1. Consider a switch in which input 1 and 2 each start receiving multicast packets with fanout 2 at the link rate, starting at time 0. The packets arriving at input 1 must be forwarded to outputs 3 and 4, while the packets arriving at input 2 must be forwarded to outputs 4 and 5. Suppose this is a ideal, bus-based switch with queues on the outputs only. After 100 time steps, how many packets have been sent to outputs 3, 4 and 5? How many packets are queued at each of the outputs at this time?

All three outputs will send 100 packets and output 4 will have 100 packets still in its output buffer.

Answer the questions again, assuming the switch is a crossbar with no speedup, in which multicast packets may not be “split” and sent multiple times. Assume that the crossbar treats the two inputs fairly.

In this case, the crossbar would alternate between the two inputs, so after 100 time steps, outputs 3 and 5 will have sent 50 packets, while output 4 will send 100. None of the outputs will have any packets queued, but inputs 1 and 2 will each have 50 packets queued.

Answer the questions again, assuming the switch is a crossbar with a speedup of 1.5 and that can split multicast packets in order to improve throughput.

In this case, the crossbar will let one input send to both of its outputs each time, while the other sends to a single output. With a speedup of 1.5, there are 150 crossbar cycles in the first 100 time steps, so outputs 3 and 5 will have sent 100 packets and have backlog, while output 4 will have sent 100 packets and will have 50 packets in its queue. Inputs 1 and 2 will each have 25 packets in their VOQs for output 4.

2. The simple performance analysis on page 6 implies that for \( \rho = 1 \), we get better performance when the traffic comes from a small number of inputs than when it comes from all inputs. Explain why this is true.

The analysis determines the number of outputs for which there are no packets and uses this to estimate the throughput. If many outputs have no packets destined for them, then the throughput of the switch will be lower than if all outputs have packets trying to reach them. The table in the notes shows that for small values of \( n \), the probability that an output has no packets addressed to it is small. It’s easiest to understand this by considering the case of \( m = 1 \). In this case, one input is supplying packets to all outputs and since the output load is 1, the arriving packets must all have fanout \( n \). Thus, there are never any outputs for which there are no packets. More generally, small \( m \) implies large fanout and the copies of a single multicast packet intrinsically go to different outputs, making the output load more even that it is when the fanouts are small.

3. Page 7 of the notes describes a generic multicast crossbar scheduler that allows multicast packets to be split and sent at different times to different sets of outputs. Describe a specific scheduling algorithm that you think might work well and explain why you think it would work well.
Perhaps the simplest approach one can take is to have outputs select among the requesting inputs at random, and have each input randomly select a packet from among all those with at least one grant. However, this may work ok, but it also seems likely that it may be unfair to inputs with lots of multicast traffic.

A more sophisticated approach is for outputs to select inputs with the largest backlog, where the backlog at an input is defined as the sum of the product of its packets fanout and length in bytes (or something similar). Each input could then select a packet based on the number of the outputs in the packet’s output set for which it has a grant. By having outputs favor inputs with large backlogs, we make it more likely that those inputs with the most multicast packets will be able to get their packets to their outputs using a minimal number of crossbar transfer cycles.

4. In the two pass multicast approach, the routing of multicast packets can be done in one of two ways. In the first way, the input line cards do a lookup that produces an output bit vector specifying the output set for a given multicast packet. This is then sent with the packet to the relays, and each relay assigns its copy to one of the destinations in the set (this requires that each relay be able to determine which copy it received).

In the second approach, the input line card lookup just determines the fanout of the packet and a second lookup is done at the relays to determine the destination for each copy. Each relay has a table entry for every multicast in the system and that table entry tells it how to forward any packet it receives for that multicast. For any particular multicast with fanout $f$, the table entries assigned to two relays $i$ and $j$ may be the same, but only if $|i-j| > f$. This is easy to guarantee using a repeating consecutive destination assignment of outputs to relays. For example, if a given multicast has an output set of {3, 7, 10}, the first relay will be assigned destination 3, the second relay will be assigned destination 7 and the third will be assigned destination 7. And then we repeat, with the fourth relay being assigned destination 3, the fifth being assigned destination 7 and so forth. By assigning destinations to relays in this way, we can guarantee that any packet for this multicast is sent exactly once to each of the destinations.

Discuss the advantages and drawbacks of these two approaches.

The first approach is a good option when the number of ports in the system is not too large (say 32 or less), since in this case the bit vector is small and does not add significant overhead to the packets as they are sent from inputs to relays. However, for larger switches, the extra overhead of the bit vector approach could become a problem (for example, for a 256 port switch the bit vector would add 32 bytes to each packet, adding 50% to the required bandwidth of a crossbar for 64 byte cells).

The second approach has the drawback that it requires a second set of lookup tables, and the lookup tables at the relays must include an entry for every multicast in the system. The low cost of memory makes it feasible to have these large multicast tables in every relay, but it does represent an added hardware cost, and of course the information all these tables must be updated whenever any multicast is added, removed or modified.