

**EE482B: Advanced Computer Organization
Interconnection Networks Architecture and Design
Stanford University**

Lecture #4

TOPOLOGY

(Packaging Torus, Slicing, Distributors, Concentrators and Non-Blocking Networks)

Lecture #4: Monday, April 16, 2001
Lecturer: Prof. William J. Dally
Scribe: Lesley Chang, Suresh Sankaralingam, Prashant Nedungadi
Reviewer: Kelly Shaw

Topics Covered in this lecture

1. Research Paper and Project Assignments
2. Torus Network Packaging
3. Slicing
4. Distributors and Concentrators
5. Introduction to Non-Blocking Networks

1.1 Research Paper Assignment

The Research Paper assignment is posted on the web. It involves the following major activities:

- * Pick a specific topic in any area of interconnection networks.
- * Write a critical review on 3 selected papers on the topic which was identified.
- * Describe the problem, describe the solution, and write a critique. See the Research Paper handout for more details.

The Research papers are due by May 16, 2001.

1.2 Project Assignment

The Project assignment is a much more involved task. It involves the following major activities:

- * Choose from the number of possible topics that are presented on the web or generate your own.
- * Take some small part of a problem in the chosen topic and try to solve it.

There are two checkpoints and a final report. See the Project handout for more details.

2.1 How to package Torus Networks?

Torus networks are easy to map to physical space. But, since they have one long end-around channel, from node $k-1$ to and from node 0 in each dimension they will have a length of k for the end-around channels. Such long channels may result in excessive latency or require slower signaling rates and causes non-uniform delay.

This problem can be solved by folding a torus network as shown in Figure 1. The first half of the nodes are packaged on top and the second half of the nodes are packaged on the bottom. By folding a k -ary n -cube torus network, in each plane, the distance between $k-2$ nodes is 2 and 2 nodes is 1. This is because at the ends of the ring, the length is 1 and between all other nodes the length is 2.

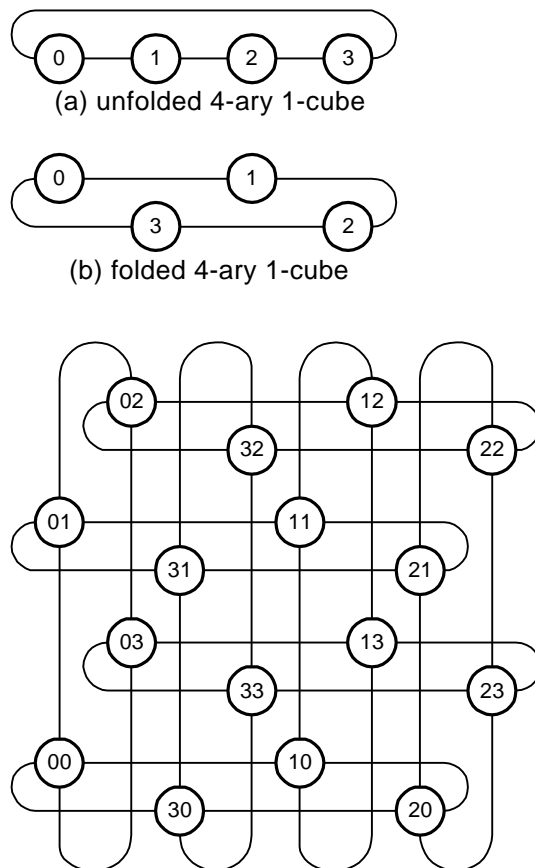


Figure 1. Examples of folded torus networks

By folding, the minimum distance between the nodes increase. But, the overall wire length is reduced between nodes. Also, the placement is easier. Depending on the topology, there is a trade-off between choosing a folded network or not.

2.2 How to package a two dimensional torus network in one dimension?

Quick answer: Compress it down and stretch it out.

When the number of logical dimensions exceeds the number of physical dimensions, several logical dimensions must be mapped into each physical dimension in order to realize it easily in the physical space. A 2 dimensional mesh (3-ary 2 mesh) is packaged into 1 dimensional mesh in the following way. The same concept can be applied to folded torus networks also. Figure 2 illustrates how the 2 dimensional logical channels are mapped to a single physical dimension.

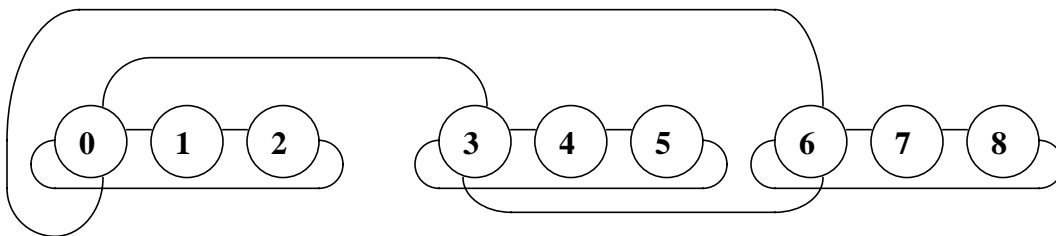


Figure 2. Mapping a 3-ary 2-cube to 1 dimension (Nesting)

This technique is called nesting.

Daily answered the question on folding for a 3 node ring:

Folding does not help for 3 nodes in a ring (this can be part of a network)

3. SLICING

Consider the following switch in Figure 3. There are 8 channels which are 64 bits wide and they are bidirectional. The total pinout for the switch would be $64 \times 2 \times 8 = 1024$. A switch with a large pinout may not be able to fit on a single chip. Therefore, we try to spread it across multiple chips. What are the options to reduce the pin count?

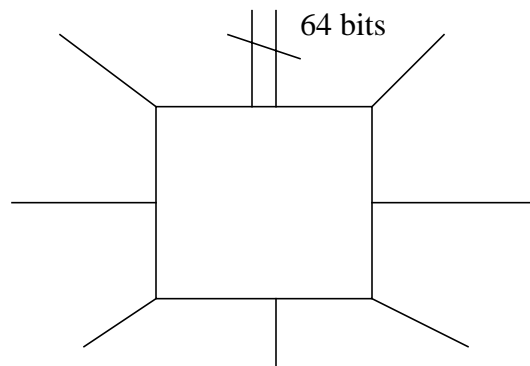


Figure 3. Eight-channel switch with 64 bits for each channel

There are 3 major ways to do slicing.

- 1) Bit Slicing
- 2) Channel Slicing
- 3) Dimensional Slicing

3.1 Bit Slicing

Bit slicing is the most straightforward method of dividing a network node across multiple chips. The example shown below has an 8-bit 2-D node being sliced into two 4-bit wide modules. Each of the 8 channels are divided into a pair of 4-bit channels. A number of control lines (ctl) are used to pass information between the two router bit slices.

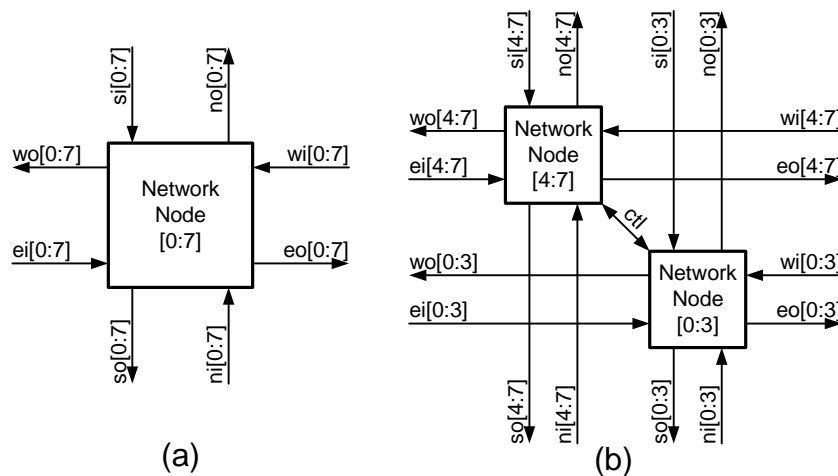


Figure 4. Example of bit-slicing for a 2-D torus network with $w=8$

Consider a 8-bit channel (0:7). It can be split into two 4-bit channels(0:3 and 4:7). In that case, what would be the structure of the payload and the header information in the packet? One option is that they can be duplicated. Each slice will carry a copy of the header as shown in the following diagram.

Payload	
vc	vc
dest	dest
dest	dest

Figure 5. Duplication of headers in packet for bit slicing

The disadvantage of this method is that for small packet sizes, the overhead in the packet is multiplied by the number of slices which results in increased serialization latency. Some of the alternatives for reducing the effect of the multiple headers are shown below.

One option of pipelining the header latency is to have header connected to master and payload connected to both master and slave. The header proceeds the payload on the master side, leaving room at the end of the current payload to store the next message, as in Fig 6(a). The slave buffers up the payload until it receives the control from master.

Another option is to send the header first as in Figure 6(c) and then follow it up with payload. Dally has seen bit slicing where they leave a gap between the header and the payload and then shape the payload in such a way that the next header can be processed in a pipelined manner as in Figure 6(d).

One more option is to split the header information as in Figure 6(b) and then process them in the bit slices and exchange the control information. When the 2 subnodes work in concert, both making the same decisions, there is no problem. But, when there are errors, they are no longer in lock-step. Also, in this case, the control decisions have to be distributed from the master to the slave and the chip crossing causes an increase in per hop latency and the serialization latency also increases.

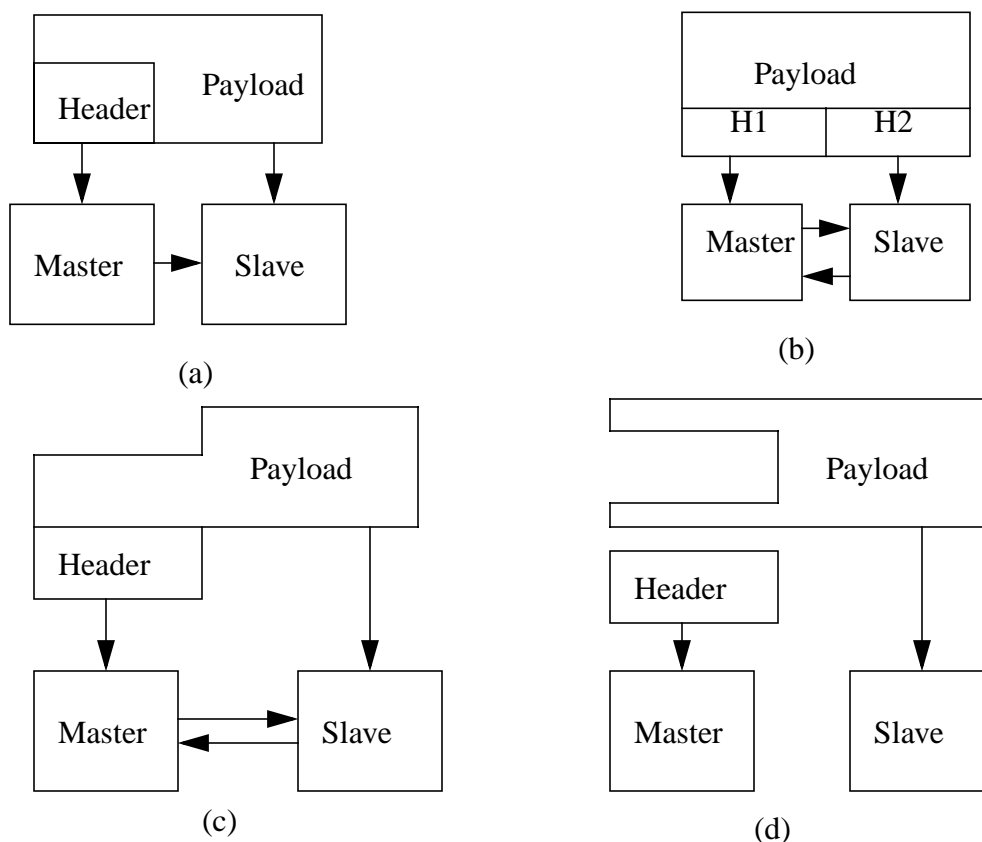


Figure 6. Header Pipelining and Split header for bit slicing

3.2 Dimension Slicing

In dimension slicing, the router is divided across its ports. For example, a 2-D router can be sliced into two 1-D routers: one that handles north-south traffic, and one that handles east-west traffic. The cost here is that you're adding an extra channel and thus, increasing the hop count.

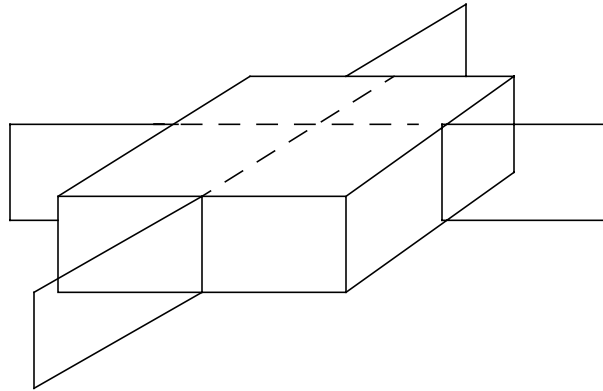


Figure 7. Dimensionally sliced network

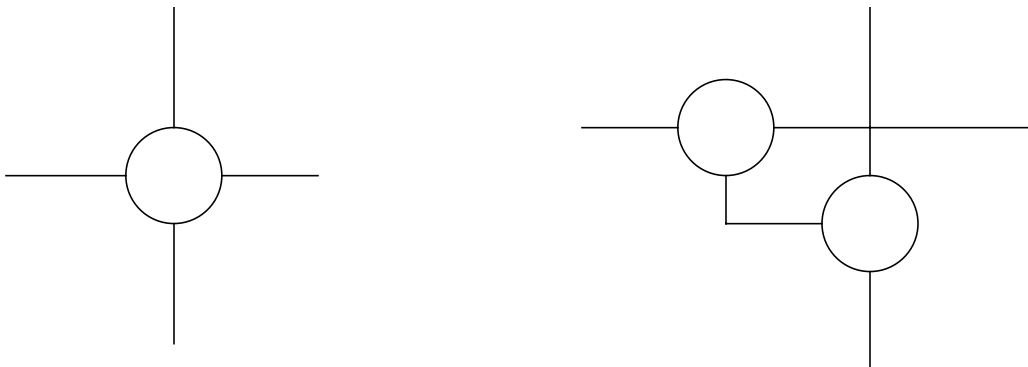


Figure 8. Example of dimension sliced network

Consider the figure shown below where a 2-D node is partitioned into two 1-D nodes. One carries the north-south traffic and the other carries the east-west traffic. An inter-slice channel handles traffic that crosses between these directions. By doing this, the hop count of the dimension sliced network increases. Consider that the traffic enters from the east and wants to go out in the north channel, it has to take the hop between the two dimension and that is why the hop count increases.

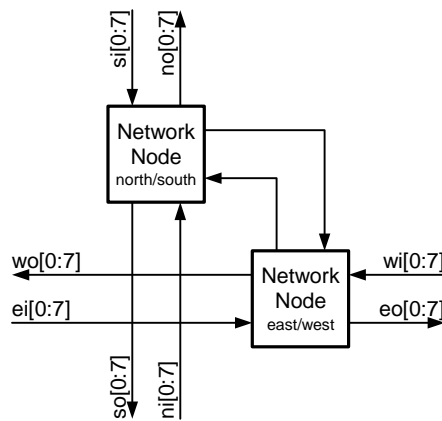


Figure 9. Example of dimension sliced network

3.3 Channel Slicing

In the channel sliced network, one wide channel can be broken down into multiple narrow channels which are completely separate networks. In channel slicing, there is no duplication of header information and no communication between the partitions. Channel slicing as shown in Figure 10 (a) and (b) effectively doubles the serialization latency while reducing the cross connect logic between two packets.

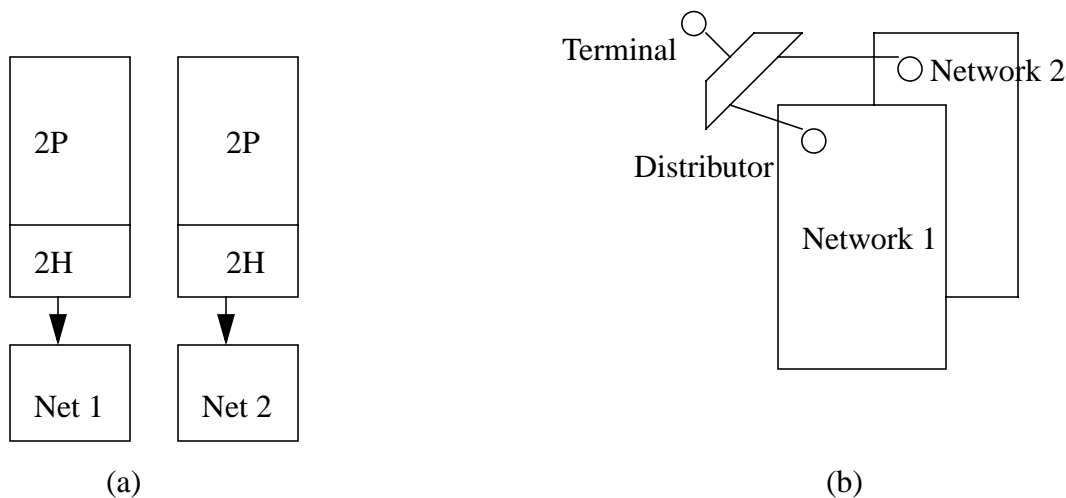


Figure 10. Examples of Channel Sliced Network

By doing channel slicing, it is clear that you end up with 2 halfwide network nodes. Channel slicing has fault tolerance, but there could also be load balance issues if the distributor doesn't distribute the traffic between the two half-wide networks properly. Another problem with channel slicing is that since bandwidth is halved, the serialization latency is doubled.

3.4 Summary of Slicing:

Any time you slice a node up, you pay for it one way or another.

1. Bit Slicing

- distributing control information, which increases t_r .
- best if packets are very long and you don't care about latency

2. Dimension Slicing

- add a hop, which increases H .
- best with small packets that are dimension ordered

3. Channel Slicing

- decreases bandwidth, which increases T_s .
- has fault tolerance
- best if don't care about T_s .

4. DISTRIBUTORS AND CONCENTRATORS

4.1 Distributor:

A distributor takes one high bandwidth channel and distributes its packets over several lower bandwidth channels. Distributors provide scalability when the nodes of the network move towards handling higher bandwidths than what the link currently supports. In the example given below the 10 Gbps node is distributed into two 5-Gbps links

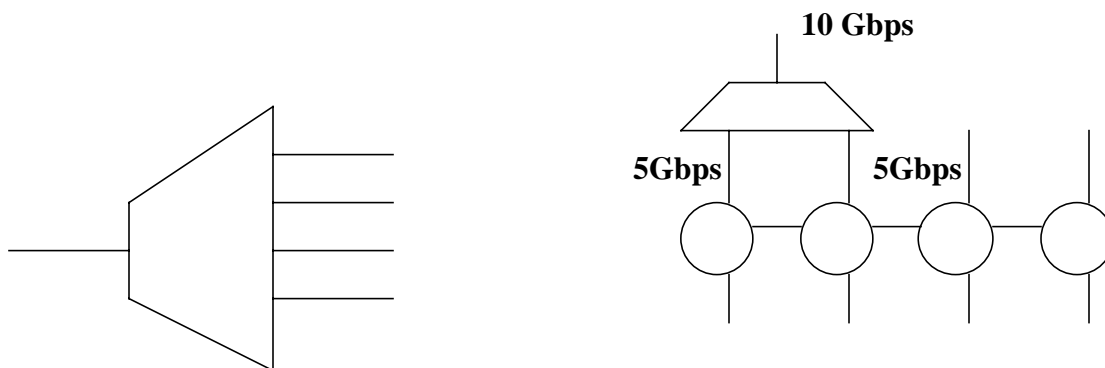


Figure 11. Examples of Distributors

Why use distributors?

- change in bandwidth
- fault tolerance
- machine upgrades (A single node in the future may have a higher bandwidth than the links in the network, so you will need a distributor and ≥ 2 links to support the higher bandwidth)

4.2 Concentrator:

A concentrator does the reverse of the distributor. It combines traffic from multiple terminal nodes to form one wide-bandwidth channel. This is good for links which have bursty behavior, like in processor-memory interconnects, where average demand for bandwidth is small, but immediate demand for bandwidth is high. Also, we start with a higher dimension network (8x8x8) and then get down to a lower dimension network (4x4x4) using 8:1 concentrators.

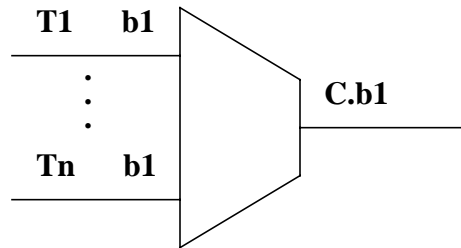


Figure 12. Concentrator

Example problem:

No. of Nodes = $N = 512$

Average Bandwidth = $b_{avg} = 1 \text{ Gb/s}$

Peak Bandwidth = $b_{pk} = 10 \text{ Gb/s}$

* A good metric for cost is total pin bandwidth.

Pin Bandwidth = $B_{pin} = N * \text{channels/node} * \text{peak bandwidth for one node}$

Pin Bandwidth = $512 \times 6 \text{ channels/node} \times 10 \text{ Gbps} = 30 \text{ Tb/s}$

With concentrators:

$N = 64$

Peak Bandwidth = 10 Gbps

$B_{pin} = 64 \times 6 \times 10 \text{ Gbps} = 3.84 \text{ Tb/s}$

The calculations are only for unidirectional channels ($2 * n = 6$ channels, in a 3-D network). In the above case, we are not counting the injection/ejection channels since the nodes require those irrespective of what concentration we have.

When we do concentration, the one factor we are considering is that the network operates at a maximum peak bandwidth which is equal to the peak bandwidth of one terminal, considering that the other terminals would not transmit during that time. We can probabilistically prove that the probability of the conflict between two or more terminals is very small.

In Figure 13 (a), we find that the probability of only one terminal transmitting at 10Gbps is 0.1 and the probability of zero traffic is 0.9. When 2 terminals transmit, the probability of no traffic drops to $0.9 \times 0.9 = 0.81$ as shown in Figure 13(b). The probability that only one node transmits becomes 0.18 and the probability that 2 nodes transmit (conflict case) is 0.01. This can be extended to the case where three nodes transmit as shown in Figure 13(c). As you increase the number of concentration, the probability of collision increases.

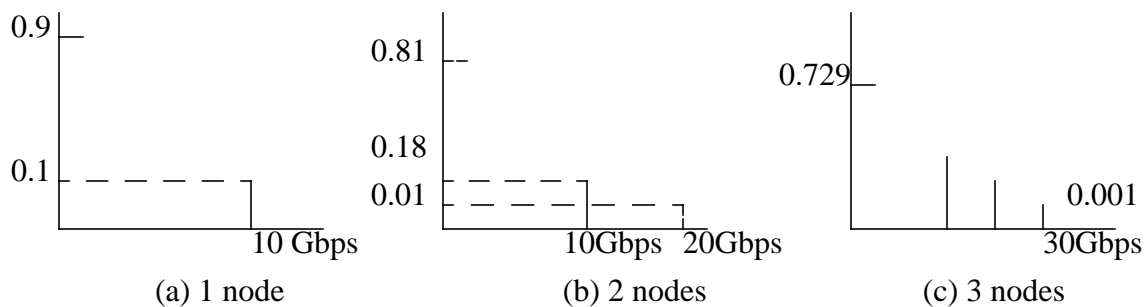


Figure 13. Conflict probabilities in concentrators

The probability that one of the node transmits at peak bandwidth when the other nodes transmit at peak bandwidth is very small. There is a Cost-Performance trade-off when we try to switch to a concentrator/distributor based network. You can either fix the cost, and look at performance or fix the performance, and look at the cost. Additionally, the concentrator and the network node are usually placed on the same package.

5. NON-BLOCKING NETWORKS

Blocking or Non-Blocking behavior of a network is essentially a “circuit switching concept”. If the network is blocking, then rearrangement is needed. The rearrangement is done in response to protection switching. If a network line is lost because somebody ripped it out of the ground, you have to make sure that the connection is re-established within a small amount of time. Since people don't have enough time to compute new arrangement of routes, non-blocking is used. Non-Blocking Networks are categorized into 2 categories

- 1) **Strictly Non-Blocking:** if you have an unused input and an unused output, you can always find a connection between them without altering the path taken by any other traffic.
- 2) **Rearrangeable (Rearrangeably Non-Blocking):** it can connect any unused input to any unused output, but it may need to reroute some traffic to make this connection.

Rearrangeable networks can also have hitless switching.

Hitless switching: cannot tell that you switched in a frame boundary; no bits were lost on the switchover.

Looking from the ports perspective, rearrangeable with hitless switching is indistinguishable from strictly non-blocking. However, rearrangeable routes need to be computed off-line and in some situations, there may not be enough time to compute the new routes.

5.1 Crossbar Networks

The crossbar is a strictly non-blocking network with stiff flow control. An $n \times m$ crossbar or crosspoint switch directly connects n inputs to m outputs with no intermediate stages. In effect, such a switch consists of $m \times n$ multiplexers, one for each output. Many crossbar networks are square in that $m=n$. Others are rectangular with $m > n$ or $m < n$.

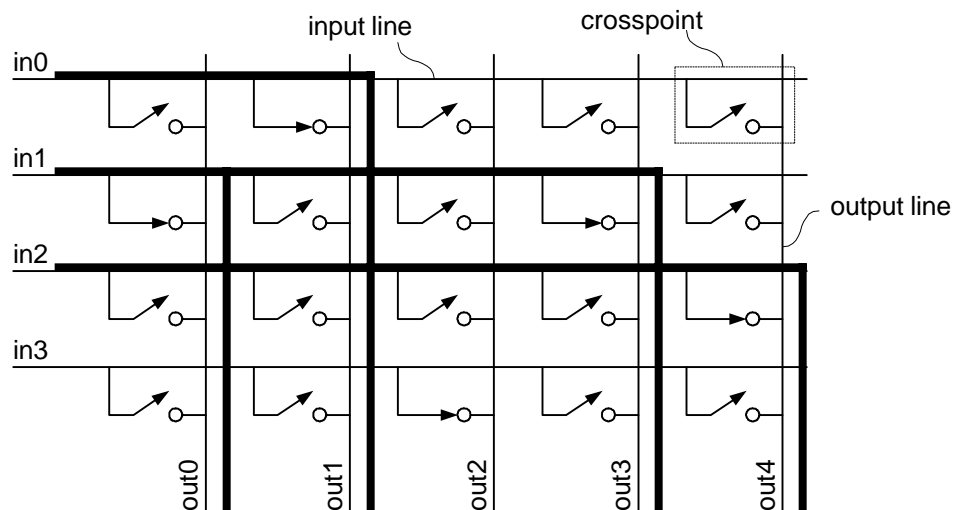


Figure 14. Relay based crossbar switch

In Figure 14, we have a relay based crossbar. Here, at each cross connect, you have a switch. When the relays are closed, then they form a path between the input and output ports connected to that switch. By closing more than one switch, the same input port can transmit to multiple output ports. This feature is used in multicasting.

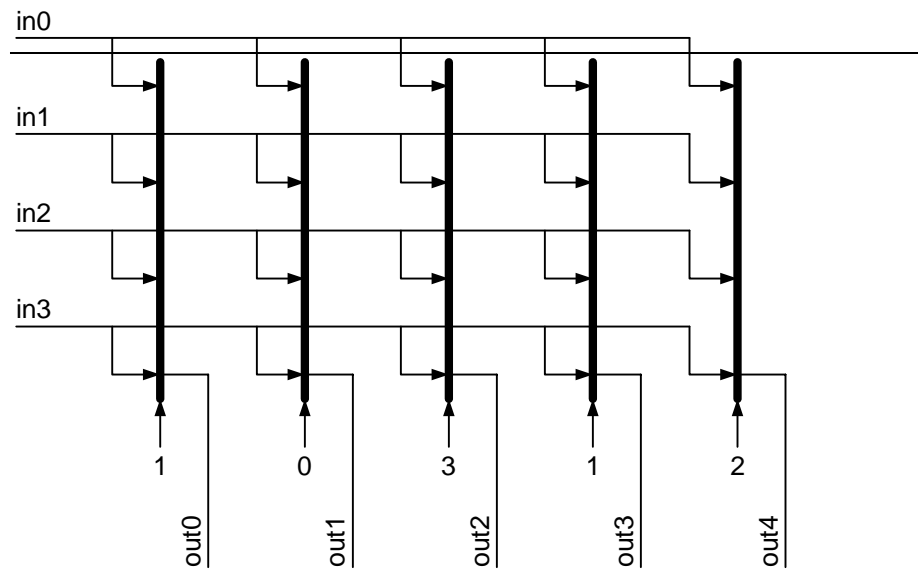


Figure 15. Digital Mux based Crossbar

In the mux based switch as shown in Figure 15, the relay switches are replaced by digital muxes. Each output port selects the input it wants. Thus the switching is accomplished in case of a mux based switch. Both kinds of crossbars allow unicast and multicast traffic.

How to build a larger crossbar out of smaller ones?

1. Bit slicing
2. Grid of crossbars

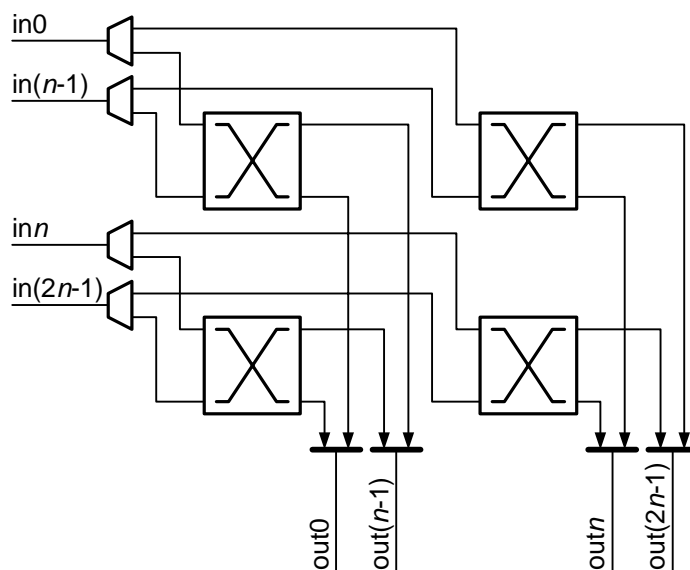


Figure 16. A $2n \times 2n$ crossbar constructed from 4 $n \times n$ crossbars

A $kn \times kn$ crossbar can be constructed from $k^2 n \times n$ crossbars as shown in the Figure 16. The amount of cross bar switches needed are $kn \times kn$ which grows as n^2 . The fan-in and fan-out logic will increase by $n \log(n)$. Effectively, the number of crossbar switches dominates the cost. The crossbar is pin limited by the above problem. But, if the entire crossbar can fit on the chip, then you are all set.

Scheduling crossbars is a very interesting topic, which we will talk about later this quarter. Although the idea of connecting in to out is easy, this is not a trivial topic and we will find out why in a future lecture.

5.2 Clos Networks

In a Clos network, each middle-stage switch has one input link from every input switch and one output link to every output switch. Thus, the r input switches are $n \times m$ crossbars to connect n input ports to m middle switches, the m middle switches are $r \times r$ crossbars to connect r input switches to r output switches, and the r output switches are $m \times n$ crossbars to connect m middle switches to n output ports. This is shown clearly in Figure 15.

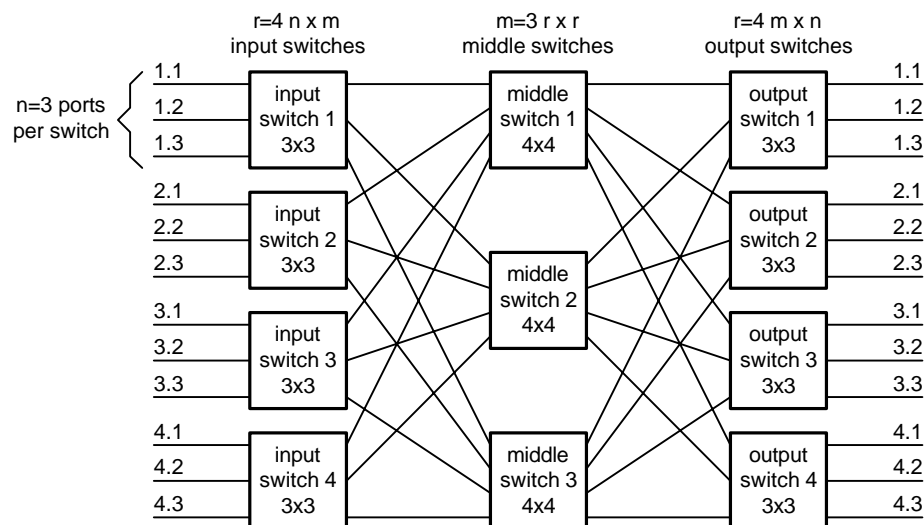


Figure 17. A $(m=3, n=3, r=4)$ symmetric Clos network

A symmetric Clos network is characterized by (m, n, r) where:

m: the number of middle-stage switches

n: the number of input (output) ports on each input (output) switch

r: the number of input and output switches.

This network has m -fold path diversity because there is 1 path through each middle switch. It is strictly non-blocking if $m \geq 2n - 1$. It is because when all others are transmitting, then there are $2 \cdot (n-1)$ switches busy. In order to route the call, we need at least one free middle stage. So, we need at least one more than $2 \cdot (n-1)$ in order for the network to be non-blocking.

The model being

Stage 1 : 1111111...111000000000...000

<---n-1--->

Stage 2 : 0000000...0000111111...111

^<---n-1--->

|

available

We therefore need $2(n-1) + 1$ middle stage switches. This is strictly non-blocking because there is no way for an adversary to concentrate traffic because traffic can go over any middle switch (and there will always be a free middle switch).

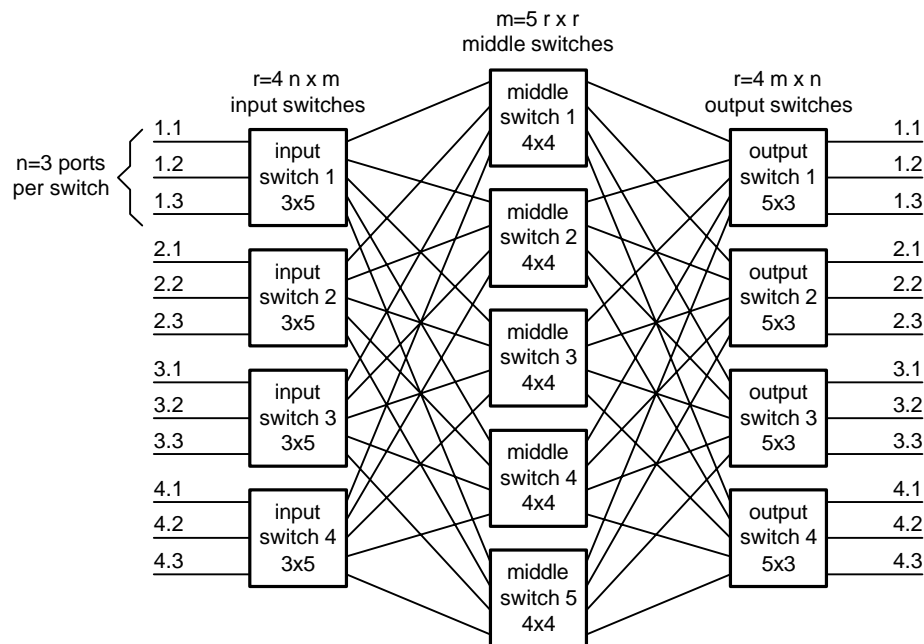


Figure 18. A (5,3,4) Clos network

As we derived the condition for a Clos network to be strictly non-blocking, we know that the network shown in Figure 17 is not a strictly non-blocking network since $m = 3$ is not greater than $2n - 1$ (5). The network shown in Figure 18 is strictly non-blocking for unicast traffic since it satisfies the condition ($m > 2n - 1$). So, at any point of time, you can always find a free middle stage to switch the incoming traffic.

For a complete routing example, consider the permutation $\{5, 7, 11, 6, 12, 1, 8, 10, 3, 2, 9, 4\}$. That is, input (1.1) routes to output 5 (2.2), input 2 (1.2) to output 7 (3.1) and so on. Since we need only consider switches and not ports, we can simplify this to $\{(2, 3, 4), (2, 4, 1), (3, 4, 1), (1, 3, 2)\}$. Input switch has connections to output switches 2, 3 and 4 as you can see from Table 1. The first 3 columns show the input switch from which the call is coming (In), the output switch to which the call is going (Out), the middle switch allocated to the call (Middle). The remaining eight columns give bit vectors showing which middle switches are free from each input and output switch.

In	Out	Middle	Input Free				Output Free			
			1	2	3	4	1	2	3	4
3	1	1	11111	11111	01111	11111	01111	11111	11111	11111
2	1	2	11111	10111	01111	11111	00111	11111	11111	11111
3	3	2	11111	10111	00111	11111	00111	11111	10111	11111
3	4	3	11111	10111	00011	11111	00111	11111	10111	11011
1	4	1	01111	10111	00011	11111	00111	11111	10111	01011
4	2	1	01111	10111	00011	01111	00111	01111	10111	01011
4	1	3	01111	10111	00011	01011	00011	01111	10111	01011
2	4	4	01111	10101	00011	01011	00011	01111	10111	01001
1	2	2	00111	10101	00011	01011	00011	00111	10111	01001
4	3	4	00111	10101	00011	01001	00011	00111	10101	01001
1	3	3	00011	10101	00011	01001	00011	00111	10001	01001
2	2	3	00011	10001	00011	01001	00011	00011	10001	01001

Table 1. Example of routing call on a strictly non-blocking Clos network

Suppose you want to switch a call from 9 (3.3) to 3 (1.3). The input switch is 3, the output switch is 1, and since there are no paths blocked for this first call, it is assigned middle switch 1. After doing the allocation, the bit vectors are cleared corresponding to the connection (0 means busy in the bit vector). Basically, you look to find a middle switch that isn't used by either the input switch or the output switch. Similarly, all the calls are routed.

The procedure for routing a strictly non-blocking Clos network is very simple as we discussed above. The penalty comes in the form of cost. A strictly non-blocking Clos network is nearly twice as expensive, requiring $(2n-1) / n$ times as many middle switches, than a corresponding rearrangeable network.

Next time we will talk about rearrangeable non-blocking Clos networks and how to set up calls. We will also finish up non-blocking networks.