CS-534: Packet Switch Architecture
Spring 2004

Exercise Set 7:
Input Queueing and Crossbar Scheduling

Assigned: Wed. 21 (Fri. 23) April 2004 (week 7) - Due: Wed. 28 April 2004 (week 8)

7.1 Saturation Throughput in a 3x3 Input Queued Switch

As we saw in class, a simplified formulation for computing the saturation throughput of a 2x2 input queued switch is the following. The head-of-line cells of the two input queues may have the following destination combinations: *(i)* "00", meaning that they are both destined to output 0; or *(ii)* "01", meaning the first is destined to output 0 and the other is destined to output 1; or *(iii)* "10"; or *(iv)* "11". If we assume that the switch is saturated, the queues are always non-empty. If we assume that arrivals are i.i.d. (independent identically distributed) with uniformly distributed destinations, then the probability of each of the above combinations is 0.25, and it is independent of the combination in the previous time slot and of the service decision made in the previous time slot. Combinations 01 and 10 yield 2 outgoing cells per time slot for the two outputs (average throughput 1.0 per output), while combinations 00 and 11 yield only 1 outgoing cell per time slot for both outputs (average throughput 0.5 per output). Hence, the average (saturation) throughput per output over a long time window will be: 0.25*0.5 + 0.25*1.0 + 0.25*1.0 + 0.25*0.5 = 0.75.

Using the same technique, calculate the saturation throughput of a 3x3 input queued switch. The head-of-line cells of the three input queues may form 27 combinations. List these combinations; compute the probability of each; compute the average per-output throughput of the switch in each of these cases (and briefly explain); and, finally, compute the average saturation throughput.

7.2 Hand-Simulation of a CIOQ Switch with a Speedup of 2

In this exercise, you will simulate by hand the operation of a 3x3 CIOQ (combined input-output queueing switch) with a speedup factor of 2 for its internal crossbar. Each of the 3 input buffers of the switch contains 3 per-output queues (VOQ - virtual output queues). At each output of the switch there is a single output queue (no per-priority or per-flow queues, for simplicity).

The simulation will proceed in steps of one "time-slot" each. One time-slot is the time it takes for one cell to arrive through an input port or one cell to depart through an output port. Since the crossbar uses a speedup of 2, the crossbar operates **twice** per time-slot: in the first phase of each time-slot it can transfer up to one cell from each input buffer and up to one cell to each output queue, and in the second phase it can again transfer up to one cell from each input buffer (same or different queue in the buffer) and up to one cell to each output queue; each time-slot contains both of these phases.

Use the "form" below to simulate the operation of the switch. In this form, the first paragraph --following the number "(8)"-- represents the events, the state, and the operation during the 8th time-slot, and the second paragraph --following the number "(9)"-- represents the 9th time-slot. (The queue state just before the 8th time-slot was created artificially, and may not result from any real sequence of switch inputs and operation steps).

```
(8)
  inputs: a8y; b8y; c8z;
  inQs a: -; a4y,a8y; a5z,a6z,a7z;
  inQs b: -; b5y,b8y; b6z,b7z;
  inQs c: c7x; c6y; c8z;

  xbar 1: c7x; a4y; b6z;
  xbar 2: -; c6y; b7z;

  outQ's: b4x,c7x; a4y,c6y; c5z,b6z,b7z;
  outputs: b4x; a4y; c5z;
(9)
  inputs: a9z; b9y; c9z;
  inQs a: -; a8y; a5z,a6z,a7z,a9z;
  inQs b: -; b5y,b8y,b9y; -;
  inQs c: -; -; c8z,c9z;
```

```
xbar 1: -; b5y; c8z;
xbar 2: -; a8y; c9z;

outQ's: c7x; c6y,b5y,a8y; b6z,b7z,c8z,c9z;
outputs: c7x; c6y; b6z;
```

At the beginning of each time-slot, we list the cells that arrived from the 3 inputs during the previous time-slot; up to one cell may have arrived from each input. The **inputs** of the switch are called **a**, **b**, and **c**; the **outputs** of the switch are called **x**, **y**, and **z**. Each **cell** has a unique name, formed as follows: we concatenate the name of the input through which the cell arrived, the name (number) of the time-slot at the beginning of which the cell was fully received, and the name of the output to which the cell is destined. Thus, at the beginning of time-slot 8 three new cells have arrived: a8y, b8y, and c8z; a8y has arrived through input a and is destined to output y; b8y has arrived through input b and is also destined to output y; c8z came from c and goes to z.

After listing the arriving cells, we list the contents of the 9 input queues. Line "inQs a" lists the contents of the 3 VOQ's contained in input buffer a; line "inQs b" contains the queues of input buffer b, and similarly for "inQs c". Each line contains first the queue of cells destined to x, then the queue for y, and lastly the queue for z. The head of each queue is on the left, and the tail is on the right. The cells that just arrived from the inputs, at the beginning of the current time-slot, are assumed to have already been enqueued into the proper queue, and are listed here.

Next, we list the crossbar activity during the two phases of the current time-slot: the line "xbar 1" lists the phase-1 transfers, and the line "xbar 2" is for phase-2. In each phase, we list the cell that is transferred to output-queue x, if any, followed by the cell transferred to y, if any, followed by the cell going to z, if any. Obviously, each of these cells must come from the head of a VOQ; the cells transferred during phase-1 are assumed to have been dequeued right away, so phase-2 sees the updated heads of the VOQ's.

Below the crossbar activity, we list the contents of the 3 output queues; again, the head is on the left and the tail is on the right. The cells just delivered by the crossbar, during both phases of the current time-slot, are assumed to have already been enqueued into these queues. After that, we list the departing cells, which obviously are the heads of each output queue (when non-empty). Note that this simulation style is able to transfer an arriving cell to its output queue and from there to the departure port, all within a single time-slot, if the proper queues were in a favorable state and the crossbar made favorable decisions. In a real switch, these steps normally take many clock cycles, corresponding to multiple time-slots, but these activities are pipelined, so that the net link throughput is 1 cell per time-slot, and the net crossbar throughput (in the case of speedup of 2) is 2 cells per time-slot per port.

(a) Simulate, by hand, the above switch, using the above "form", for the following input traffic. Assume that all queues were empty before time-slot 1, and all inputs remain idle after time-slot 6. continue your simulation until all queues drain out.

```
(1) inputs: a1x; b1x; c1x;
(2) inputs: a2x; b2x; c2x;
(3) inputs: a3x; b3x; c3x;
(4) inputs: a4x; b4x; c4x;
(5) inputs: a5y; b5z; c5y;
(6) inputs: a6z; b6z; c6z;
(7) inputs: -; -; -;
```

Assume that, in each phase of each time-slot, the crossbar is scheduled according to a maximal (not necessarily maximum) matching, found by following the "Lowest Occupancy Output First" Algorithm (LOOFA) seen in class (Krishna, Patel, Charny, Simcoe: IEEE JSAC, June 1999). Observe that the switch is work-conserving, i.e. each output is busy transmitting a cell in (at the end of) each and every time-slot when there is at least one cell destined to that output *anywhere* inside the switch.

(b) Repeat your simulation, under the same input traffic pattern above, but using different crossbar scheduling decisions. Do continue using maximal (not necessarily maximum) matchings. However, change in an arbitrary and "bad" manner the matchings that the crossbar scheduler "happens" to make, so that some queues "happen" to be served more frequently than others, in such a way that the switch ends up not being work-conserving anymore (I believe that this can be made to "happen", at least for cell b5z, for one time-slot; I am not sure if it can happen any more than that --may be you want to play with other input patterns or with larger switches, and see if you can make it happen for longer delays or in a sustained manner that affects throughput as well, not just delay... -:)).