CS-534: Packet Switch Architecture
Spring 2004

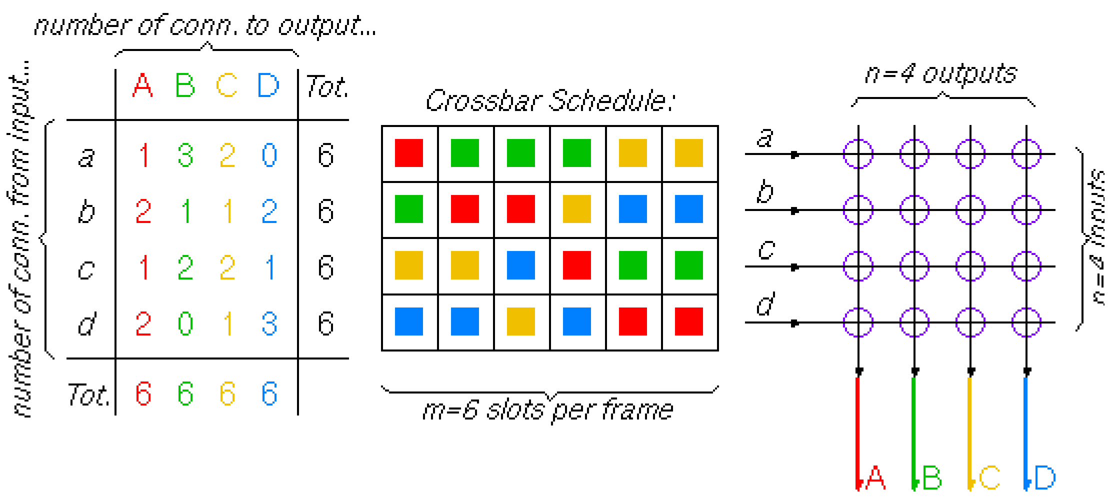
Exercise Set 6: TST Circuit Switch Scheduling

Assigned: Fri. 2 April 2004 (week 6) - Due: Wed. 21 April 2004 (week 7)

6.0 TST Switch Schedule in Colors

In section 4.1 of the course we discussed the need for time-slot interchanges (TSI) in front of the crossbar, in a time-space-time (TST) circuit switch, in order to rearrange the position of the various connections inside the (synchronized) incoming frames, so as to eliminate output contention in the crossbar (i.e. no two connections in similar positions of two different frames have the same outgoing link).

In this exercise set we will study the way in which these input-TSI's should rearrange the incoming connections. To make it easier to think about the problem, we will formulate it using colors, as in figure 1. Colors, in our case, will correspond to outgoing links; in figure 1 there are $n=4$ outgoing links, so there are $n=4$ colors. For simplicity we will assume that the number of incoming links is also n, equal to the number of outgoing links, and that all liks have the same speed, hence the same number of slots in their frames --call this number m.



	A	B	C	D	Tot.
a	1	3	2	0	6
b	2	1	1	2	6
c	1	2	2	1	6
d	2	0	1	3	6
Tot.	6	6	6	6	

🟥	🟩	🟩	🟩	🟨	🟨
🟩	🟥	🟥	🟨	🟦	🟦
🟨	🟨	🟦	🟥	🟩	🟩
🟦	🟦	🟨	🟦	🟥	🟥

Figure 1

Let us call "crossbar schedule" the colored rectangular array shown in figure 1; in this array, the horizontal direction corresponds to time (slots in a frame), the vertical direction corresponds to crossbar inputs, and the color corresponds to crossbar outputs, as discussed above. Our topic is the construction of this schedule; once the "schedule" is set, we know how to configure the crosspoint of the space switch during each time slot of the frame. The schedule corresponds to the frames produced by the input TSI's. Once the schedule is constructed, setting the TSI's so as to generate it is straightforward, so we will not discuss that here. Also, assigning specific connections (circuits) to specific entries in the schedule can be done in *any* random way, as long as an entry of the correct color (output) is used in the correct row (input) of the table. Thus, for our purposes here, all entries of the same color in the same row of the schedule are completely equivalent to each other and interchangeable. We can now formulate the inputs to our problem, and the constraints that the solution sought must satisfy.

For a given switch size n and frame size m, the input to our problem is the n x n array of numbers shown in the left of figure 1. Each entry in this array specifies the number of connections (circuits) on a given incoming link to be switched onto a given outgoing link. Obviously, the sum of the numbers in each row (the total number of circuits on an incoming link) cannot exceed m (the frame size);

similarly, the sum of the numbers in each column (the total number of circuits on an outgoing link) cannot exceed *m*. In one sense, the problem of schedule construction is hardest when the schedule is full, as in figure 1, i.e. when all these horizontal and vertical sums are equal to *m* (in another sense, constructing a full schedule has some advantages --see exercise 6.3 below). The problem to be solved is the construction of a schedule for the given numbers of connections. A schedule is an *n x m* array of colors, as shown above, that satisfies the following constraints:

- the number of entries of each color on each row are equal to the corresponding number of connections given; and
- each color appears at most once in each column of the schedule.

6.1 Schedule Rearrangement when adding new Connections

When adding new connections (circuits) to a (not fully utilized) switch, the crossbar schedule can*not* always be updated by merely adding new entries to it --there are situations where existing entries in the schedule have to be *rearranged* (this corresponds to the so-called "rearrangeably non-blocking" switching fabrics, to be seen later in the course).

(a) To see this, construct the following small scenario. Consider a 3x3 switch (*n=3*) with a frame size of *m=2*. First, set-up a red connection on input *a*, then add a green connection on input *b*, and then another green connection on input *c*. Now, try various scenaria of adding more red or blue connections, in ways such that the new connections require or do not require rearrangement of the first three entries in the schedule.

(b) Can you modify your placement of the first three entries in the schedule of question (a) so that no new addition after that will require rearrangement?

(c) Make some scenaria similar to question (a) for a 3x3 switch (*n=3*) with a frame size of *m=4*.

This exercise 6.1 guides us to consider that algorithms for schedule construction should probably fall in one of the following two categories: either *(i)* the algorithm considers the connections in a random order, but then it must be prepared to rearrange schedule entries made earlier; or *(ii)* the algorithm must start with "global" knowledge of all connections to be made, and must then consider them in some particular "clever" order.

Next, we want to think whether a schedule always exists and work towards an algorithm for constructing a schedule for any given set of numbers of connections (that do not exceed line capacities), under the assumptions in this exercise set (equal number of crossbar inputs and outputs, equal frame size on inputs and outputs). This is a hard and beautiful problem, and it is worth thinking about it for a while.

6.2 Building a Schedule one Column at a Time

Let us try to construct a schedule one column at a time. In this exercise we will assume that the switch is *fully utilized*, i.e. the total number of connections per input as well as per output is equal to the frame size *m*. Under this assumption, all columns of the schedule must be full, which means that each column is "merely" a *permutation of all n colors*.

After one column of the schedule is built, constructing the rest of the schedule is equivalent to constructing a schedule for a switch with frame size *m-1* and for a connection-number array that results from the original array after subtracting 1 from each entry that corresponds to each of the colors and inputs that were included in the first column that was built. This is shown pictorially in figure 2, and it is equivalent to decomposing the original connection-number array into a sum of *m* "permutation" arrays (arrays like the one in the middle of figure 2).

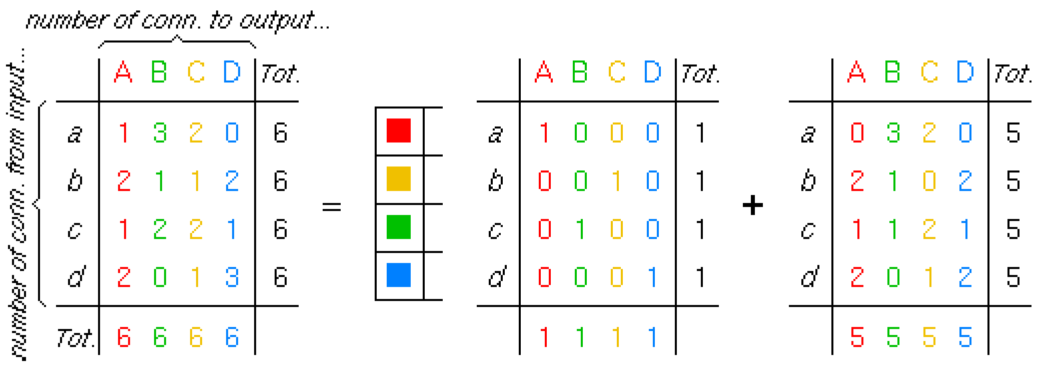


Figure 2

(a) Try to see whether this iterative algorithm will result in the construction of a valid schedule. A crucial point in to see whether it is always possible to find a full permutation of all colors in each and every step where a new column is built (without backtracking to rearrange previously made columns). Does it help to observe that the sums of the entries in the connection-number array, per row and per column, are all equal to *m-i* after the *i*-th step of the algorithm? Write your thoughts down, without spending an excessive amount of time to fully solve the problem.

(b) Think about, and discuss, various methods for constructing the permutations of colors that define the columns of the schedule during each step of the algorithm. Does it make a difference if you try to "consume" first the larger entries of the connection-number array, with the hope of ending up with few zero entries, or, conversely, if you try to consume first the smaller entries (the 1's), with the hope of ending up with many zero entries? Of course, a row or a column with a single non-zero entry immediately provides you with a uniquely-defined entry for the schedule, but does it also constrain you by making it harder to come up with the rest of the entries? Write your thoughts down, without spending an excessive amount of time to fully solve the problem.

6.3 Building a Schedule for a Non-Fully-Utilized Switch

Revisit exercise 6.1 in view of the algorithm developed in exercise 6.2. In exercise 6.1, connections were added to a *non*-fully utilized switch, and, as we saw, this necessitated, in some cases, the revision of the previously constructed schedule. The algorithm of exercise 6.2, on the other hand, assumed a *fully-utilized* switch.

Make some proposal(s) on how to adapt the algorithm of exercise 6.2 to non-fully utilized switches. Does the new algorithm look easier or harder? Apply your new algorithm to the scenaria of exercise 6.1(a). Where exactly in the algorithm is the point where a decision is made based on an "assumption" about future connections, such that, if the "assumption" turns out to be false, the schedule will have to be rearranged when the actual new connections arrive? Write your thoughts down, without spending an excessive amount of time to fully solve the problem.

Bibliographic References:

The questions posed in exercises 6.2 and 6.3 are beautiful, non-trivial problems. The following bibliographic references are related to them (you are not required to read them for answering this exercise set):

- T. Inukai: "An Efficient SS/TDMA Time Slot Assignment Algorithm", *IEEE Trans. Communications*, vol. 27, Oct. 1979, pp. 1449-1455.
- I. Gopal, D. Coppersmith, C. Wong: "Minimizing Packet Waiting Time in a Multibeam Satellite System", *IEEE Trans. Communications*, vol. 30, 1982, pp. 305-316.
- M. Bonuccelli, I. Gopal, C. Wong: "Incremental Time-Slot Assignment in SS/TDMA Satellite Systems", *IEEE Trans. Communications*, vol. 39, no. 7, July 1991, pp. 1147-1156.
