Computer Networks Design – Lab 2

As you may have noticed in the previous lab, the Switch has queues. These queues and where to put them were out of the scope of the first lab (but will be revisited in Lab 3). When a frame is received by a switch, it must be processed in one way or another.

For example, a **real** switch *with input queues* places the frames inside the port's queues and waits until the head of the line finishes its transmission. A processed frame is first *classified* to one of the switch's output ports (according to the MAC table). After this assignment, assuming no other frame is transmitted to that port, the frame is transmitted.

A **real** switch *with output queues* has a queue for each output. When receiving a frame, the switch immediately assigns it to a destination port according to its MAC table and puts it in the appropriate queue.

In this lab, you will implement a simulation of both and a simulation of virtual output queues and compare their performance in terms of finishing times and blocking percentage.

To ease the simulation, we will slightly "tweak" the behavior of some components from the previous lab. Accordingly, we recommend backing up your previous lab (e.g., by saving your files into a *private* Git project).

Furthermore, for the "tweaked" objects, we recommend creating new classes that inherit from the previous—for example, HostLab2 would be a subclass of Host, SwitchLab2 would be a subclass of Switch, etc.

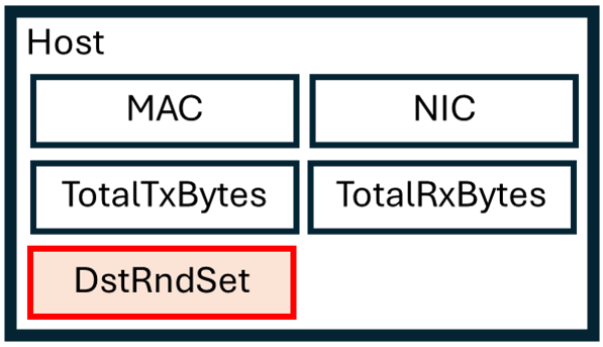
Updating Component Behavior

Host

Previously, the Host object created L2 messages for random destinations. In this lab, the Host will randomly select its destination from some *given set of destinations*. In other words, the new host has the following new field:

1. **DstRndSet:** A set of MAC addresses (or Object IDs, if it is easier to program) from which the Host is allowed to draw (uniformly) destinations.

Visualization:



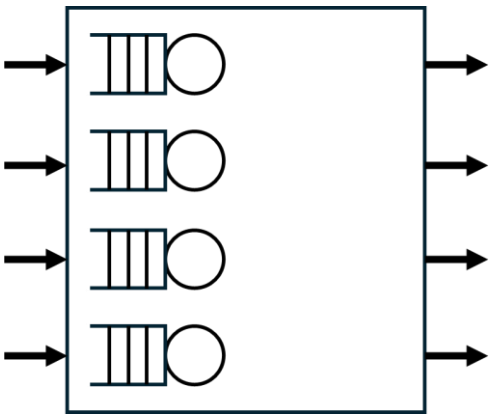
In red – the update. A few notes before continuing:

- How can we let each host select its destination deterministically?
- If you happen to have queues on the Links, remove them. Now, it is a good point to ensure you have queues at each Host's output (without the fluid model flag since the host does not use it). Don't forget to ensure that the "isBusy" flags you might have missed are implemented in the Host, too.
- Tip for testing your queues – repeat Test 1A from the previous lab with high message rate creation and low rate/high propagation links to see if the queues are filled and emptied one at a time and in a FIFO manner.

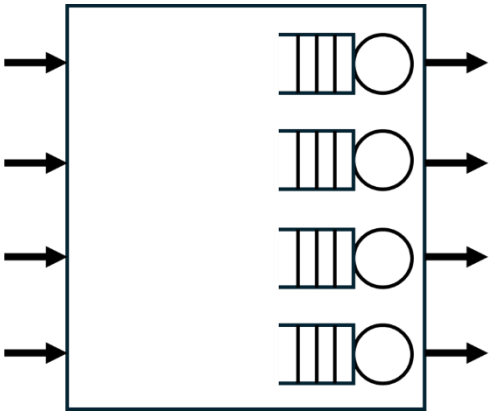
Switch

We will add the Switch object queues. You may implement the queues as you wish (e.g., using the queue or heap data structures). We will focus on the following three:

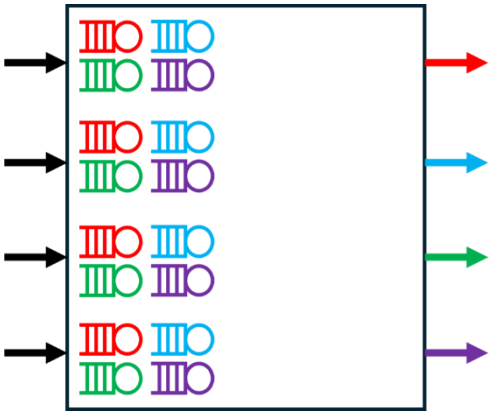
1. **Input Queues** – the queues are located in the Switch's input. When a packet arrives at the Switch, it is automatically placed in the queue. When the queue is not empty, and no frame is processed, the Switch takes the top of the queue and classifies it (i.e., determines its output). If the output is available (i.e., there is no transmission in progress to that output), it transmits the frame.
   Otherwise, the frame waits until the output is available.
   Illustration:



2. **Output Queues** – the queues are located in the Switch's output. When a packet arrives at the Switch, it is automatically classified and placed in the determined output's queue.
   Illustration:



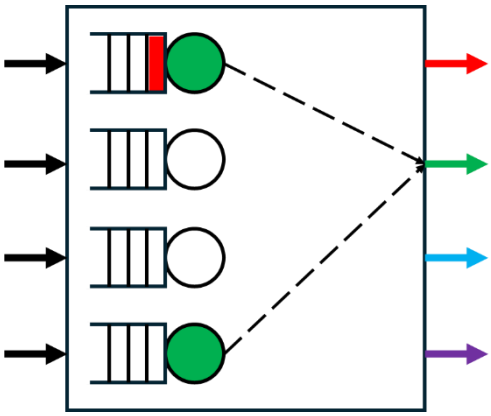
3. **Virtual Output Queues** – Each input has a queue designated for each output. For example, if there are 4 inputs and 8 outputs, each output input has 8 queues or 32 queues in total.
   When a packet arrives at the Switch, it is automatically classified and placed in the determined input's output queue.
   Illustration:



Now, we will discuss updating the Switch object. The Switch will now have the following new fields:

1. **isFluid flag:** a flag indicating whether or not the Switch object operates over a fluid model (will be used in the next lab). When set to "False" (the default), the Switch operates over frames.
   *Fluid model is **not** implemented in this lab.*
2. **Q_Type:** a field controlling which queue type to use – input, output, or virtual output.
3. **Queue:** queues placed at the switch according to "Q_type." Notice that the number of queues can be either the number of inputs, outputs, or their product.
   For statistics, maintain the last departure time for each queue.
4. **ScheduleAlg:** a field controlling the scheduling discipline used by the Switch object (will be implemented in the next lab). The default is "FIFO."

5. **TotalHoLTime:** Head of Line (HoL) Blocking is an undesired phenomenon where the first message in the queue holds the entire queue AND is not processed (e.g., not transmitted). Note that the queue length may be larger than 1.
   For example, see the following illustration:

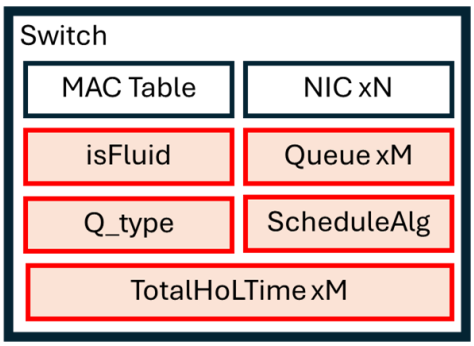


   In this illustration, the 1st and 4th queues wish to send messages (top of the queues, in green) to the 2nd output. However, when the message transmission model is not fluid, only one can be sent to the 2nd output, and the other message waits inside (and holds!) the queue. If, for example, the 4th queue is scheduled to send its message via the 2nd output, the 1st queue suffers from HoL Blocking since its top is not processed despite being the top.

Note: do not confuse this phenomenon with similar phenomena such as deadlock or race conditions.

   For each queue, the Switch collects the (total) time it observed a HoL Blocking. That is, "TotalHoLTime" is the sum of times the queue top was not transmitted.
   At the end of the simulation, the percentage of time spent due to HoL Blocking should be printed if the "HoL printing flag" is ON.

Visualization:

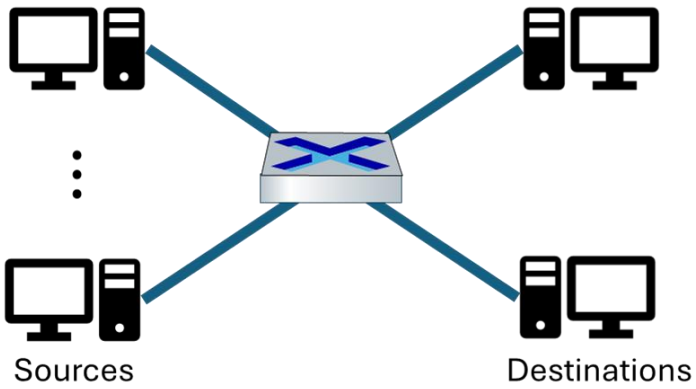


Before we move to testing, make sure you understand the following:

- Can you order “input queues,” “output queues,” and “virtual output queues” in terms of HoL Blocking times? That is, which has the highest and lowest HoL Blocking times?
- What is the disadvantage of using output queues? ***Hint:*** would you need any processing power at the output if you used input queues?
- What is the disadvantage of using virtual output queues?

Testing Part A

Test 1 – Comparing HoL Blocking Times



Construct the above topology, where the number of sources is random between 3 and 6. Each source randomly selects one of the 2 destinations for each L2 Message.

Put the "HoL print flag" ON and let each Host create a flow of 10000 L2 Messages. Print the percentage of time spent due to HoL Blocking and each queue’s last departure time.

This test should be used on all three queue types *using the same random seed*.

- What should the link rates and propagation delays be so your queues will not explode? ***Hint:*** At what rate do the L2 Messages come?
- Which architecture has the highest HoL Blocking percentage in practice? Which one has the lowest?
- What are the last finishing times of the input, output, and virtual output queues?