Switch Fabrics

Switching Technology S38.3165
http://www.netlab.hut.fi/opetus/s383165

Switch fabrics

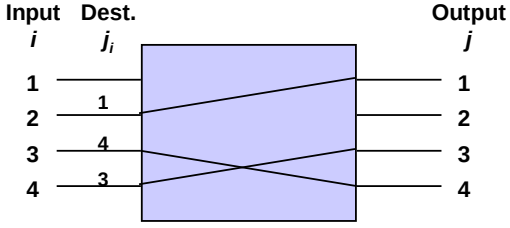
- Multi-point switching
- Self-routing networks
- **Sorting networks**
- Fabric implementation technologies
- Fault tolerance and reliability

Sorting networks

- Types of blocking
  - Internal blocking
  - Output blocking
  - Head of line blocking
- Sorting to remove internal blocking
- Resolving output conflicts
- Easing of HOL blocking

Internal blocking

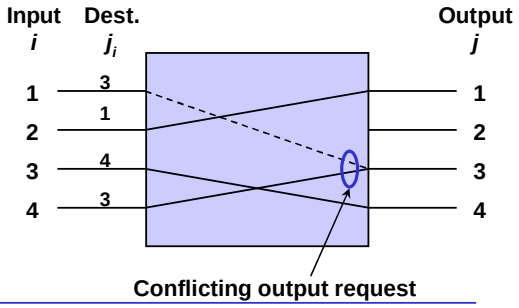
- Internal blocking occurs at the internal links of a switch fabric
- In a switch fabric, which implements synchronous slot timing, internal blocking implies that some input (i) to output (j) connection cannot be established (even if both are idle ones)
- Internally non-blocking switch makes all requested connections (i, j_i), provided that there are no multiple request to the same output ($j_i \neq j_{i'}$ if $i \neq i'$, $1 \leq i,j \leq N$)



Connection pattern = {(2, 1), (3, 4), (4, 3)}

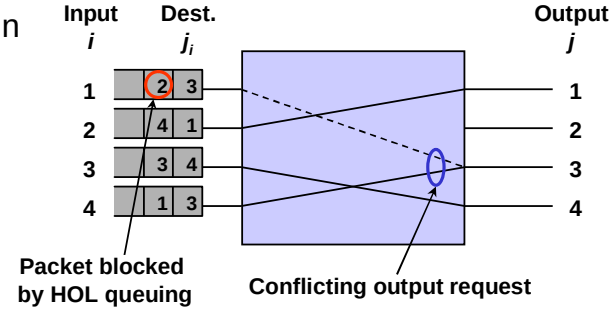
Output blocking

- Internally non-blocking switch can block at an output of a switch fabric due to conflicting requests, i.e., $j_i = j_{i'}$ for some $i \neq i'$
- When an output conflict occurs, switch should connect one of the conflicting inputs to requested output => output conflict resolution
- Major distinction between a circuit and packet switching node
  - a packet switching node must solve output conflicts per time-slot (time-slots are not assigned beforehand)
  - a circuit switching node solves possible output conflicts and assigns a time-slot for entire duration of a connection beforehand



Head of line (HOL) blocking

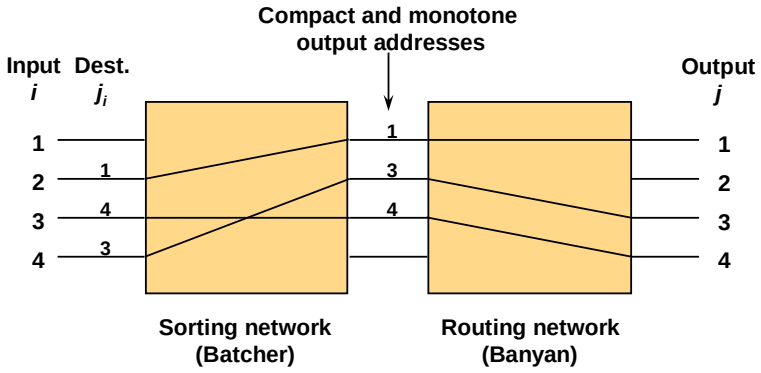
- Packets not forwarded due to output conflict are buffered
  => more delay experienced
- Buffered packets normally served in a FCFS (First Come First Served) manner
  => HOL blocking introduced at the input queues
- Packet facing HOL blocking may prevent the next packet in the queue to be delivered to a non-contended output
  => throughput of a switch reduced



Sorting to remove internal blocking

- If connection requests at the inputs of a banyan network are compact and in strictly increasing order
  => input-output paths are link-disjoint
  => banyan internally non-blocking
- A method for building an internally non-blocking network is to apply a sorting network in front of a banyan network to generate a strict increasing order of destination addresses for the banyan network
- A sorting network connects an input ***i***, which has a connection request to output j_i, to an output of a sorting network according to the position of j_i in the sorted list of destination requests (see figure)
- Sorting networks can be formed by interconnecting nodes of smaller sorting networks (such as 2x2)
- Self-routing should be applied in the sorting network

Internally non-blocking and self-routing switch



Sorting to remove internal blocking

- A permuted list ($a_1, a_2, \ldots, a_N$) can be restored to its original order by sorting
- A switching network for a maximal connection pattern can be obtained from a sorting network by treating 2x2 sorting elements as 2x2 switching elements
- Asymptotic lower bound for 2x2 sorting elements to build a *N*x*N* sorting network is $N\log_2 N$ (as for a respective switching network) - however, no sorting network found so far to obtain this bound
- Sequential merge-sorting process can be used to obtain $N\log_2 N$ bound for the number of binary sorts

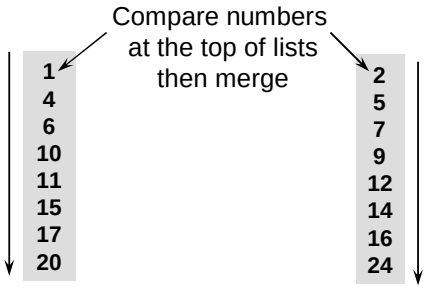
Merge-sorting algorithm

Merge-sorting algorithm

- Input : unsorted list $A_N = (a_1, a_2, \ldots, a_N)$
- Sort procedure:
  Sort (A_N) = Merge {Sort$(a_1, \ldots, a_{½N})$, Sort $(a_{½N+1}, \ldots, a_N)$}
- Merge procedure:
  Merge $\{(a_1, \ldots, a_m), (a'_1, \ldots, a'_m,)\}$
  = $\{a_1, (\text{Merge } ((a_2, \ldots, a_m), (a'_1, \ldots, a'_m,))\}$ if $a_1 \leq a'_1$
  = $\{a'_1, (\text{Merge } ((a_1, \ldots, a_m), (a'_2, \ldots, a'_m,))\}$ if $a_1 > a'_1$
- Procedure Merge, called by procedure Sort, takes two sorted lists and merges them by comparing the smallest elements in each of the two sorted lists

Merge-sorting algorithm (cont.)

- Merging of two sorted lists (N/2 numbers in each) requires *N* binary sorts
- Total complexity of sorting *N* numbers, which are in any order, is $C(N) = 2C(N/2) = N + 2(N/2 + 2C(N/4)) = \ldots = N\log_2 N$
- Due to sequential nature of procedure Merge the sorting takes at least O(*N*) time



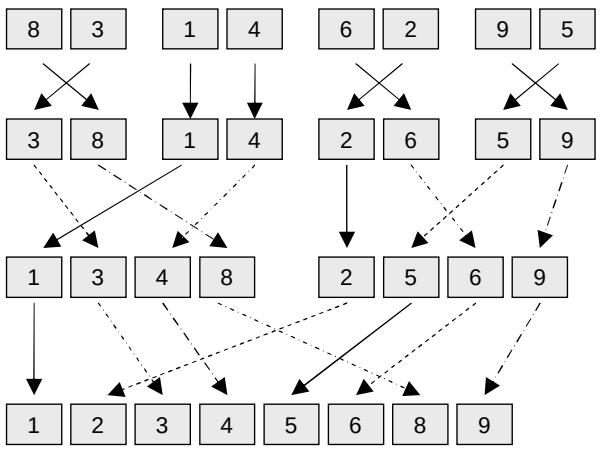
Merge-sorting example

Sort the set {8, 3, 1, 4, 6, 2, 9, 5}

Step 1: 4 times 2 sorts

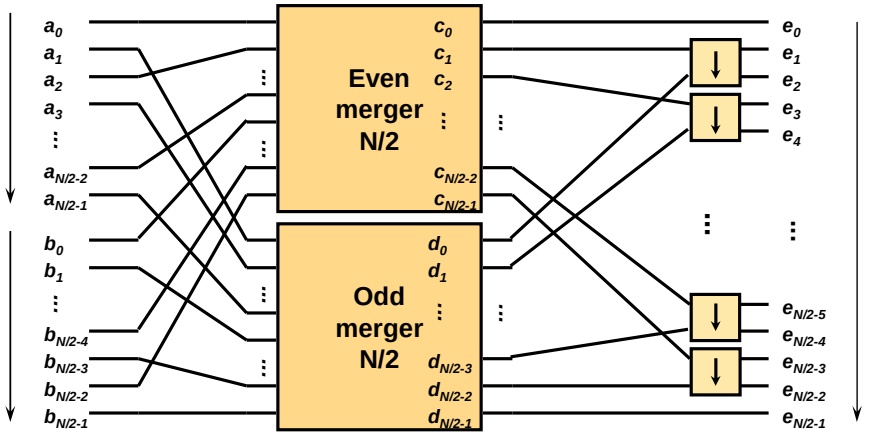
Step 2: 2 times 4 sorts

Step 3: 1 time 8 sorts



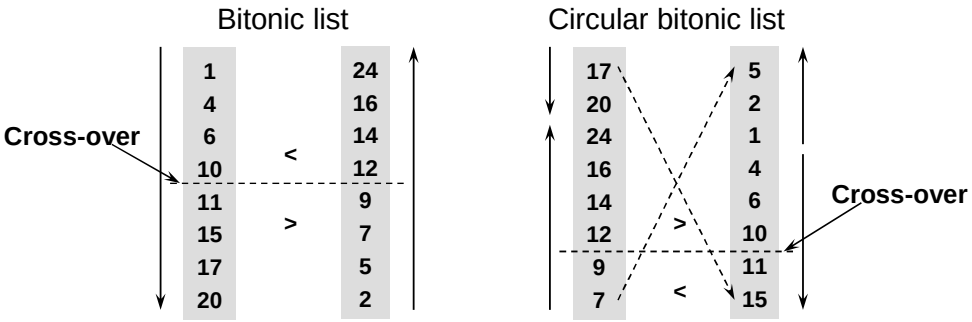
Odd-even merging

Recursive construction of an odd-even merger
- number of sorting stages is $\log_2 N$
- number of sorting elements is $0.5N\,[\log_2 N\text{-}1]+1$



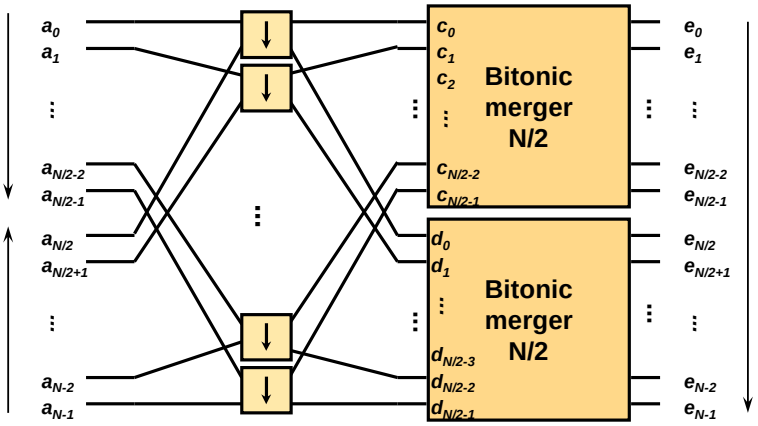
Bitonic list

- Bitonic list $A_N = (a_1, a_2, \ldots, a_N)$ is a list for which it holds that $a_1 \le a_2 \le \ldots \le a_{k-1} \le a_k$ and $a_k \ge a_{k+1} \ge \ldots \ge a_{N-1} \ge a_N$ $\quad (1 \le k \le N)$
- Unique cross-over property - when comparing a monotonically increasing list with a monotonically decreasing list, there is at most one position where the two lists cross-over in their values (see figures)



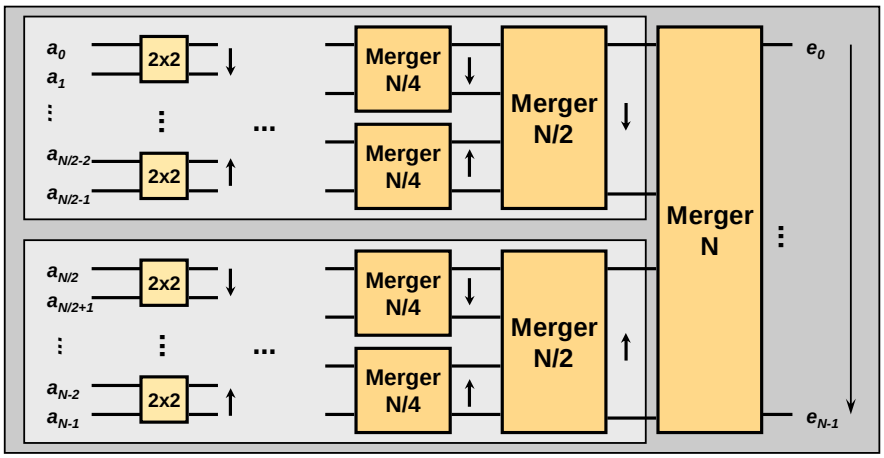
Bitonic merging

Recursive construction of a bitonic merger
- number of sorting stages is $\log_2 N$
- number of sorting elements is $0.5N \log_2 N$



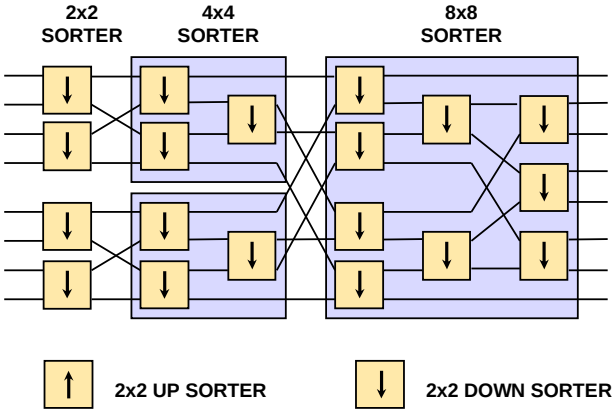
Sorting by merging

Recursive construction of a sorting by merging network
- number of sorting stages is $0.5N\log_2 N(\log_2 N + 1)$



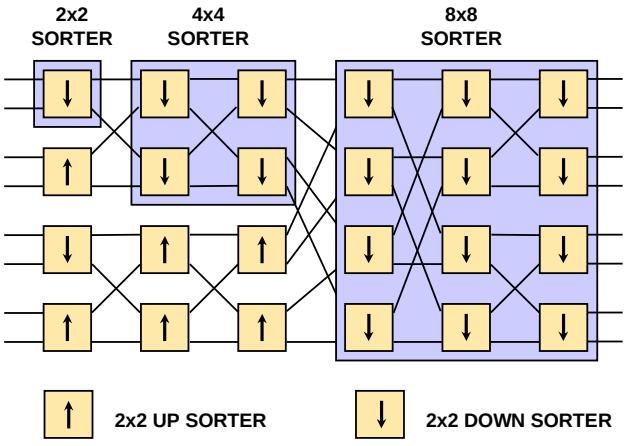
Odd-even sorting network example

- Number of sorting stages is $0.5\log_2 N(\log_2 N + 1)$
- Number of sorting elements is $0.25N[\log_2 N(\log_2 N - 1) + 4] - 1$

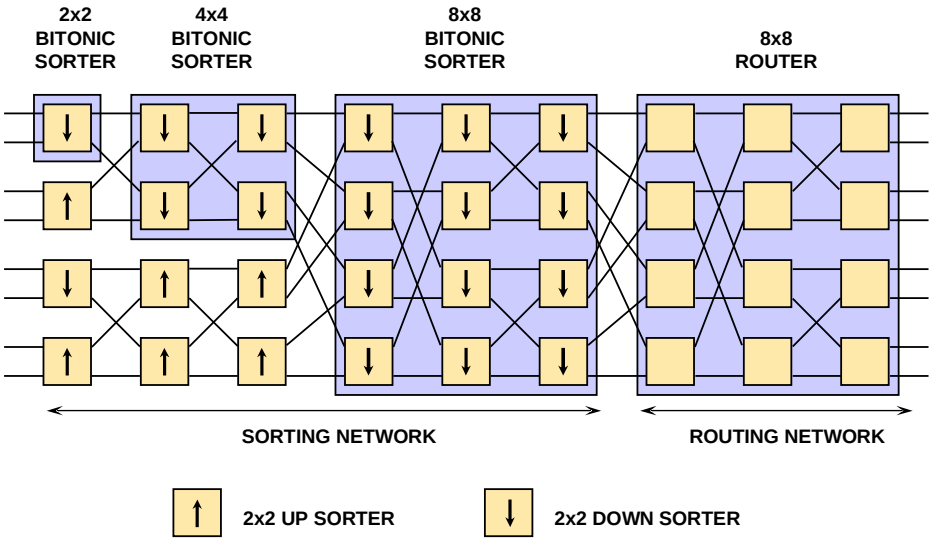


Bitonic sorting network example

- Number of sorting stages is $0.5\log_2 N(\log_2 N + 1)$
- Number of sorting elements is $0.25N\log_2 N(\log_2 N + 1)$



Batcher-Banyan self-routing network

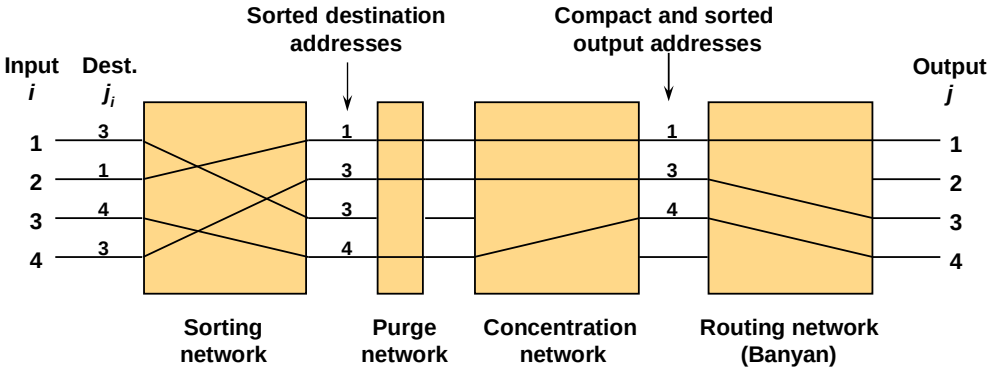


Resolving output blocking

- Packet switches do not maintain a scheduler for dedicating time-slots for packets (at the inputs)
  => output conflicts possible
  => output conflict resolution needed on slot by slot basis
- Output conflicts solved by
  - polling (e.g. round robin, token circulation)
    - do not scale for large numbers of inputs
    - outputs just served have an unfair advantage in getting a new time-slot
  - sorting networks (making a banyan network internally non-blocking)
- An example of sorting networks is *sort-purge-concentrate* network
  - when sorting self-routing addresses, duplicated output requests appear adjacent to each other in the sorted order (see figure)
    - either one has to be purged (deleted)
    - successful delivery is acknowledged and purged packets are re-sent

Sort-purge-concatenate network

- A sorting network can easily handle packet priority by
  - adding a priority field in the self-routing address
  - higher priority packets are placed in a favorable position before purging
  - support of priority is an essential feature when integrating circuit and packet switching in a sort-banyan network

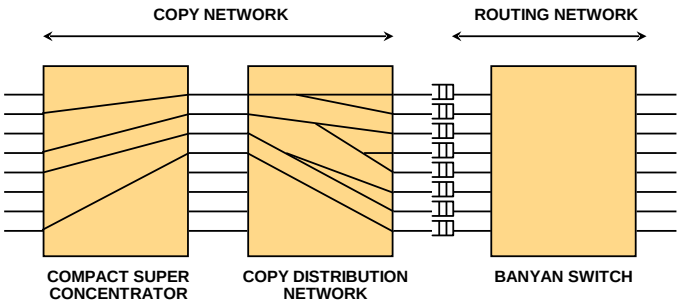


Resolving HOL blocking

- HOL blocking solved by
  - allowing packets behind a HOL packet to contend for outputs
  - allowing delivery of multiple conflicting HOL packets to an output buffer (during a time slot)
    - multiple rounds of arbitration for sort-banyan network
    - multiple planes of sort-banyan networks
  - a good solution is to implement multiple input buffers (one for each output if possible) and if the packet in turn cannot be transmitted due to HOL, transmit an other packet from another buffer

Construction of a multipoint packet switch

- Multipoint switch can be constructed by cascading a copy network and a point-to-point (routing) network
  - copy network is a cascade of a compact super-concentrator (e.g. reverse banyan network) and copy distribution network (e.g. multicast banyan network)

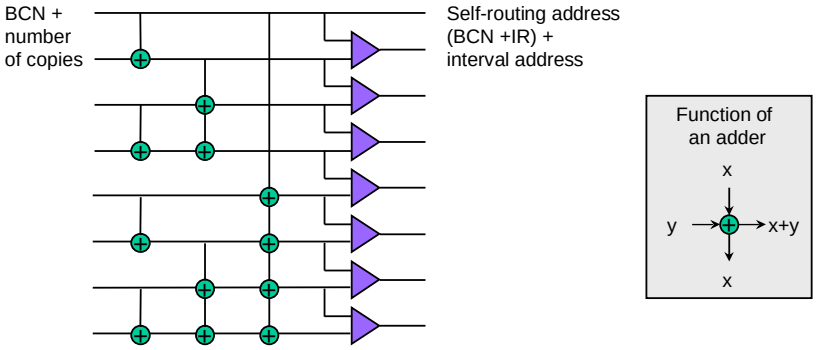


Multipoint packet switch

- In a self-routing multipoint switch
  - incoming packets may be destined to multiple outputs
  - if packets carry all needed destination addresses, headers would be variable length (and long)
  - header problem can be avoided by labeling the packet (to be copied) by a virtual address (Broadcast Channel Number, BCN) and number of copies - each copy (with the same BCN) is given a distinct destination address
- Compact super-concentrator connects active inputs so that their destination addresses form a compact and sorted set at the outputs
- Copy distribution network establishes a multicast tree for each multicast connection
- Copy network becomes a self-routing one by providing
  - self-routing address for compact super-concentrator
  - self-routing interval address for copy distribution network

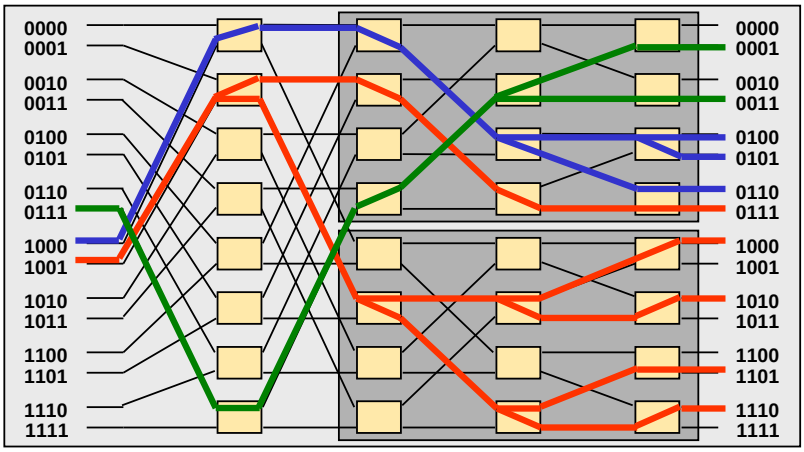
Multipoint packet switch (cont.)

- Running-sum-adder is user for calculating self-routing address and self-routing interval address

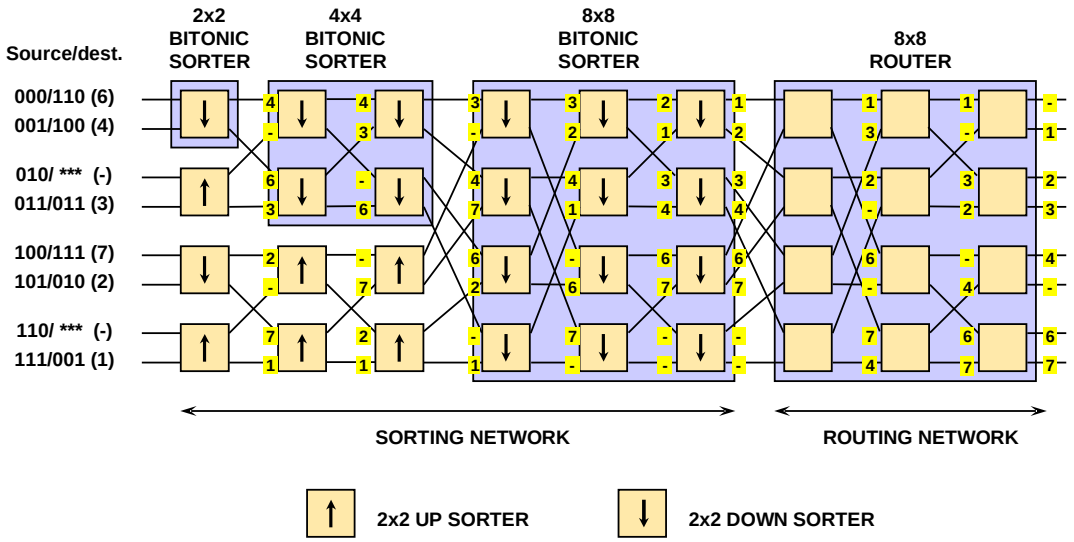


Example of a multicast banyan network

Multipoint connections:
$X_1 = 7$ => $y_1 = \{1, 3\}$
$X_2 = 8$ => $y_2 = \{4, 5, 6\}$
$X_3 = 9$ => $y_3 = \{7, 8, 10, 13, 14\}$



Batcher-Banyan example



Switch fabrics

- Multipoint switching
- Self-routing networks
- Sorting networks
- **Fabric implementation technologies**
- Fault tolerance and reliability

Fabric implementation technologies

- Time division fabrics
  - Shared media
  - Shared memory
- Space division fabrics
  - Crossbar
  - Multi-stage constructions
- Buffering techniques

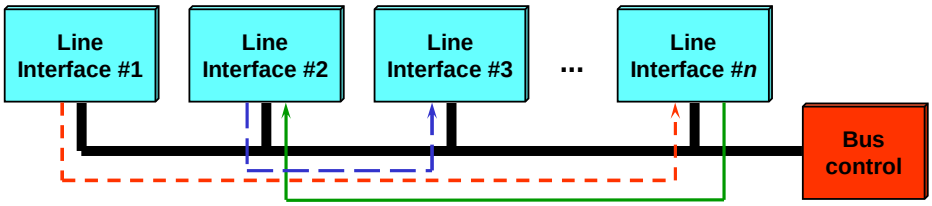
Time division fabrics

- **Shared media**
  - Bus architectures
  - Ring architectures
- **Shared memory**

Shared bus

Bus architecture

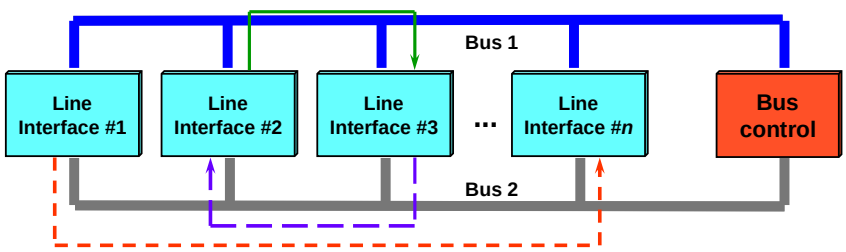
- Switching in time domain, but time and space switching implementations enabled
- Easy to implement and low cost index (= *N*)
- One time-slot carried through the bus at a time
  => limited throughput (multi-casting possible)
  => low number of line interfaces
  => limited scalability



Shared bus (cont.)

Bus architecture

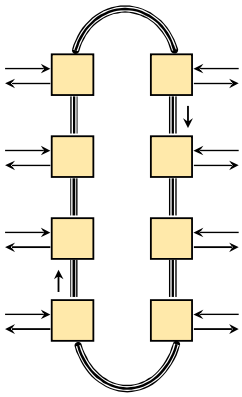
- Internally non-blocking implementations require high capacity switching bus => throughput ≥ aggregate capacity of line interfaces
- Inherently a single stage switch, but TST-switching possible if line-cards support time division multiplexing (TDM)
- Multiple-bus structures can be used to improve reliability and increase throughput



Ring architectures

Ring architecture

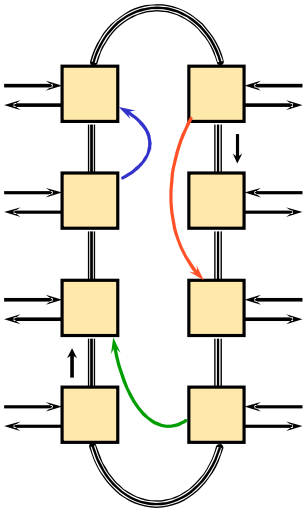
- Rings coarsely divided into source and destination release rings
  - in source release (SR) rings only one switching operation in progress at a time
    => limited throughput (like a shared bus)
  - destination release (DR) rings allow spatial reuse, i.e., multiple time-slots can be carried through the ring simultaneously
    => improved throughput
- Switching in time domain, but time and space switching implementations enabled
- Usually easy to implement and low cost index (= *N*)
- Scales better than a shared bus



Ring architectures (cont.)

Ring architecture

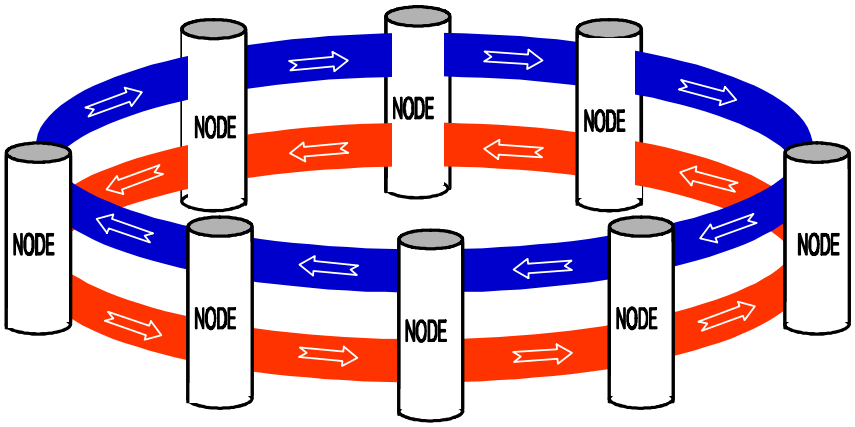
- Internally non-blocking implementations require that throughput of a ring bus ≥ aggregate capacity of line interfaces
- Throughput can be improved by implementing parallel ring buses - control usually distributed
  => MAC implementations may be difficult
- Multi-casting relatively easy to implement
- Inherently a single stage switch, but TST-switching possible if line-cards support TDM
- Multiple rings can be used to implement switching networks



Ring architectures (cont.)

Dual ring architecture

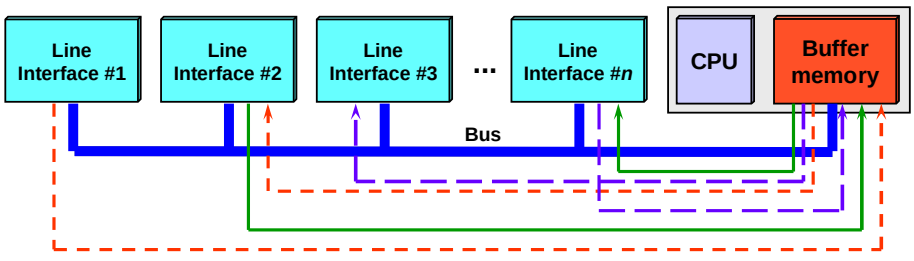
- Multiple rings used to improve throughput, decrease internal blocking, improve scalability and increase reliability



Shared memory

Shared memory architecture

- Switching in time domain, but time and space switching implementations enabled
- Inherently a single stage switch, but allows TST-switching if line-cards support TDM
- Easy to implement and low cost index (= *N*)



Shared memory (cont.)

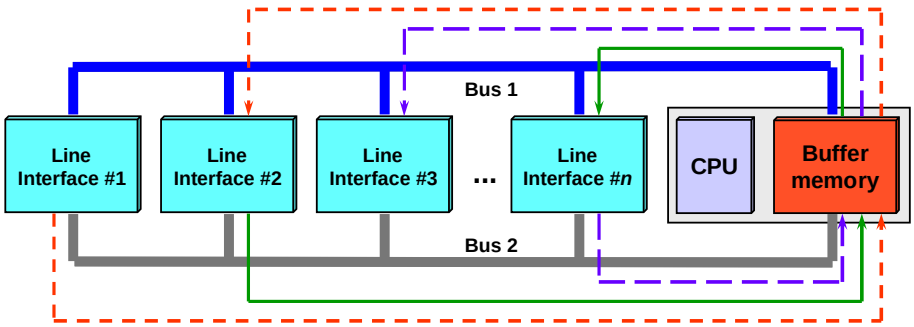
Shared memory architecture

- Every time-slot carried twice through the bus
  => low throughput
  => low number of line interfaces
  => limited scalability
- Internally non-blocking if throughput of a switching bus and speed of shared memory ≥ aggregate capacity of line interfaces
- Performance can be improved by dual bus architecture or replacing the bus with a space switch (such as crossbar)

Shared memory (cont.)

Shared memory architecture

- Dual-bus architecture improves throughput, decreases internal blocking, improves scalability and increases reliability
- Memory speed requirement equal to that of single bus solutions



Dimensioning example

A shared memory architecture, which uses a shared bus to connect line interfaces to the memory, is used to implement a switching equipment. The bus is 32 bits wide and bus clock is 150 MHz. Three clock cycles are needed to transfer a 32 bit word through the bus and 20 % of the bus capacity is used for other than switching purposes. How many E1 interfaces can be supported by the switch ? What is the required memory speed ?

Dimensioning example (cont.)

Solution:

The bus transfers 32-bit wide data words at the speed of $(150/3)x10^6$ transfers/s = $50x10^6$ transfers/s.

If the bus carries an eight bit time-slot (of a 64 kbit/s PDH channel) across the bus at a time, a single bus solution can transfer $0.8 \times (150/3)$ Mbytes/s = 40 Mbytes/s

In a single bus solution, half of the bus capacity (20 Mbytes/s) is used for storing time-slots to the memory and another half for reading time-slots from the memory

=> during a 125 μs period (= duration of an E1 frame) the bus switches $(125x10^{-6})x(20x10^6)$ bytes = 2500 incoming (and outgoing) time-slots and thus the number of supported (input and output) E1 link pairs is $2500/32 \approx 78$

Dimensioning example (cont.)

Solution (cont.):

If the memory interface logic accesses the buffer memory at the speed of the data bus, then the memory speed requirement is $1/(50 \times 10^6)$ s = 20 ns.

If the memory interface logic allows lower access rate than the data bus transfer rate, then the memory speed requirement is $1/(40 \times 10^6)$ s = 25 ns.

Throughput of the switching system could be increased by adding a 32 bit receiver-register to the shared switch memory block, which enables to transfer 4 time-slots (in parallel) through the bus at a time. By doing so, the throughput of the bus gets four fold and the number of supported E1 link pairs increases to 312. However, the time-slots are still written one by one to the switch memory, and the corresponding memory speed requirement is (depending on the memory interface logic) either 5.0 ns or 6.25 ns.

Space division fabrics

- **Crossbar**
- **Multi-stage constructions**

Crossbar

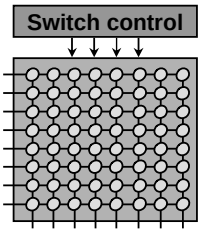
Crossbar architecture

- Inherently a space division switch
- Allows to build TST-switches if interfaces implement TDM functionality
- Hard to implement large switches due to complicated control schemes
  => high cost index (= N^2)
- Commercial high-speed *NxN* crossbar components enable modular and relatively inexpensive fabric constructions, but still control of the switch is a problem

Crossbar (cont.)

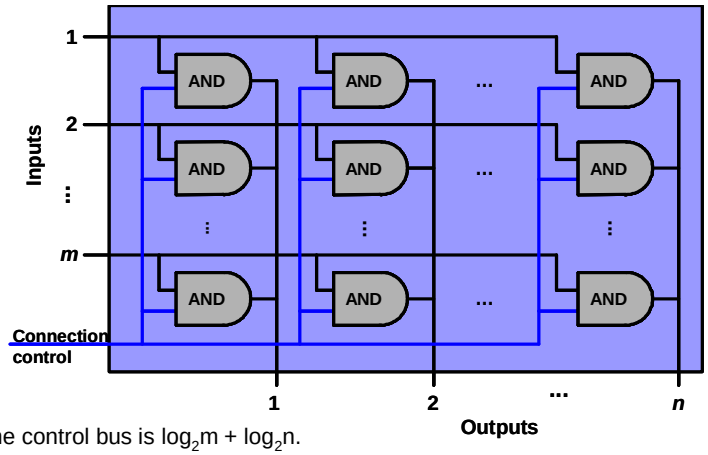
Crossbar architecture

- Inherently a strict-sense non-blocking fabric architecture
- Possible to carry *N* time-slots through the switch at a time
  => high throughput
  => possible to implement a large number of line interfaces
  => scales well within the limits of the available modular components
  => scaling up means increase of cross-point count from *N*x*N* to (*N*+*k*)x(*N*+*k*)
- Multi-casting easy to implement



Crossbar (cont.)

Example implementation of a crossbar

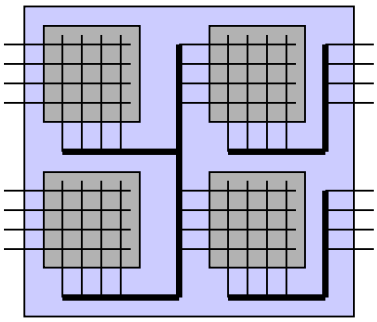


Width of the control bus is $\log_2 m + \log_2 n$.

Crossbar (cont.)

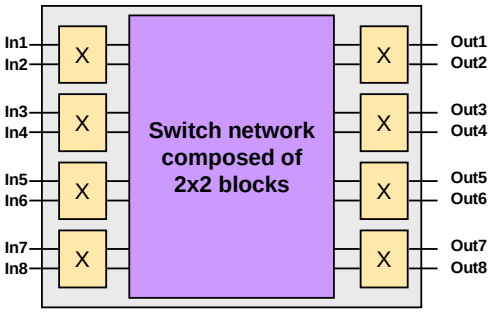
An 8x8 switch constructed of four 4x4 crossbar blocks

Notice that doubling of input/output count increases the number of crossbar components from one to four.



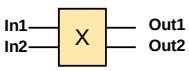
Multi-stage building blocks

- Multi-stage switches normally constructed of 2x2 switching blocks
- Implemented usually in FPGAs (Field Programmable Gate Arrays) and/or ASICs (Application Specific Integrated Circuit)
  - FPGA for experimental use and low volume production
  - ASICs for high volume production
- Batcher-banyan network most popular
- Used to implement space division switching
- Allows to build TST-switches if interfaces support TDM functionality



Multi-stage building blocks (cont.)

- Hard to implement large circuit switches due to complicated control schemes (especially rearrangeable fabrics)
  => high cost index (~ $CN\log_2 N$)
- Suitable for packet switching when self-routing functionality included
- Fixed length time-slot implementations favored to obtain strict-sense non-blocking fabrics
- Possible to carry N time-slots through the switch at a time
  => relatively high throughput
  => scalable only if larger networks can be factored using smaller NxN components
  => scaling up means increase of cross-point count from ~ $CN\log_2 N$ to ~ $C(N+k)\log_2(N+k)$



Problems with multi-stages

- Path search required
- Fast connection establishment implies need for fast control system
  => part of switching capacity is lost if control system is not fast enough
- Multi-cast is not self evident, because multi-cast complicates path search and control scheme and increases blocking probability
- Multi-slot connections (i.e. several slots used for a particular connection) complicate matters
  - especially if path delay is not constant, e.g., slots belonging to the same connection may arrive to outputs in different order than they arrived at the inputs
  - blocking increases

Trends in fabric technologies

- Memory technology getting faster and faster
- Current SRAM (Static Random Access Memory) technology allows easy implementations of large PDH switches, e.g., full matrix for 8000 E1 (2M) PDH circuits - bigger fabrics hardly needed in narrow band networks
  => in narrow band networks the trend over the last 10 years has been to build full matrix fabrics based on shared memory
- However, when striving for broadband communications, memory based switch fabrics do not scale to bandwidth needs
  => multi-stage and crossbar switches have their change

Trends in fabric technologies (cont.)

- Multistage fabrics were “reinvented” at the advent of ATM
  - ATM suits perfectly for fixed length time-slot switching
  - self-routing and sorting applies for ATM cell routing
  - blocking and buffering causes headache

  => in spite of huge research effort, there have been very few commercial multi-stage fabrics available (mostly proprietary ASICs)
- Development of IC technologies, increased packing density (number of gates/chip) and increased speed have enabled crossbar fabrics suitable for high-speed switching applications (N = 2 … 64 and line rate 2.5 … 40 Gbit/s)
  - examples: Cx27399/Mindspeed, ETT1/Sierra, CE200/Internet Machines and PI140xx/Agere
- Packet switching and advent of optical networking favors multi-stages and crossbars

  => packet switching introduces a new problem - **buffering**

Technological tradeoffs in switch fabric design

- When trying to simplify path search and to speed up connection establishment

  => bus speed increases (inside fabric)

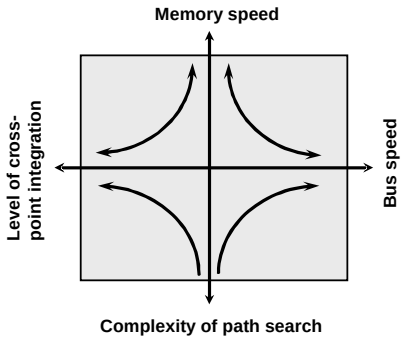
  => faster memory required => power consumption increases

  => integration level of a cross-point product needs to be increased

  => faster memory required, etc.
- If fast memory not available, use

  => crossbar fabrics (for small switches)

  => multistage fabrics (for large switches)
  - real switching capacity may be less than theoretical
  - minimization of cross-point count often pointless



Electronic design problems

- **Signal skew** - caused by long signal lines with varying capacitive load inside switch fabric and/or on circuit boards
- **Mismatching line termination** - caused by long signal lines combined with varying (high) bit rates
- **Varying delay** on bus lines - caused by differently routed bus lines (non-uniform capacitive load)
- **Crosstalk** - caused by electro-magnetic coupling of signals from adjacent signal lines
- **Power feeding and voltage-swing** - incorrectly dimensioned power source/lines cause non-uniform voltage and lack of adequate filtering causes fluctuation of voltage
- **Mismatching timing signals** - different line lengths from a centralized timing source cause phase shift and distributed timing may suffer from lack of adequate synchronization

Some design limitations

- Speed of available components vs. required wire speed and slot time interval
- Component packing density and power consumption vs. heating problem
- Maximum practical fan-out vs. required size of fabric
- Required bus length inside switch fabric
  - long buses decrease internal speed of fabric
  - diagnostics get difficult
- IPR policy
  - whether company wants to use special components or more general all-purpose components

Design optimization example

- An *N*x*N* switch fabric is to be designed and there are three alternative crossbar components ***a***, ***b*** and ***c*** available
  - **a** is an N_axN_a fabric component
  - ***b*** is an N_bxN_b fabric component
  - ***c*** is an N_cxN_c fabric component

  and $N_a<N_b<N_c \leq N$
- Component ***a*** has entered the market at time t_a, ***b*** at time t_b and ***c*** at time t_c
- Product development starts at t_{pd} and the switch product should come in the market at t_m. Components are expected to be available when the product development starts => $t_a < t_b < t_c \leq t_{pd} < t_m$
- Price of a component develops with time and is generally given by P(t)=*Cf(t)* + *D*, where *Cf(t)* is a time dependent and *D* a constant part of component's price
- Question: Which one of the three components to choose for constructing an *N*x*N* switch fabric ?

Design optimization example (cont.)

- As an example, let's assume that price of each component is a function of time and is given by $P(t)=Ce^{-t/T}+ D$,
  where *C*, *D* and *T* are component specific constants
  => $P_a(t)=C_ae^{-t/T_a}+ D_a$, $P_b(t)=C_be^{-t/T_b}+ D_b$ and $P_c(t)=C_ce^{-t/T_c}+ D_c$
- Number of alternative crossbar components needed to build an *N*x*N* switch
  => K_a = ceil$[N/N_a]^2$, K_b = ceil$[N/N_b]^2$, K_c = ceil$[N/N_c]^2$
- Individual component and total component costs as a function of time *t*
  => $P_a(t)=C_ae^{-(t- t_a)/T_a}+ D_a$ and $T_a(t)= K_a P_a(t)$, $t \geq t_a$
  => $P_b(t)=C_be^{-(t- t_b)/T_b}+ D_b$ and $T_b(t)= K_b P_b(t)$, $t \geq t_b$
  => $P_c(t)=C_ce^{-(t- t_c)/T_c}+ D_c$ and $T_c(t)= K_c P_c(t)$, $t \geq t_c$
- These functions can be used to draw price development curves to make comparisons

Design optimization example (cont.)

Numerical example:

- Let N = 64, N_a = 16, N_b = 32, N_c = 64, $T_a = T_b = T_c$ = 3 time units (years), C_a = 20,C_b = 50, C_c = 100 and D_a = 10, D_b = 20, D_c = 40 price units (euros)
- Product development period is assumed to be 1 time unit (year) and $t_b = t_a + 1.5$, $t_c = t_a + 3$, $t_m = t_a + 4$ => $t_{pd} = t_a + 3$
- Choosing that $t \geq t_o = t_{pd} = 0$ => $t_a = -3$, $t_b = -1.5$, $t_c = 0$ and $t_m = +1$
- Number of components needed K_a = 16, K_b = 4, K_c = 1
- Switch fabric component cost functions

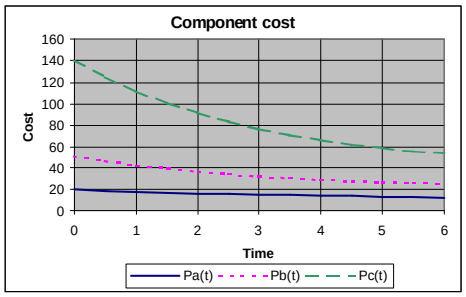
  => $P_a(t) = [20e^{-(t+3)/3} + 10]$ and $T_a(t) = 16 \times P_a(t)$

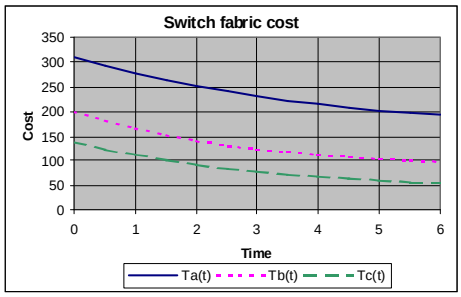
  => $P_b(t) = [50e^{-(t+1.5)/3} + 20]$ and $T_b(t) = 4 \times P_b(t)$

  => $P_c(t) = 100e^{-(t)/3} + 40$ and $T_c(t) = 1 \times P_c(t)$

Design optimization example (cont.)

Numerical example (cont.) :





- Although the price of component ***c*** is manifold compared to the price of component ***a*** or ***b***, ***c*** turns out to be the cheapest alternative
- Another reason to choose ***c*** is that it probably stays longest in the market giving more time for the switch product