February 11th, 2013
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
Prove that the weight of a greedy maximal weight matching (GWM) is at least equal to half the weight of the maximum weight matching (MWM). In other words,

\[ W(\text{GWM}) \geq \frac{1}{2} W(\text{MWM}) \]

(1)

In the proof, denote by \( E \) the set of edges in the bipartite graph, by \( G \) the sub-set of edges selected by the GWM, and by \( M \) the sub-set of edges selected by the MWM scheduler.

What is the main consequence of (1) in terms of maximum throughput achievable by GWM?

PROBLEM B
Design a rearrangeable switch of size 600 x 800 using only modules of size 10 x 10, with the aim of minimizing the number of modules.

1. Describe the architecture
2. Compute the total number of modules required
3. Describe the configuration algorithm
4. Write the formula to compute the minimum theoretical number of modules to build the switch and to compare the actual complexity to the optimal one

PROBLEM C
Consider an input queued switch of size \( N \times M \) 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 both multicast and unicast traffic.

Consider a scheduling algorithm that does not allow fanout-splitting and serves broadcast traffic at highest priority.

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 \( \text{destInMCQueue}(j, i) \) returns \( \text{true} \) 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 \( \text{true} \) 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.
HINTS FOR THE SOLUTIONS

Problem A
See class notes.

Problem B
See exercise 8.

Problem C
The switch is able to achieve the maximum throughput when all the inputs are receiving broadcast packets with probability $\leq \frac{1}{N}$ 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%.

```cpp
// 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...N
            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...M // for each output port
                    if (destInMCQueue(j, i)) // j is in the fanout set of the packet
                        output_reserved[j]=true
                        X[i][j]=true
```