Switch Design

[§10.3.2] In earlier lectures, we have seen that switches in an interconnection network connect inputs to outputs, usually with some kind buffering.

Here is a basic diagram of a switch.

![Switch Diagram]

Usually, the number of input ports is the same as the number of output ports. This is called the degree of the switch.

Each input port includes a receiver. A received message goes into an input buffer.

Each output port includes a transmitter to drive the link. There may also be an output buffer.

The control logic implements the routing algorithm. It must include some way to arbitrate among competing requests for the same output port.

[§10.7] A major constraint on switch size is the number of pins. How would this be determined?
This tends to favor fast serial links. They use the least pins and eliminate problems with skew across bit lines in a channel.

With parallel links, one of the wires is essentially a clock for the data on the others.

Logically, the crossbar in an \( n \times n \) switch is just an \( n \)-way multiplexer associated with each dimension (at right).

In VLSI, it is typically realized as a bus with a set of \( n \) tristate drivers (below).

An increasingly common technique is to use memory as a crossbar, by writing for each input port and reading for each output port.

While crossbars are expensive, it makes sense to build a switch as a crossbar.

- An interconnection network can be built out of relatively low-degree switches.
The limiting factors in switch design are the length of wires and number of pins, so reducing the internal bandwidth doesn’t save much.

**Channel buffers**

In VLSI switches, buffers are on chip and therefore compete for space with the datapath and control section.

There are four options.

- No buffering (beyond input and output latches).
- Buffers at the inputs.
- Buffers at the outputs.
- A centralized buffer pool.

The choice of buffering will affect the utilization of other components in the network.

**Input buffering**

Independent FIFOs can be associated with each input port.

Each buffer needs to be able to accept a phit each cycle and deliver one phit to the output. (A phit is the amount of data transferred across a link in a cycle.)
Each input has independent routing logic.

- Trivial for source-based routing.
- Requires an arithmetic unit for algorithmic routing.
- Requires a routing table per input for table-driven routing

With cut-through routing, the logic makes a new decision once per packet. The routing logic is a FSM that spools all the flits of a packet to the same output channel before making a new routing decision.

Input buffering is susceptible to head-of-line blocking. What is this?

How bad is this problem? Consider a switch with two input ports, and assume that packets are directed to random outputs.

The first packet makes it, and the second has a 50/50 chance of choosing the unused output.

Therefore, the expected number of packets moving through the switch per cycle is \( \frac{n}{2} \), and the expected utilization of each output is

Generalizing this, if \( E(n, k) \) is the expected number of output ports covered by \( k \) random inputs to an \( n \)-port switch, then

\[
E(n, k+1) = E(n, k) + \frac{n - E(n, k)}{n}
\]

According to this formula, the expected output channel utilization of a single cycle in a fully loaded switch quickly drops to about 65%. Queuing analysis shows that the expected utilization in steady state is 59%.

However, these analyses tend to understate the problem. There may be bursts of traffic for one output, followed by bursts to another. Why does this make the situation worse?

In a wormhole-routed network, the entire worm will be stuck in place, effectively consuming link bandwidth.
To solve this problem, we can look at other buffering approaches.

**Output buffering**

One approach to preventing head-of-line blocking is to create a separate input buffer for each output port.

Then, regardless of input traffic, outputs can be driven at 100%. However, this is costly.

The extra buffers in this design can be viewed as associated with the outputs. Each output port has enough internal bandwidth to receive a packet from every input port in one cycle.

**Shared pool**

With a shared pool, each input port deposits data into a central memory, and each output buffer reads from it.

There’s no head-of-line blocking. Why?
How can this design match the bandwidth of the input and output ports? One way is to make the datapath to the pool twice as fast as the links.

**Output scheduling**

OK, we’ve now designed a switch that can get packets to output ports as fast as they arrive. Suppose more than one packet hits an output at a time; how do we decide which one goes first?

Each input buffer has a request line to each output port, and a grant line from each port.

In a tristate-logic implementation, the output-port \( j \)'s scheduler would enable input buffer \( i \) by asserting control-enable \( e_{ij} \).

One good way of choosing an output is oldest-first. This can be done with a FIFO maintaining a queue of requests for each output.

In the general switch, we need to be able to connect any input to any output. However, the routing algorithm may simplify this.

What simplification arises in dimension-order routing in a \( d \)-cube?
Stacked-dimension switches

The simplest switches are those that connect two inputs to two outputs. It’s possible to build any switch using $2 \times 2$ switches as building blocks.

With dimension-order routing, we need to make a decision about whether the current input gets routed to the output in the same dimension, or gets routed “up the stack.”

Routing and switch-design summary

Routing algorithms restrict the set of routes within the topology

- simple mechanism selects turn at each hop
- arithmetic, selection, lookup

Deadlock-free if channel dependence graph is acyclic

- limit turns to eliminate dependences
- add separate channel resources to break dependences
- combination of topology, algorithm, and switch design

Deterministic vs. adaptive routing

Switch-design issues

- input/output/pooled buffering, routing logic, selection logic

Real networks are a “package” of design choices.