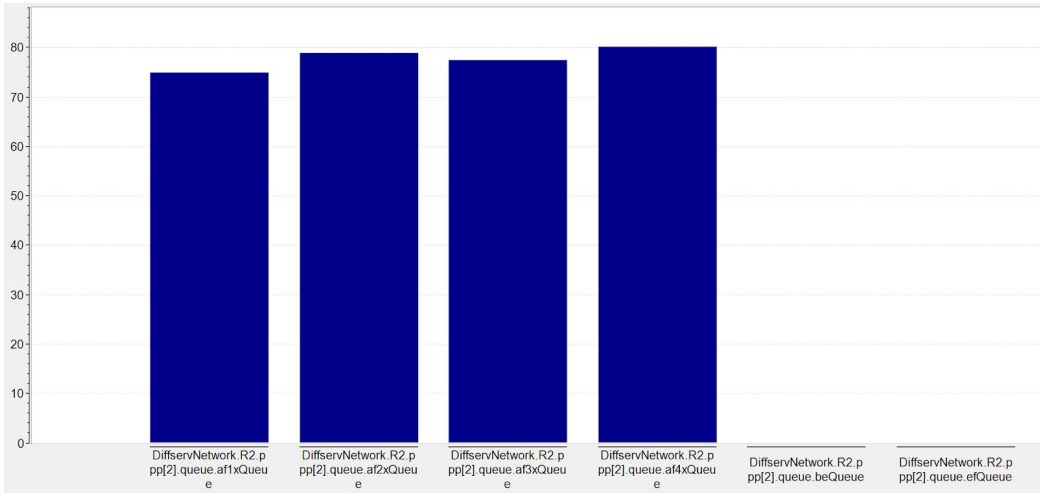

Figure 14. Average Queue Length for each type of queue in Experiment 1.5.

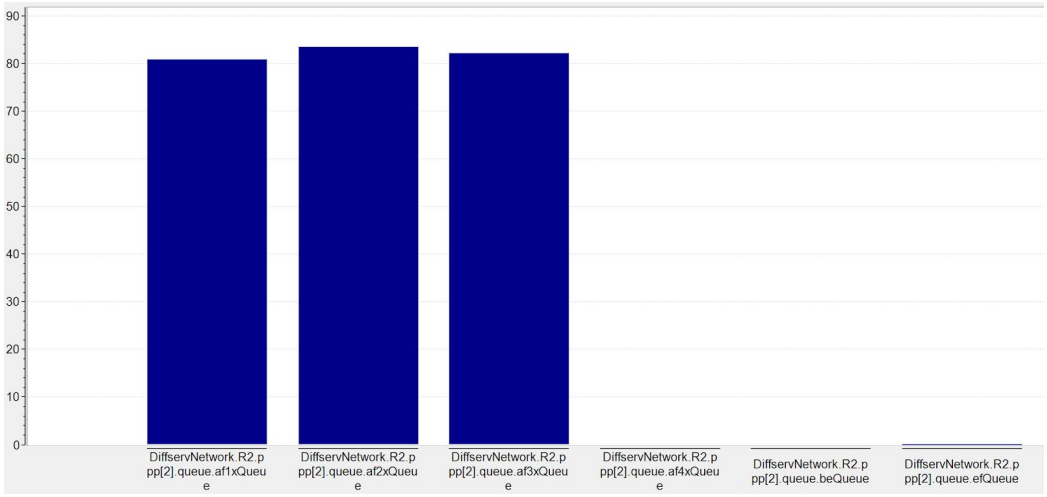

Figure 15. Average Queue Length for each type of queue in Experiment 1.6.

3.2. Packet Loss

Each of the blue bars represents the total packet loss experienced by each queue during the various Experiments. The faint red bars show the Average Queue Length of each queue in seconds. A better depiction of this variable can be seen above in Section 3.1. Experiment 1.7 did not have a scalar file for Packet Loss since only the BE service was used as shown above in Figure 7. **The order of queues from left to right is AF1, AF2, AF3, AF4, BE, EF.**

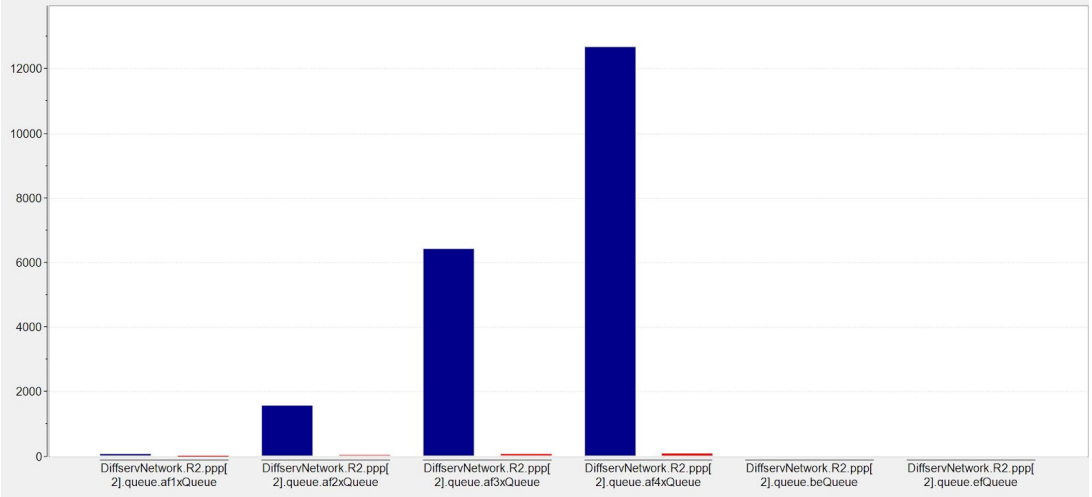

Figure 16. Packets dropped for each queue in Experiment 1.1.

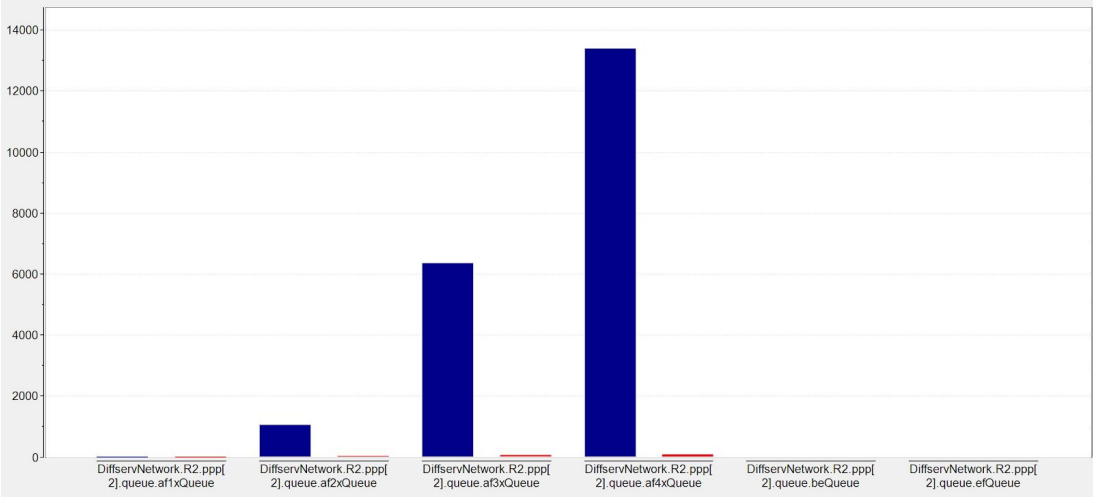

Figure 17. Packets dropped for each queue in Experiment 1.2.

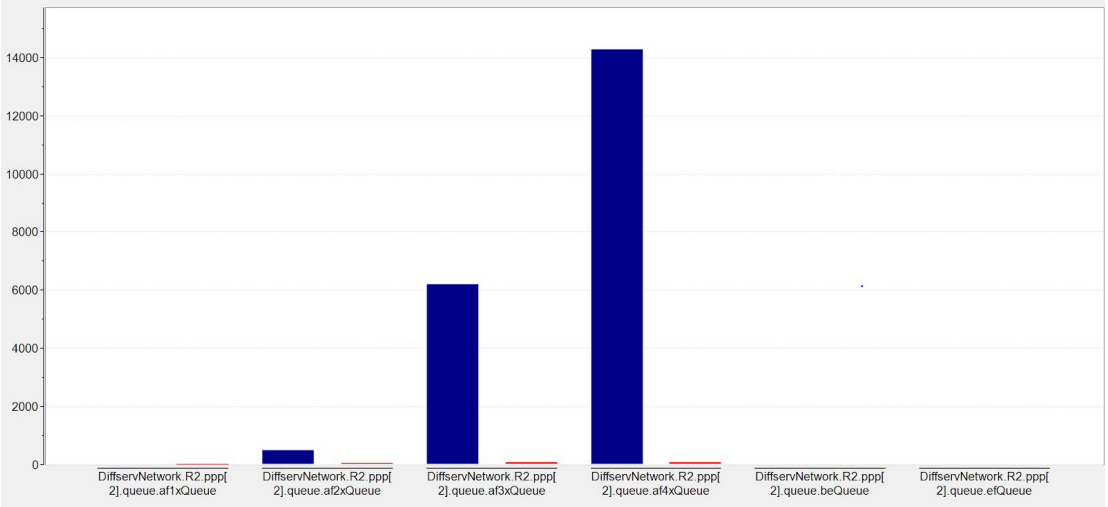

Figure 18. Packets dropped for each queue in Experiment 1.3.

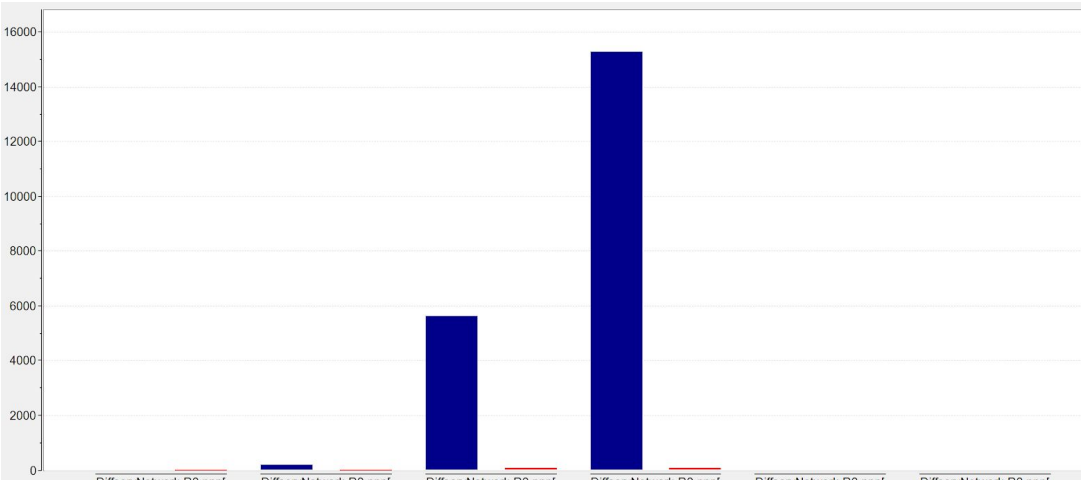

Figure 19. Packets dropped for each queue in Experiment 1.4.

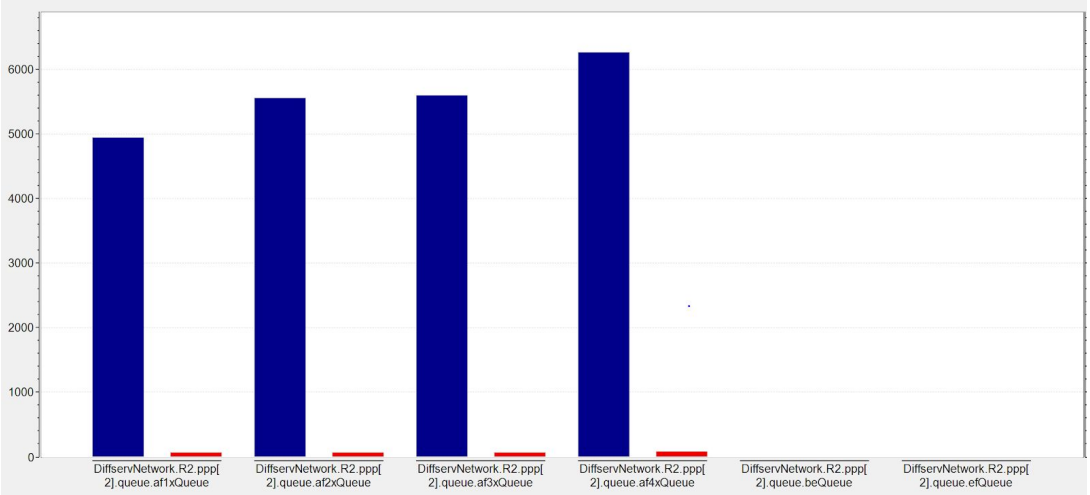

Figure 20. Packets dropped for each queue in Experiment 1.5.

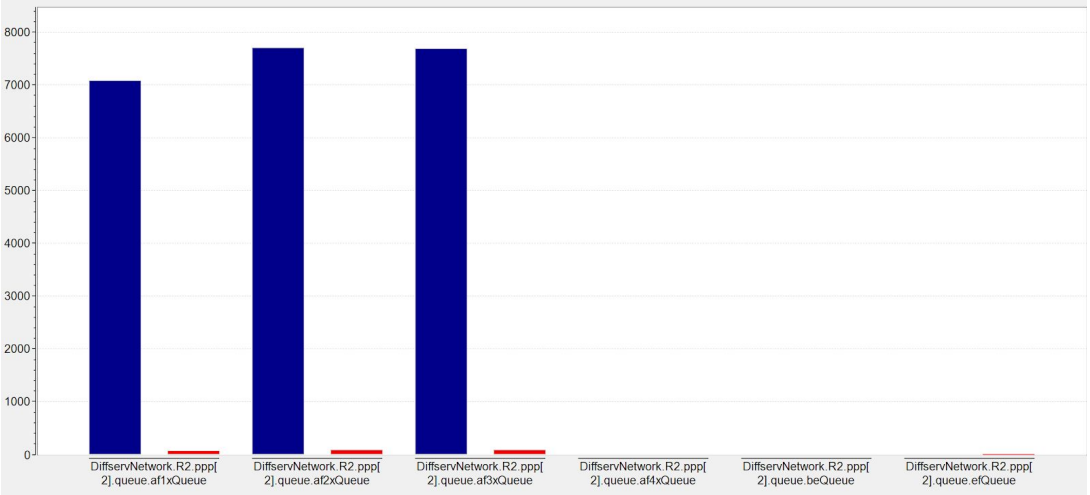

Figure 21. Packets dropped for each queue in Experiment 1.6.

4. Discussion

After having performed simulations to analyze how our network setup allocates resources using different priorities on forwarding classes compared to the default BE service, we concluded that the use of a DiffServ network outperformed the BE service as it was more effective in delivering packets more effectively in an efficient manner.

For Experiments 1.1 to 1.4, a decreasing amount of weights was used on each Service Level Agreement (SLA) (highest weight on AF1, AF2, AF3, lowest weight on AF4). This caused the greatest packet loss on AF4, and the lowest amount of packet loss on AF1. This is to be expected since a higher weight would mean a higher priority, thus having a lower chance at dropping a packet. Another important discovery is the fact that the packet loss for each SLA on each Experiment (1.1 - 1.4) becomes “increasingly diverse” [4] meaning there is a larger difference in packet loss between each SLA as we move through each Experiment. The reason

for this is due to the fact that we made the weights closer together relatively speaking (10, 9, 8, 7) to (9, 8, 7, 6) to (8, 7, 6, 5). The same analysis can be applied to Average Queue length.

For Experiment 1.5, we had used a WRR weight of 1 for all AF queues. This is a simple simulation setup and the results were as expected. Each SLA experienced roughly an equal amount of packet loss and this makes perfect sense since an equal weighting was given to share bandwidth.

For Experiment 1.6, a different setup was used for the simulation. We had used the EF class instead of the AF4 class. Since the EF class is of higher priority than all other classes as seen in above in Figure 6 above, the EF class showed no packet loss. Once again, similar to Experiment 1.5, we had used equal weights of 1 for the rest of the 3 AF queues (AF1, AF2, AF3) and as such, the packet loss shown by each of these three classes showed little variance.

For Experiment 1.7, since only the BE service was used and all traffic entered a single queue with a buffer size of 400, as already mentioned, there was no data in the scalar files and only the end to end delay was shown in the vector files, which was very high. This makes sense because BE service is not known to be very efficient.

Overall, the results were expected since we were able to predict them with the help of the weights of each queue and the priority queuing.

5. Contributions

Jack	Dhaimil
Section 1.1, Section 2.4, Section 4, Performed the simulation on omnet++ with Dhaimil and screenshotted the result for section 3 and added by both of us. Formatting and proofreading.	Section 1.2, Section 4. Jack and I both performed the experiments together. Screenies were taken after each experiment and added by both of us to Section 3. I wrote small descriptions for each result. Described simulation/emulation setup and experiment parameters from Sanjya Ramroop's paper describing different scheduling scenarios [4]. Formatting and proofreading.

6. References

[1] Wikipedia, “Queueing theory” [Online]. Available:
https://en.wikipedia.org/wiki/Queueing_theory
[Accessed: 03-Apr-2019].

[2] “Queuing and Scheduling”. [Online]. Available:
https://connex.csc.uvic.ca/access/content/group/73785f13-ba2a-4175-b5f8-e64211790433/Class%20Slides/3_2_queueing_scheduling.pdf.
[Accessed: 03-Apr-2019].

[3] MetanoiaInc, ”How Do Schedulers in Routers Work? Understanding RR, WRR, WFQ, and DRR Through Simple Examples” 9-Mar-2016. [Online]. Available:
https://www.youtube.com/watch?v=LHpxfwYTDY8.
[Accessed: 03-Apr-2019].

[4] Sanjay, “FINAL REPORT,” 24-Sep-2010. [Online]. Available:
http://www.eng.uwi.tt/depts/elec/staff/rvadams/sramroop/Report_parts/Sanjay%20Ramroop%20807004374%20ECNG%203020%20Report%20April%204th%20Final.pdf.
[Accessed: 02-Apr-2019].