Space-division fabrics

Keshav Chapter 8
Textbook references

• Keshav: Ch. 8.4
• Varghese: Ch. 13
  • switch cost: 13.9.1 of Varghese
  • other space division switch fabrics: Varghese pp. 443-4
Original References


Tutorial/survey References


Online texts (see the TELE9751 reading list):
Outline

Single-stage: Crossbar switches
  Handling crossbar output port contention
    Knockout
    Arbitration

Staged switches
  Networks of basic elements
  Clos
  Banyan
    Structure
    Blocking & Motivation for sorting

Batcher sorting networks
Crossbar switches

At the intersection of each input and each output, there is a “cross point”, which can be selectively enabled, allowing communication from input to output.

e.g. Intel 470 switches, Cisco Catalyst 6500, 12000 series routers

Feasible to implement in VLSI (e.g. PMC-Sierra PM9312), limited by high-speed I/O to chip - pincount

[Sketch from znet.net/~cdk14568/mpet/contacts/fig23-5.gif. Photos from Lucent]
Advantages of crossbar switches

✓ Simple structure
✓ Suitable for circuits or packets
✓ Low latency – minimal number of connecting points between arbitrary input and output.
✓ No internal blocking (may have output port blocking)
✓ Multicast is easy with electrical crossbars
  × Difficult with MEMS optical crossbars, since mirror will (hopefully) reflect all signal power to one intended output.
  × Easy to reach output, but scheduling multicast s.t. all outputs are simultaneously available (other inputs aren’t transmitting to some) can be tricky.
Disadvantages of crossbar switches

- **Scalability:**
  - # of crosspoints \((N_x)\) grows with \((P=\text{number of ports})^2\)
  - Difficult to incrementally expand
  - Output buffer speed \(\propto P\), not line speed (\(\Rightarrow\) knockout process).

- **Fanout:** Number of crosspoints on each line increases linearly with number of ports, increasing capacitive loading, slowing transmission (just as bad as time-division bus, but inferior to space-division fabrics such as Banyan)

- No **fault tolerance:** each crosspoint is needed for one connection or another.
Handling crossbar output port contention

Crossbar switches (like all switches) can suffer output port blocking.

**Responses:**

- **Internal buffering** at crosspoints (next slide)
- **Buffers at outputs.** Issue: How to reduce speed of buffer, or reduce number of low (input-port) speed buffers? → **Knockout tournaments**
- **Block at input.** Issue: How to be fair to inputs s.t. none is blocked forever? → **arbitration systems.**
Internal crossbar buffering

Internal buffering, e.g. to handle output contention

Fig. 8.14 from Keshav
Buffering at crossbar outputs

Multiple inputs might

• **Time-division access to one output port buffer.**
  • Small, simple, but need fast buffer.
    • Speedup problem: as number of inputs increases, speed of buffer at output must increase (like shared medium switch)

• **Lead to multiple distinct buffers on each output port.**
  • Low speed, but no sharing => more buffer space needed.

How about something in-between: some $L<P$ buffers, and mechanism (Knockout tournament) to spread inputs evenly over buffers?
Knockout switch

Can we exploit the directional nature of traffic (few inputs will be directed to any one output at a time) to reduce the output port buffer capacity (size+speed)?

So use a “knockout tournament” to decide which inputs get buffered:

• Competitors (inputs) play and those who win progress to the next round. At the end, one competitor is selected (transferred to buffer); others have lost one match.

• Losers then compete again (clean slate) to determine next competitor to be selected.

• Repeat to capacity of output port.

For details, see Peterson and Davie, 1st edition, pp. 193-6; Keshav p. 199
Knockout structure

$P:L$ knockout concentrator selects up to $L$ pkts from $P$ inputs
(here $P=8$, $L=4$)

Scenarios

$P_x =$ packet from L/R input

$P_L$ $P_R$ $P_L$ - $P_R$ -

$P_L$ - $P_R$ -

$P_L$ or $P_R$ (to be fair to RHS ports)

Drawings based on L. Peterson

Copyright © 15/03/11 Tim Moors
Crossbar arbitration†

**Problem:** When multiple inputs have info for one output, crossbar controller needs to determine when each should be switched through.

**Aims:** Schedule aims for
- **Performance:** Maximise throughput by minimising transfer rounds; minimise delay
- **Fairness:** each input should receive equal access.
- **Cost:** Shouldn’t require much state info or many iterations.

**Generic method:** May achieve this through multiple iterations (per round) of negotiation between inputs & outputs

† Often (e.g. by Varghese) called “scheduling”, but called arbitration by others (e.g. Cisco) & “arbitration” avoids confusion with later lecture on “packet scheduling”: Scheduling affects the timing of packets; here timing for crossing the fabric; later packet scheduling timing when sent on a line.
Arbitration basis

• Each input maintains a separate “Virtual Output Queue” of cells destined for each output. (Fixes head of line blocking)

• Each of $P$ inputs maintains $P$ VOQs => $P^2$ bits of activity info control arbitration. Feasible to optimise with tens of ports (e.g. $P=32$ => 1024 bits)

• Need to
  • convey activity info between inputs and outputs
    • Inputs send requests to outputs
    • Each output offers to serve one request (must choose fairly)
    • Each input accepts one offer
  • coordinate inputs and outputs, e.g. multiple iterations: Reiterate to use outputs whose offers weren’t accepted.
    • Trade-off between performance and number of iterations
Parallel Iterative Matching (PIM)

- Offers are made to random inputs
- Used in DEC AN-2 30 port switch
PIM example

Figure from Varghese
Correction: Requests from A to 1 should be gray; they are similar to other requests. “a=1” on input c is an artifact of iSLIP example.
PIM example
(repeated for non-animated display)

Figure from Varghese
Correction: Requests from A to 1 should be gray; they are similar to other requests. “a=1” on input c is an artifact of iSLIP example.
PIM vs iSLIP

• “PIM … requires a logarithmic number of iterations to attain maximal matches”
  vs “iSLIP … gets close to maximal matches after just one or two iterations.”
  (though “maximal” ≠ “close to maximal” & PIM example uses 1 iteration and iSLIP example uses 2 iterations)

• in terms of fairness mechanism
  PIM : Ethernet as iSLIP : token ring
    randomness taking turns
iSLIP

• Each port maintains a pointer to next port desired to use.
• Choose lowest port $\geq$ pointer; update pointer
  • “$\geq$” in circular sense: $P+1 = 1^{st}$ port
• Pointers initially all point to 1$^{st}$ port, but desynchronise as randomly directed traffic passes through.
• Used in Cisco GSR router

iSLIP round 1

Pointers only get updated after iteration 1 => B still points to 1 & 2 still points to A

B overheads A accepting 1 => know that 1 is busy => no need to request

Figure from Varghese
iSLIP round 2

Figure from Varghese
Round2 iteration 1 should show C with packet for output 3
iSLIP round 3

Round 3, Iteration 1

Round 3, Iteration 2

Figure from Varghese
Outline
Staged switches

Multistage switches are composed of “networks”† of smaller switches (e.g. crossbars or shared-media), often 2×2

Potential benefits:

√ **Fewer crosspoints** than in a crossbar switch

√ **Diversity of paths**

**How?**: Selection of switch in first and last stage is determined by which input & output are being connected

=> use *three* or more stages to get benefits

**Why?**: Intermediate stages can offer choice of switch =>

√ **Lower blocking** probability

√ **Increased reliability**: can still connect input and output even if a component switch has failed

† Here we are interested in networks *within* the switch. Of course switches can also be interconnected to form external networks (the more common use of the word).
Clos switches

• Each switch (“array”) of a stage has one output feeding into each switch of the next stage (# inputs may ≠ # outputs for internal switches)

• Simplest Clos switches have 3 stages, but more are possible by recursively replacing middle stages by 3-stage Clos structures.

• Advantages:
  √ can switch circuits
  √ path diversity
  √ fewer crosspoints for large $P$

Ignoring the ellipsis, this example has $P=8, n=2, k=3$

$$N_x = (n \times k) \left(\frac{P}{n}\right)^2 + \left(\frac{P}{n} \times \left(\frac{P}{n}\right)\right)k$$

$$= 2Pk + kP^2/n^2$$

What are the optimal values for $n$ and $k$?
\( k = \text{Number of arrays in middle stage} \)

To be non-blocking, must be able to create a connection in the presence of existing connections on all other ports of input and output arrays.

Each array has only one connection to switches in adjacent stages.

How many middle-stage arrays may be in use?

- \( n-1 \) may be tied up with other inputs from the same input array. (\( n=3 \) here)
- \( n-1 \) may be tied up with other outputs to the same output array.

Worst case: may have different mid-stage arrays tied up by inputs and outputs.

\[ \Rightarrow \text{need at least } (1+1)(n-1)+1 \text{ middle-stage arrays, i.e. } k \geq 2n-1 \]

\[ \Rightarrow N_x = 2P(2n-1) + (2n-1)P^2/n^2 \]
$N \rightarrow$ Number of arrays in outer stages

Differentiate $N_x$ wrt $n$

\[
0 = 4P + \frac{n^2 2P^2 - (2n - 1)P^2 2n}{n^4}
\]

\[
0 = 4PN^4 + 2P^2n^2 - 4P^2n^2 + 2P^2n
\]

\[
0 = 2n^3 - Pn + P
\]

For $n$ large, neglect $+P \equiv +Pn^0$ factor

\[
n \approx \sqrt[4]{P} / 2
\]

\[
N_x(\text{min}) = 4P\left(\sqrt{2P} - 1\right)
\]

<table>
<thead>
<tr>
<th>Ports</th>
<th>$N_x(\text{3-stage Clos})$</th>
<th>$N_x(\text{Cross})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>96</td>
<td>64</td>
</tr>
<tr>
<td>128</td>
<td>7,680</td>
<td>16,384</td>
</tr>
<tr>
<td>2048</td>
<td>516,096</td>
<td>4,194,304</td>
</tr>
</tbody>
</table>

† Bonus mark on offer if you can justify why $n$ (not $P$) should be large.

Is it because large $n$ leads to large $k$ for non-blocking, and with large $k$, adding another array in the middle to provide fault tolerance with little overhead?
(Rearrangeably) non-blocking Clos

- $2n-1 \leq k$ ensures Clos switch won’t block
- $n \leq k < 2n-1$ ensures Clos switch is **rearrangeably** non-blocking, i.e. Can connect input to output, but might have to rearrange existing connections

e.g. $n=k=2$
- to connect each input to corresponding output requires pattern on left
- straight connections may end; but cross connections block connection between bottom input to top output
- need to rearrange
Clos switches

Used in some commercial products:

• Juniper T-series routers
• NEC ATOM ATM switch
• Myrinet switch

Photo from http://www.cs.utk.edu/~dongarra/lyon2000/Talk02-Chuck-Seitz.ppt
Outline
Pipelining

Pipeline: “A sequence of functional units ("stages") which performs a task in several steps, like an assembly line in a factory. Each functional unit takes inputs and produces outputs which are stored in its output buffer. One stage's output buffer is the next stage's input buffer. This arrangement allows all the stages to work in parallel thus giving greater throughput than if each input had to pass through the whole pipeline before the next input could enter.

The costs are greater latency and complexity due to the need to synchronise the stages in some way so that different inputs do not interfere. The pipeline will only work at full efficiency if it can be filled and emptied at the same rate that it can process.” [http://foldoc.org/]

- Multistage switches can operate as pipelines, with each stage operating (at any instant) on a different set of packets.
- Cells (rather than variable-length packets) enable all switches in a stage to keep operating for the same period.
Multistage networks: 1. Physically separate stages

Multistage switches work well as pipelines if stages take same time
=> Prefer that all units of information have the same length (cells)
=> ATM, or port processors may do segmentation and reassembly

Figure from H. Peyravi
Multistage networks: 2. Recirculation

Multiple logically separate stages (emulating physically separate stages) can be created by recirculating, over time, data through one physical stage.

Inputs of switches A-D: External, Recirculated

Outputs of switches A-D: External, Recirculated
Pipelined multistage switches

- With multiple stages, each transfer (from input to output) passes through successive stages.
- At any instant:
  - circuit engages the complete path through the fabric
  - packets need only engage one stage at a time
  => can pipeline packets through: at any instant, each stage processes a separate wave of packets
- Packet switching, stages could act autonomously, each responding to different packet header bits.
- If we identify output ports with binary addresses, successive stages could handle progressively less significant bits of the address.
  
  (Note: “address” here is for internal use within switch. Packet classifier will take address from packet (e.g. IP) header to determine output port, and hence internal address)
Banyan switches

Self-routing using binary representation of output port (extra header for use inside switch)
Direction for each stage specified by bit corresponding to that stage (MSb 1st)
=> can’t multicast
$P$-port switch has $\log_2 P$ stages each with $P/2 \times 2$ switches
=> $N_x = 2P\log_2 P$

1 => switch to right output port of switch in this stage
0 => switch to left output port

“There is a very large, famous Banyan tree that occupies more than 3 acres of land in a park in Lahaina, on the island of Maui in the Hawaiian islands.” – Seifert p. 208
Switch complexity compared

Number of crosspoints for a switch with $P$ ports:

- Crossbar: $N_x = P^2$
- Clos: $N_x = 4P(\sqrt{2P} - 1)$
- Banyan: $N_x = 2P\log_2P$

Banyan switches

✓ require the fewest crosspoints (of those covered)
× but suffer from the potential for blocking
× typically limited to packet switching
Project: Software Banyan

- Rudimentary (e.g. fixed size), but demonstrates the idea.
- Full source code in 3FabricBANYAN.c

```c
struct banyanElement {
    int elementNumber;
    // flags indicate if outputs busy top=0,bot=1
    int topFlag;
    int bottomFlag;
    // pointers to next element
    banyanElement *zero;
    banyanElement *one;
};

banyanElement *elements[12];
```

```c
void fabricElements() {
    elements[7]->zero = elements[10];
    elements[7]->one = elements[11];

    /* Will print out graphical representation */
    printf("\n");
    printf("000 ---| 0 |---| 4 |---| 8 |--- 000\n");
    printf("001 ---|___| \|___|\  |___|--- 001\n");
    printf("      \ \_/\ \  \\
");
    printf("010 ---| 1 |---| 5 |/  | 9 |--- 010\n");
    printf("011 ---|___|---|___|\  |___|--- 011\n");
    printf("      \ /  \\
");
    printf("100 ---| 2 | \| 6 |---| 10|--- 100\n");
    printf("101 ---|___|---|___|\  |___|--- 101\n");
    printf("      \ /  \\
");
    printf("110 ---| 3 | | 7 |/  | 11|--- 110\n");
    printf("111 ---|___|---|___|---|___|--- 111\n");
}
```
if(out_addr[stage] == '0') { // send from top node of element
   // check whether output zero has been used, if not allow this
   // packet to go through & record that it has been used; else
   // discard this packet
   if(currentElement -> topFlag == 0) {
       nextElement = currentElement->zero;
       currentElement -> topFlag = 1;
   } else { // element node busy, packed dropped.
       blocked = 1;
   }
}

else { // send from bottom node of element
   if(currentElement -> bottomFlag == 0) {
       nextElement = currentElement->one;
   } else { // element node busy, packed dropped.
       blocked = 1;
   }
}
Types of Banyan blocking

**Internal blocking (†):** Contention for an output port of a *component switch* in one of the early stages.

**Output port blocking (‡):** Contention for an output port of a switch in the last stage.

Output port blocking may even occur internally (§)
Probability of internal blocking

Blocking probability is high, e.g. if priority is given to leftmost input.

(Exercise: Find a formula expressing how high, assuming random input→output mappings)
Dealing with Banyan internal blocking

**Buffering** within the switching network

**Dilation:**

- Multiple planes of latter stages
- Move traffic blocked in one plane to higher plane ($R \rightarrow G \rightarrow B, Y$)
- Expensive: number of switches in each stage increases exponentially with stage number for the worst case.

**Preface Banyan with network** to

- reduce blocking (distribution network $\rightarrow$ Benes)
- avoid blocking (Sorting networks ...)

Copyright © 4/3/11 Tim Moors
Motivation for sorting networks

The blocking could have been avoided if only the information destined to each output was present on the correct input.

=> Preface Banyan network with a sorting network, and a Shuffle Exchange

Destinations: 000 010 111 100 001 011 110 101

Sort

Shuffle Exchange
\textit{Examples}† of non-blocking with sorting

Destinations are multiples of 2  Destinations are multiples of 3  Destinations: 1, 3, 4, 5, 7

† Not a proof that sorting+shuffling always prevents blocking
Sorting outline

- Isn’t sorting complicated? As complicated as switching?
- Sorting networks
  - Unlike serial sorting algorithms, sequence of comparisons must be independent of data values to allow fixed parallel implementation.
  - Issues (covered in ensuing slides):
    - Bitonic sequences
    - Batcher’s sorting of Bitonic sequences
    - How to create a bitonic sequence
    - Batcher-Banyan switches
    - Trapping sorted duplicates

http://en.wikipedia.org/wiki/Sorting_network
Sorting is simpler than switching

Both sorting and switching reorder items, but

1. Switch must spread after sorting to account for idle ports
   Input port: a b c d e f g h
   Destination: \[101_a \quad \cdots \quad b \quad 111_c \quad 001_d \quad 010_e \quad 110_f \quad \cdots \quad g \quad 011_h\]
   Sorted: \[001_d \quad 010_e \quad 011_h \quad 101_a \quad 110_f \quad 111_c \quad \cdots \quad x \quad \cdots \quad x\]
   Switched: \[\cdots \quad x \quad 001_d \quad 010_e \quad 011_h \quad \cdots \quad x \quad 101_a \quad 110_f \quad 111_c\]

2. Switch must deal with duplicates
   e.g. inputs a & d contend for port 101
   Destination: \[101_a \quad \cdots \quad b \quad 111_c \quad \mathbf{101_d} \quad 010_e \quad 110_f \quad \cdots \quad g \quad 011_h\]
   Sorted: \[010_e \quad 011_h \quad \mathbf{101_a} \quad \mathbf{101_d} \quad 110_f \quad 111_c \quad \cdots \quad x \quad \cdots \quad x\]
   Switched: \[\cdots \quad x \quad \cdots \quad 010_e \quad 011_h \quad \cdots \quad x \quad \mathbf{101_a} \quad 110_f \quad 111_c \quad \mathbf{101_d}\]
   i.e. adding a sorter \(\neq\) (\(<\)cost) repeating the switching process
Aside: Bitonic sequences

A bitonic sequence is either:

A. a concatenation of an increasing sequence and a decreasing sequence, i.e. \( \{a_1, a_2, \ldots, a_{2k}\} \) and

\[
\exists k : a_1 \leq a_2 \leq a_3 \ldots a_k \geq a_{k+1} \geq a_{k+2} \ldots \geq a_{2k}
\]

or

B. a sequence that can be shifted cyclically to become A.

e.g.

\[
1 2 3 8 7 6 5 4 \quad 8 7 1 2 3 4 5 6 \quad 4 5 6 8 7 1 2 3
\]

Theorem: Any bitonic sequence that has been arbitrarily split in two and had the two parts reversed is also bitonic.

e.g. \( 8 7 1 2 3 \mid 4 5 6 \rightarrow 3 2 1 7 8 6 5 4 \rightarrow 1 7 8 6 5 4 3 2 \)
Batcher’s sorting of bitonic sequences

Batcher showed† that given a bitonic sequence \( \{a_1 \ldots a_{2k}\} \)
then the sequences
\[
\{\min(a_1,a_{k+1}), \min(a_2,a_{k+2}), \ldots \min(a_{k-1},a_{2k})\} \quad \text{and}
\{\max(a_1,a_{k+1}), \max(a_2,a_{k+2}), \ldots \max(a_{k-1},a_{2k})\}
\]
• are also both bitonic, and
• no number in the sequence of minima exceeds any number in the sequence of maxima

e.g. \( \{8 \ 7 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6\} \rightarrow \)
\[
\{3=\min(8,3), 4=\min(7,4), 1=\min(1,5), 2=\min(2,6)\} \quad \text{and}
\{8=\max(8,3), 7=\max(7,4), 5=\max(1,5), 6=\max(2,6)\}
\]
i.e. can sort a bitonic sequence by dividing into two sub-sequences (one of minima and one of maxima), and repeating on sub-sequences.
\[
\{3,4,1,2\} \{8,7,5,6\} \rightarrow \{1,2\} \{3,4\} \{5,6\} \{8,7\} \rightarrow 1,2,3,4,5,6,7,8
\]

How to create a bitonic sequence

Use a “odd-even mergesort”† to make sequence bitonic before feeding into Batcher sorting network.
Aim: out.\(i\) < out.\(j\) \(\forall\ j>i\)

- Sort groups of 2
  
  \[
  \text{if}(a1<a2) \ {\text{out.}1=a1; \text{out.}2=a2;} \ \text{else} \ {\text{out.}1=a2; \text{out.}2=a1;}
  \]

- Merge pairs of groups of 2 to form groups of 4. Sort group of 4.
  
  \[
  \text{if}(a1<a3) \ {\text{out.}1=a1; \text{tmp.lo}=a3;} \ \text{else} \ {\text{out.}1=a3; \text{tmp.lo}=a1;}
  \]

  \[
  \text{if}(a2>a4) \ {\text{out.}4=a2; \text{tmp.hi}=a4;} \ \text{else} \ {\text{out.}4=a4; \text{tmp.hi}=a2;}
  \]

  \[
  \text{if}(\text{tmp.lo}<\text{tmp.hi}) \ {\text{out.}2=\text{tmp.lo}; \text{out.}3=\text{tmp.hi};} \\
  \text{else} \ {\text{out.}2=\text{tmp.hi}; \text{out.}3=\text{tmp.lo};}
  \]

  note that prior sorting of sub-groups helps sorting of merged group

- repeat for larger groups of 8, 16, ...

† Like a conventional merge sort, but comparisons are independent of data
Odd-even Merge sort  
Batcher sort  

Legend  
\[
\begin{align*}
\uparrow & = \text{Ascending sort} \\
\downarrow & = \text{Descending sort} \\
\square & \times \square & = \text{to make bitonic list}
\end{align*}
\]

Numerical example from C. Partridge: *Gigabit Networks*, p. 115
Trap modules

Sorting alone doesn’t resolve output port contention
=> trap excessive (>1) packets destined to one port

Starlite switch: **Recirculate** duplicates

Resequencing is needed. Often give priority to recirculated packets to avoid prolonged resequencing.
Benes networks

- Essentially two Banyan networks, one the mirror image of the other
- 2\textsuperscript{nd} half = standard Banyan network
- 1\textsuperscript{st} half = “distributor”: used to shift inputs so as to reduce blocking. Switch controller assigns path through distributor.
- A $P \times P$ Benes network has $s = 2 \log_2 P - 1$ stages, with each stage having $P/2$ 2\times2 switching elements (4 crosspoints) i.e. $N_x = 4P \log_2 P - 2P$

- The Cisco CRS-1 uses “a unique 3-Stage Benes topology switching fabric”