January 31st, 2012

Exam of Switching architectures

Rules for the exam. It is forbidden to use notes, books or calculators. When needed, use approximations.

Time available: 70 minutes.

PROBLEM A

Consider a 4 × 4 Cantor network.

1. Draw it.

2. Discuss its properties in terms of blocking.

3. Describe the routing algorithm to configure the network.

4. Show the connections to support the following input-output couples: 1 → 4, 2 → 3, 3 → 1 and 4 → 2.

5. Apply Lee’s method to evaluate the blocking probability of the network.

6. Is the blocking probability evaluated by Lee’s method compatible with the blocking properties of the network? Why?

7. Is there any alternative switching architecture based on 2 × 2 modules, with the same blocking properties of Cantor network, but lower complexity? If yes, which one?

PROBLEM B

Consider an input queued switch of size N × M supporting both unicast and multicast traffic. Assume that each input is equipped with just 2 queues: one queue for broadcast packets and one queue for multicast (but not broadcast) traffic, including unicast traffic.

Consider a scheduling algorithm that does not allow fanout-splitting and serves broadcast traffic at highest priority. As a reminder, the fanout set of a multicast packet is the set of its destination ports.

1. Show an example (if any) of admissible arrival pattern for which the scheduler achieves the maximum throughput.

2. Show an example (if any) of admissible arrival pattern for which the scheduler does not achieve the maximum throughput.

3. What are the performances of the switch fed by unicast traffic only?

4. Describe in pseudo-code the scheduling algorithm, using the notation below.

At each timeslot, let B[i] be the size of the queue for broadcast packets at input i. Let M[i] be the size of the queue for multicast/unicast packets at input i. Assume that function destInMCQueue(j,i) returns true iff output j belongs to the fanout set of the packet at the head of the multicast queue M[i]. Let X be the matrix describing the switching configuration chosen in the current timeslot, based on the state of the queues. More precisely, X[i][j] is a boolean variable, which assumes the value true iff the crosspoint from input i to output j is active, i.e. a packet must be sent from input i to output j in the current timeslot.

PROBLEM C

Consider a generic frame scheduling policy for input queued switches in which the arrival rates are known and the scheduling decision are taken oblivious of the queue state.

1. Prove that the average delay grows as O(N) in order sense, for one of the possible frame scheduling algorithms.

2. Is it possible to improve the average delay in order sense? How?

If needed, for a slotted Geom/Geom/1 queue with arrival probability λ and service probability μ, the average delay is:

\[ W_{Geom/Geom/1} = \frac{\eta}{\lambda(1 - \eta)} \quad \text{with} \quad \eta = \frac{\lambda(1 - \mu)}{\mu(1 - \lambda)} \]

If needed, for a slotted M/D/1 queue with binomial \((N, \rho/N)\) arrivals per slot, the average delay is:

\[ W_{M/D/1} = 1 + \frac{\rho}{2(1 - \rho)} \]
SOLUTIONS

Problem A

The corresponding Cantor network is in the figure.

The algorithm to configure the network is a just a breadth-first search in the corresponding pi-graph, considering only the available links.

The reductions in the pi-graph is shown in the figure, where only the links for which a contention may occur are shown. Note that the edges from the input to each Benes network and from each of them to the outputs are always available.

Let $\rho$ be the average load at any input. Now: $a = 4\rho/8 = \rho/2$, $b = 1 - (1 - a)^2$ and finally the blocking probability is

$$P(\text{blocking}) = b^4 = (1 - (1 - \rho/2)^2)^4 = \rho(1 - \rho/4)^4$$

Note that for this simple case it is possible to improve the approximation by using $\rho = 3\rho/8$ since, when adding a new circuit, the average number of pre-existing circuits is $3\rho$ and not $4\rho$ as used in evaluating $a$ above.

In both cases, when $\rho = 1$, the blocking probability appears to be larger than 0 (precisely, around 0.15-0.30), even if the network has been designed as strictly non-blocking. This is due to the approximated method adopted.

An example of alternative network is a classical Clos network, built with $2 \times 2$ modules. The final complexity will be:

$$C_{4 \times 4} = 2C_{2 \times 3} + 3C_{2 \times 2} + 2C_{3 \times 2} = 4C_{2 \times 3} + 3C_{2 \times 2} = 8C_{2 \times 2} + 3C_{2 \times 2} = 11C_{2 \times 2}$$

which is slightly better (one less module) than the Cantor network proposed above. The control algorithm is the same.

Problem B

The switch is able to achieve the maximum throughput when all the inputs are receiving broadcast packets with probability $\leq 1/N$ each.

The switch is not able to achieve the maximum throughput for generic multicast packets, due to the no-fanout splitting policy. For example, consider input 1 receiving packets with fanout set $\{1, 2\}$ (with prob. 0.5 per timeslot) and $\{3, 4\}$ (with prob. 0.5 per timeslot) and input 2 receiving packets with fanout set $\{1, 3\}$ (with prob. 0.5 per timeslot) and $\{2, 4\}$ (with prob. 0.5 per timeslot). Even if the traffic is admissible, at most one packet is served per timeslot; hence, half of the packets are lost.

Under unicast traffic only, the switch behaves exactly as a single queue per input switch. Under admissible uniform i.i.d. Bernoulli arrivals, the maximum throughput is around 58%.

See next page for the pseudocode.
// initialize the data structures
for j=1...M  // for each output port
    output_reserved[j]=false
for i=1...N  // for each input port
    X[i][j]=false

// first, try to serve broadcast packets
for i=1...N  // for each input port
    if (B[i]>0)
        // found broadcast packet with all available outputs
        for j=1...M
            X[i][j]=true
        output_reserved[j]=true  // (not needed)
        return  // ends since switching configuration is maximal

// second, try to serve multicast packets
for i=1...N
    if (M[i]>0)  // check if the mc queue is non-empty
        // check if all the corresponding outputs are available
        multicast_available=true
        for j=1...M  // for each output port
            if (destInMCQueue(j,i))  // j is in the fanout set of the packet
                if (output_reserved[j])
                    // the output is already reserved
                    multicast_available=false
                    break  // useless to continue to check
        if (multicast_available)
            // reserve all the outputs
            for j=1...N  // for each output port
                if (destInMCQueue(j,i))
                    output_reserved[j]=true
                    X[i][j]=true