Differentiated Service Queuing Disciplines in ns-3

Vahab Pournaghshband

Robert Chang

Mahdi Rahimi

The Network and Security Research Laboratory
Computer Science Department
California State University, Northridge

Network Simulator 3 (NS3)

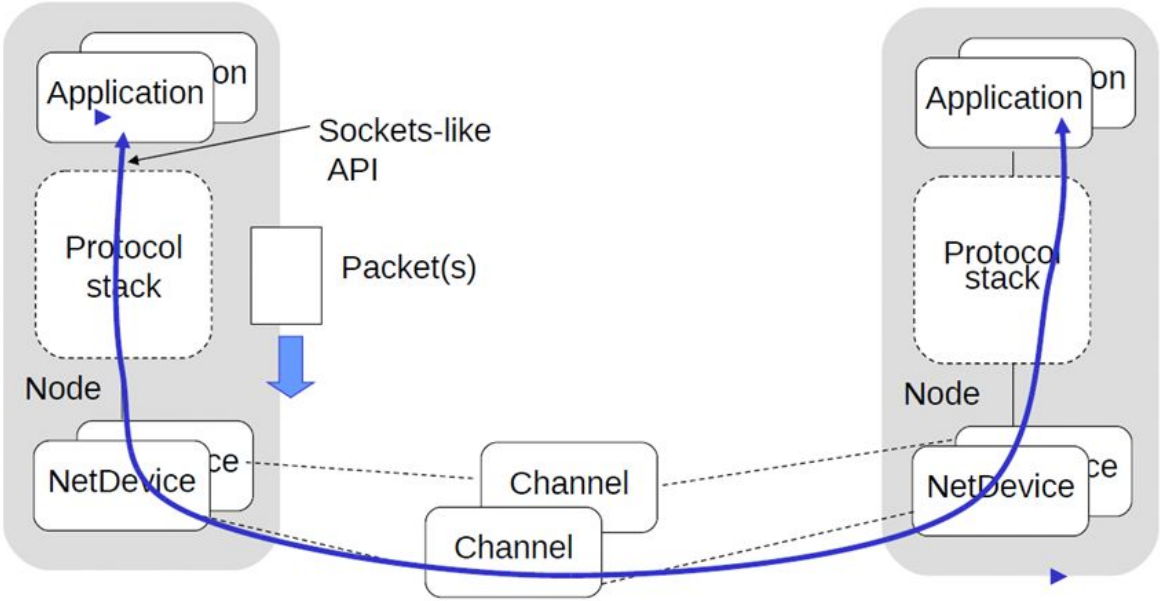
- A discrete-event open source simulator for networking research and education.
- Completely different from predecessor ns2, aiming to be easier to use and more ready for extension.
- ns3 core is written entirely in C++

Network Simulator 3 (NS3)

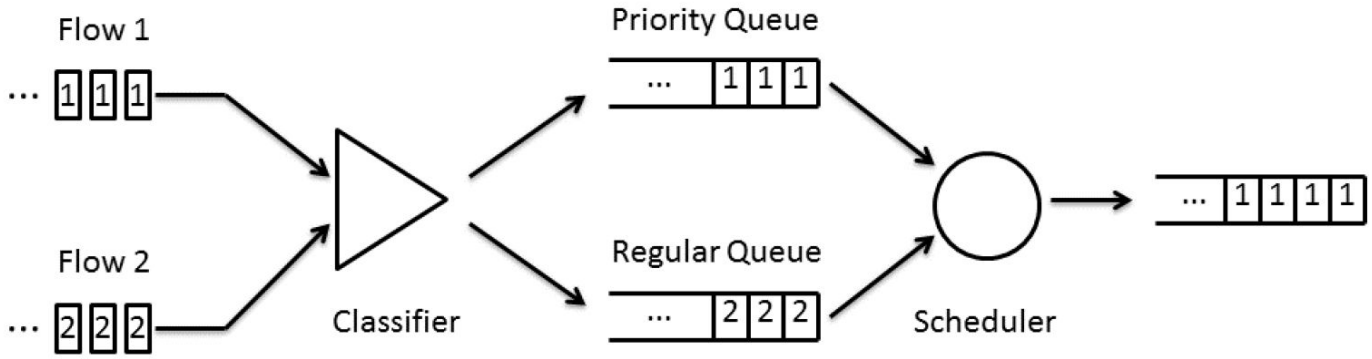
- Sophisticated simulation features included
  - Extensive parameterization system
  - Configurable embedded tracing system, with standard outputs to text logs or PCAP (tcpdump/wireshark)
- Object oriented design for rapid coding and extension
  - Automatic memory management
  - Object aggregation/query for new behaviors & state
    - E.g., adding mobility models to nodes
- Models true IP stack, with potentially multiple devices & IP addresses on every node
- BSD lookalike, event based sockets API

Network Simulator 3 (NS3)

Looks just like IP architecture stack

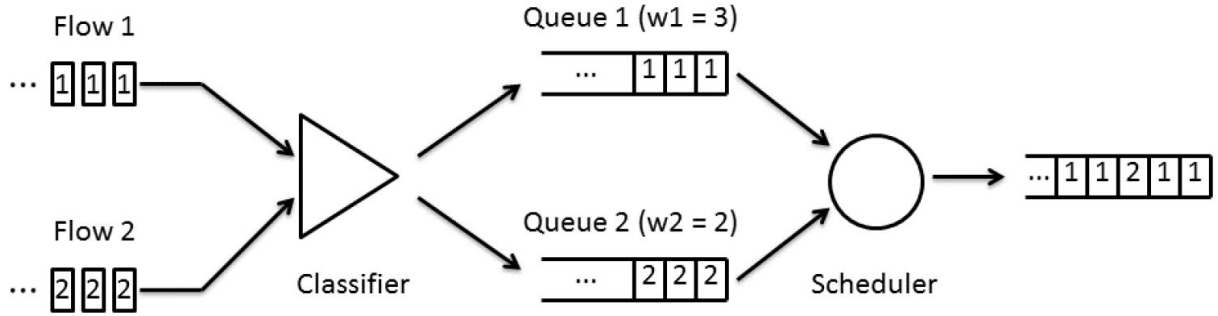


Strict Priority Queueing



Packets from flow 2 cannot be sent until the priority queue is completely emptied of packets from flow 1

Weighted Round Robin

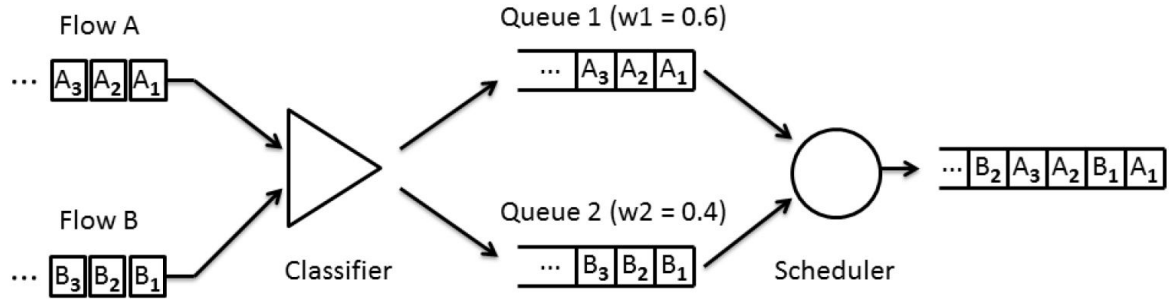


$$w_i^* = \frac{w_i}{L_i}$$

$$\left\lceil \frac{w_i^*}{w_{min}^*} \right\rceil$$

Because packets sent are rounded up, each round two packets will be sent from flow 1 and one packet from flow 2

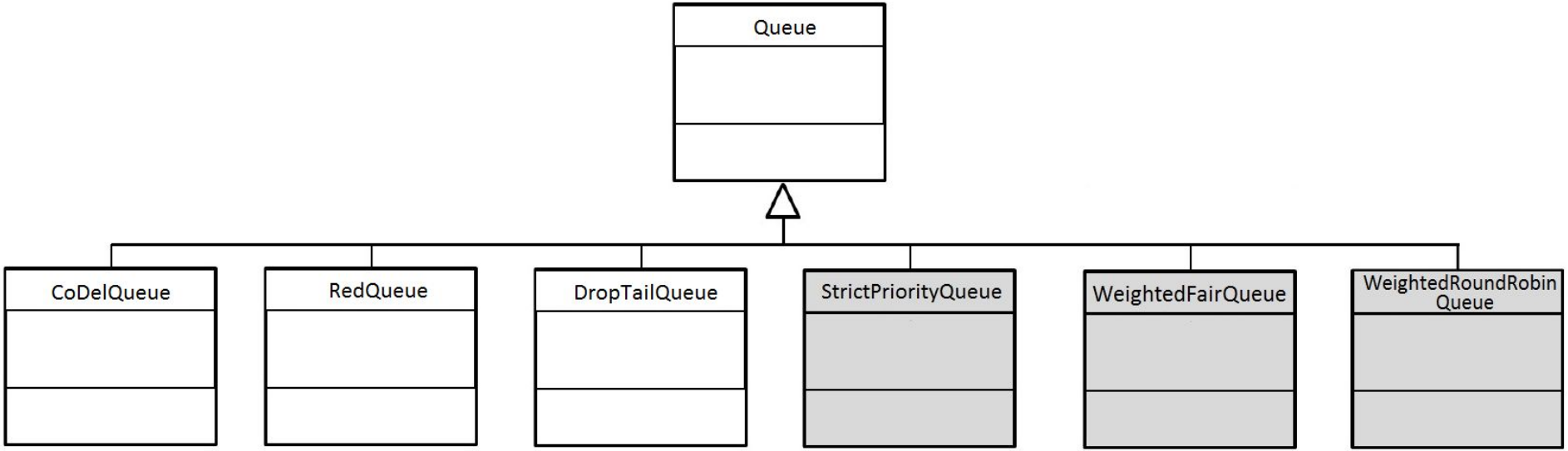
Weighted Fair Queuing



Packet	Finish Time
A_1	35
A_2	60
A_3	80
B_1	50
B_2	90
B_3	110

Packets are sent in the order determined by their virtual finish times

Design


Queue
CoDelQueue
RedQueue
DropTailQueue
StrictPriorityQueue
WeightedFairQueue
WeightedRoundRobin Queue

Design

- The Queue API has three main public members related to functionality that must be implemented by inheriting classes:
  - Enqueue()
    - takes a packet as an argument, attempts to queue it, and indicates whether the packet was successfully queued or dropped
  - Dequeue()
    - takes no arguments, attempts to pop the next scheduled packet, and returns the packet if successful or an error otherwise.
  - Peek()
    - takes no arguments and returns the next scheduled packet without removing it from the queue.
- In the Point To Point module
  - PointToPointNetDevice passes outgoing packets to Queue::Enqueue() when it has finished processing them.
  - PointToPointNetDevice calls Queue::Dequeue() when the outgoing link is free and begins transmitting the returned packet.
- Our modules were built specifically with Point To Point in mind, but can be included with any ns-3 module that utilizes Queue.

SPQ Design

- SPQ has two internal queues, Q1 and Q2, Q1 is the priority queue and Q2 is the default queue.
- DoEnqueue()
  - Calls the function Classify() on the input packet to get a class value.
  - Classify() checks if the packet matches any of the priority criterion and indicates priority queue if it does or default queue if it does not.
  - The packet is pushed to the tail if there is room in the queue; otherwise, it is dropped.
- DoDequeue()
  - Attempts to dequeue a packet from the priority queue.
  - If the priority queue is empty then it will attempt to schedule a packet from the regular queue for transmission.

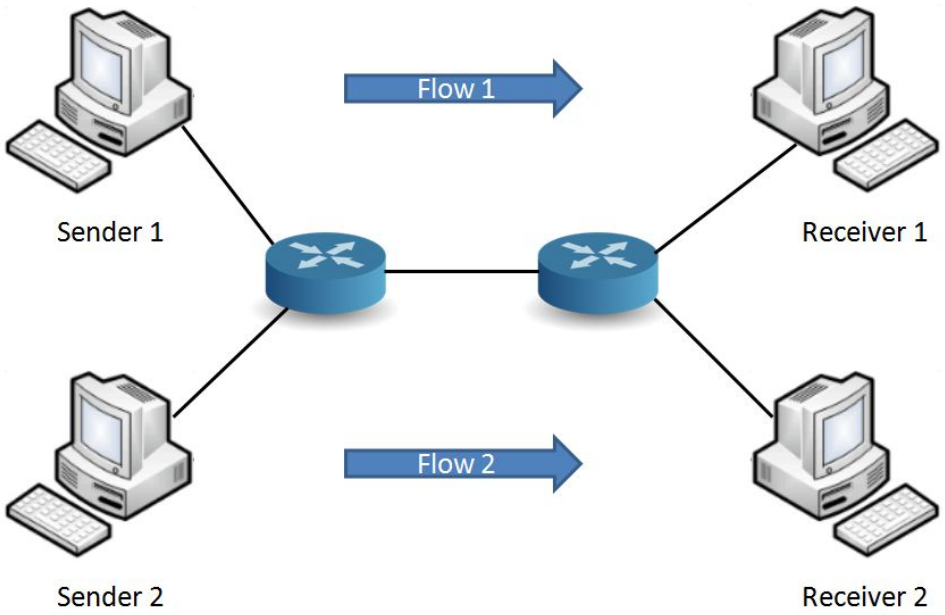
WFQ Design

- Our class based WFQ assigns each packet a class on its arrival.
- Each class has a virtual queue with which packets are associated. For the actual packet buffering, they are inserted into a sorted queue based on their virtual finish time values.
- Each Class (and queue) is assigned a weight, represented by a floating point value.
- A WFQ's scheduler calculates the time each packet finishes service under GPS (virtual finish time) and serves packets in order of that.
- DoEnqueue()
  - Calls Classify() on input packets to get a class value.
  - Classify() returns the class index of the first matching criteria, or the default index if there is no match. This class value maps to one of the virtual internal queues, if the queue is not full the packet is accepted, otherwise it is dropped.
  - The packet's virtual finish time is calculated.
  - The packet is inserted into the sorted queue.
- DoDequeue()
  - Pops the packet at the head of priority queue.
  - This packet has the minimum finish time number.

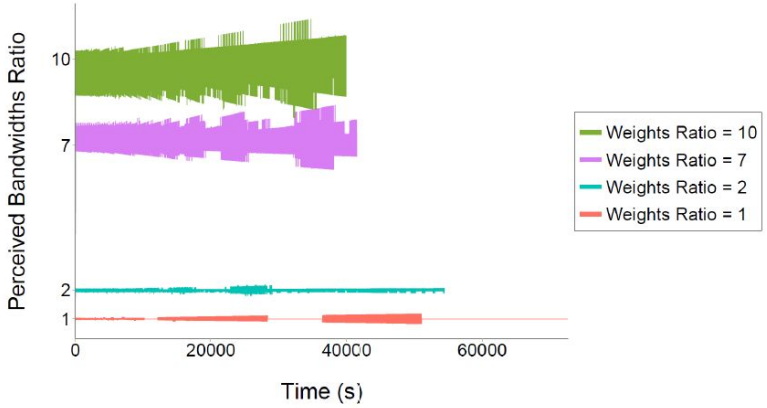
WRR Design

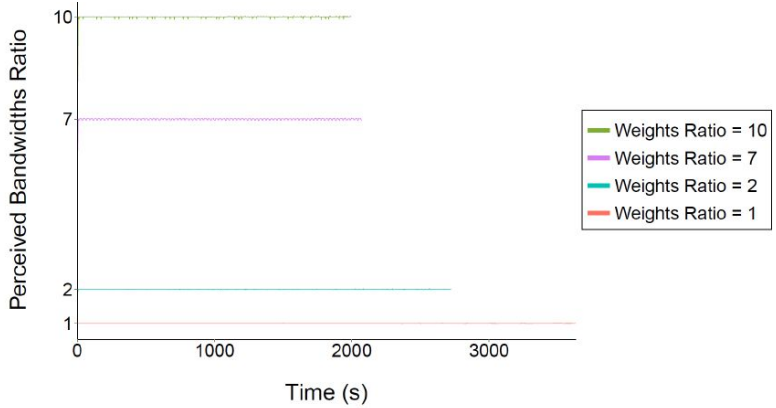
- WRR has the same number of internal queues, assigned weight representation, and classification logic as WFQ.
- The weight must be first normalized with respect to packet size.
  - In an environment with variably sized packets, weighted round robin needs to assign a mean packet size for each queue.
  - These parameters are identified by the prior to the simulation in order to correctly normalize the weights.
- CalculatePacketsToBeServed()
  - Before the start of the simulation, determines the number of packets sent from each queue in each round of service.
- DoEnqueue()
  - Uses Classify() to find the class index of the incoming packets
  - Then puts them in the corresponding queue.
- DoDequeue()
  - Checks an internal counter to track how many packets to send from the queue receiving service.
  - Each time a packet is sent the counter is decremented.
  - If the counter is equal to zero or the queue is empty DoDequeue() marks the next queue in the rotation for service and updates the counter to the value previously determined by CalculatePacketsToBeServed().

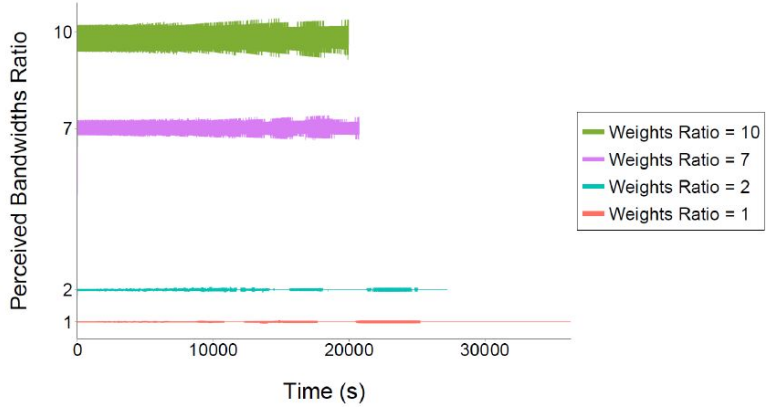
Experimental Setup

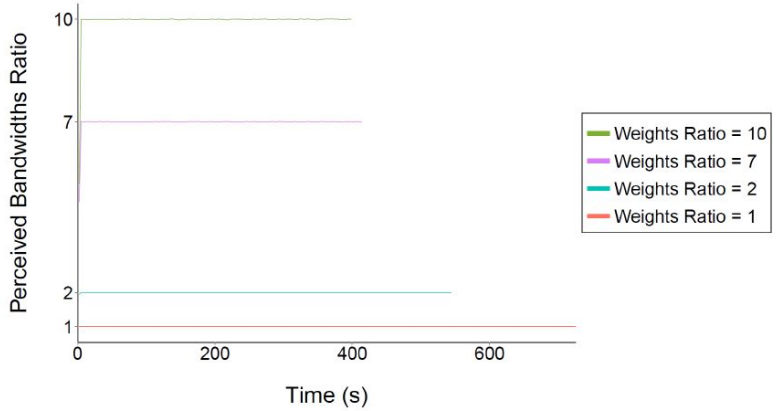


Weighted Fair Queueing Results

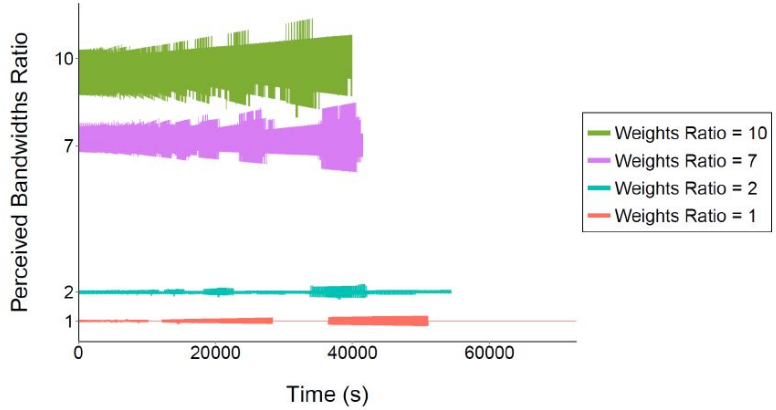


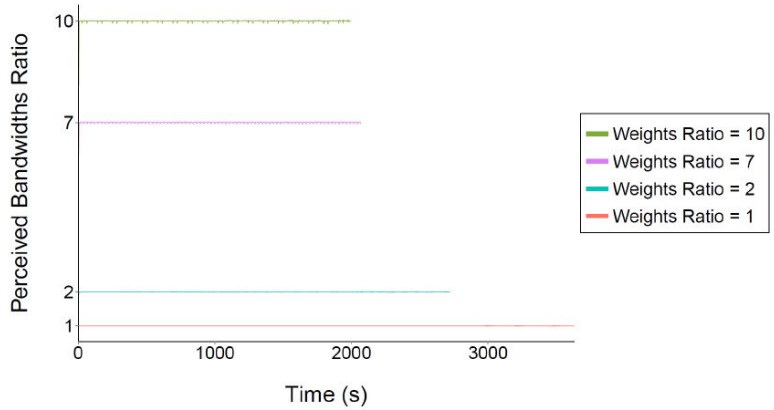


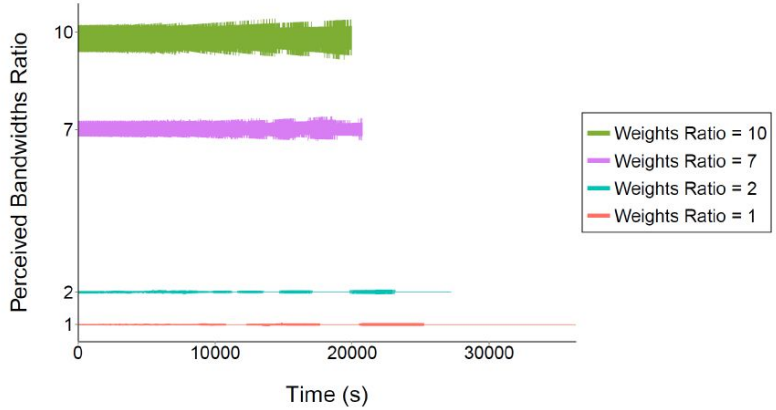


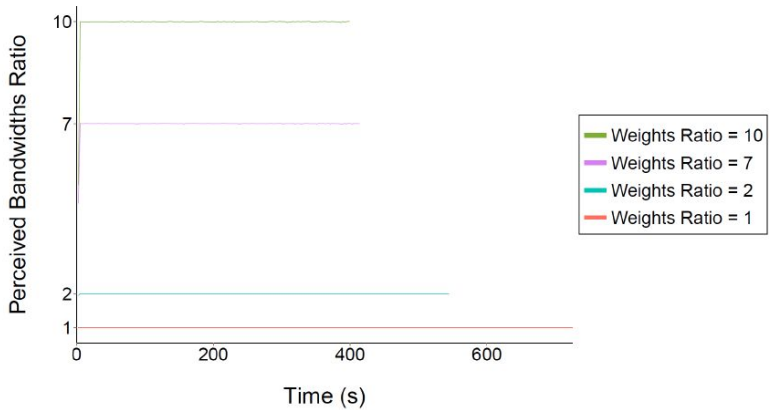


Weighted Round Robin Results









Initialization and Deployment

```
// Installing WFQ on the node N1
PointToPointHelper p2p;
NetDeviceContainer N1N2_d = p2p.Install (N1N2);

ObjectFactory m_queueFactory;
m_queueFactory.SetTypeId ("ns3::WeightedFairQueue");
m_queueFactory.Set ("FirstWeightedQueueMaxPackets", UintegerValue (firstWeightedQueueSize));
m_queueFactory.Set ("SecondWeightedQueueMaxPackets", UintegerValue (secondWeightedQueueSize));
m_queueFactory.Set ("SecondQueuePort", UintegerValue (secondQueuePort));
m_queueFactory.Set ("FirstWeight", DoubleValue (firstWeight));
m_queueFactory.Set ("SecondWeight", DoubleValue (secondWeight));
m_queueFactory.Set ("LinkCapacity", DoubleValue (linkCapacity));

Ptr<Queue> queueN1 = m_queueFactory.Create<Queue> ();
Ptr<PointToPointNetDevice>  devN1 = NetDeviceDynamicCast(N1N2_d.Get(0));
devN1->SetQueue (queueN1);
```

Usage

First method: XML file

```
<acl_list><acl id=aclFirstClass><entry>
      <source_address >10.1.1.0</source_address>
      <source_address_mask >0.0.0.255</source_address_mask>
      <source_port_number >23</source_port_number>
      <destination_address >172.16.1.0</destination_address>
      <destination_address_mask >0.0.0.255</destination_address_mask>
      <destination_port_number >23</destination_port_number>
      <protocol>TCP</protocol>
    </entry></acl></acl_list>
```

```
<class_list>
  <class id="class1" acl_id="aclFirstClass">
    <queue_size >256</queue_size>
    <weight >0.875</weight>
  </class>
</class_list>
```

Second method: Text file

```
access-list access-list-id
  [protocol] [source_address] [source_address_mask]
    [operator [source_port]]
  [destination_address] [destination_address_mask]
    [operator [destination_port]]

access-list aclFirstClass TCP 10.1.1.0 0.0.0.255 eq
23 172.16.1.0 0.0.0.255 eq 23
```

```
class [class_id]
        bandwidth percent [weight] queue-limit
        [queue_size]
class class1 bandwidth percent 0.875 queue-limit 256
```

```
class-map [class_id]
        match access-group [acl_id]
class-map class1  match access-group aclFirstClass
```

Conclusion and Future Work

- Design and implementation of three modules in ns-3
  - Strict Priority Queuing
  - Weighted Fair Queuing
  - Weighted Round Robin
- Future Work: A large amount of overlapping functionality between the three queues.
  - A stateful classifier and scheduler
  - A general framework for any queueing disciplines