Switch Fabrics

Switching Technology S38.165
http://www.netlab.hut.fi/opetus/s38165

Switch fabrics

- Basic concepts
- Time and space switching
- Two stage switches
- Three stage switches
- Cost criteria
- **Multi-stage switches and path search**

Recursive factoring of a strict-sense non-blocking network

- A strict-sense non-blocking network can be constructed recursively, but the size of network (number of cross-points) crows fast as the function of the number of inputs, namely $\boldsymbol{CNlog_2N}$
- Instead of starting with the smaller factor for *p* let's use switch blocks of $\sqrt{N} \times \sqrt{N}$
- Let $N = 2^n$ and $n = 2^l$ then we are factoring square switches with number of inputs and outputs being power of 2
  => condition for a strict-sense non-blocking network states that there are $r_2 \geq 2x2^{n/2} - 1$ second stage SBs
- Let choose $r_2 = 2x2^{n/2}$ then the sizes of the
  - 1st stage switches are $2^{n/2} \times 2^{n/2+1}$
  - 3rd stage switches are $2^{n/2+1} \times 2^{n/2}$
- Each of these can be made of two SBs each of size $2^{n/2} \times 2^{n/2}$

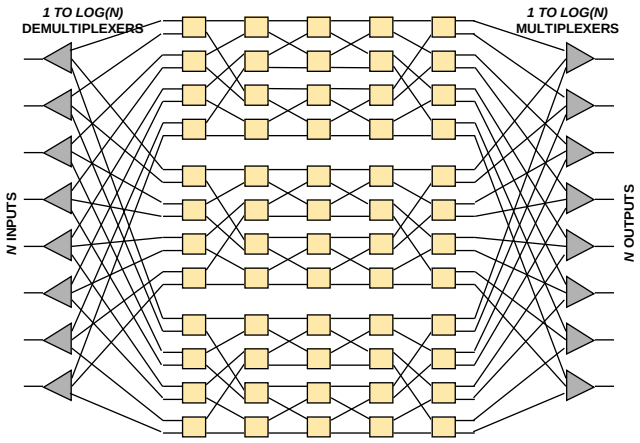
Recursive factoring of a strict-sense non-blocking network (cont.)

- 2nd stage switches are of size $2^{n/2} \times 2^{n/2}$
- The three stages consist of $6x2^{n/2}$ SBs, each of size $2^{n/2} \times 2^{n/2}$
- Let $F(2^n)$ be the cross-point complexity of an *N*x*N* switch then
- $F(2^n) = 6x2^{n/2}F(2^{n/2})$
  $= 6^l x2^{n/2+n/4+\ldots+1}F(2^1)$
  $< 6^l x2^n F(2)$
  $= N(log_2N)^{2.58}F(2)$
  $= 4N(log_2N)^{2.58}$
- The difference between rearrangeable and strict-sense non-blocking networks lies in the exponent for the log_2N term

Strict-sense non-blocking network with smaller number of cross-points

- Strict-sense non-blocking networks with smaller number of cross-points than $F(2^n) = 4N(log_2N)^{2.58}$ can be constructed
- One alternative is to use Cantor network, which is constructed using Benes networks, multiplexers and demultiplexers
  - ***i***-th input of Cantor network connected to ***j***-th input of ***j***-th Benes network using ***j***-th output of a 1x*m* demultiplexer
  - ***i***-th output of ***j***-th Benes network connected to ***i***-th output of Cantor network using ***j***-th input of a *m*x1 multiplexer
- When *N* is known, number of required Benes planes to have a strict-sense non-blocking Cantor network is ***m = log$_2$N***
- Since a Benes network has a cross-point count of $4Nlog_2N$, number of cross-points of a Cantor network is roughly $\mathbf{4N(log_2N)^2}$ (when ignoring cross-points of the multiplexers and demultiplexers

Cantor network



Cantor network strict-sense non-blocking

Proof:

- Markings
  - ***m*** number of parallel Benes networks
  - ***k*** number of a stage in a Benes network
  - **A(*k*)** number of reachable 2x2 SBs without rearrangements in stage *k* ($1 \le k \le log_2 N$) starting from an input of a Cantor network
- Reachable 2x2 SBs in consecutive stages
  - A(1) = *m*
  - A(2) = 2A(1) - 1
  - A(3) = 2A(2) - 2
  - $A(k) = 2A(k\text{-}1) - 2^{k-2} = 2^2 A(k\text{-}2) - 2x2^{k-2} = 2^{k-1}A(1) - (k\text{-}1)x2^{k-2}$
  - $A(log_2 N) = 2^{log_2 N - 1} m - (log_2 N - 1)\, 2^{log_2 N - 2}$
    $= \frac{1}{2} Nm - \frac{1}{4}(log_2 N - 1)N$

Cantor network strict-sense non-blocking (cont.)

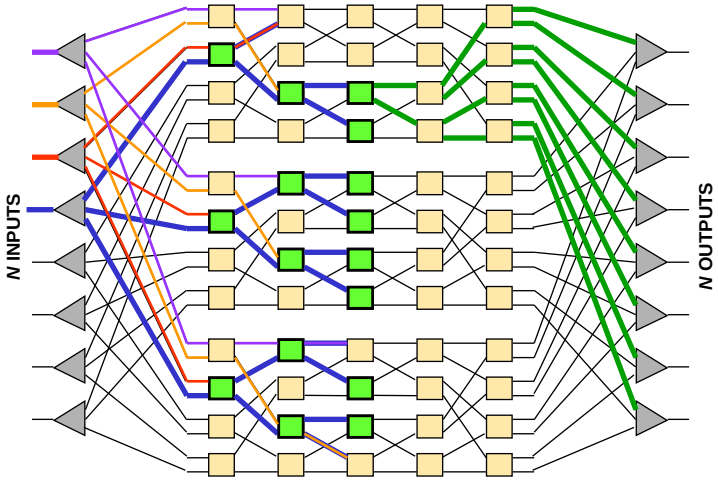
- Cantor network is symmetrical at the middle
  => the same number of center stage nodes are reachable by an output of a Cantor network
- Total number of SBs in center stages is *Nm*/2 (*m* Benes networks)
- If the number of center stage SBs reached by an input and an output exceeds *Nm*/2 then there must be a SB reachable from both
- Hence strict-sense non-blocking is achieved if

$$2\left[\frac{1}{2} Nm - \frac{1}{4}(log_2 N - 1)N\right] > \frac{Nm}{2}$$

$$\Rightarrow m > log_2 N - 1$$

Notice that a strict-sense non-blocking Cantor network is constructed of $log_2 N$ rearrangeably non-blocking Benes networks

Visualization of proof



Dimensioning example of Cantor network

Number of inputs and outputs of a switching network should be
N = 32 x 2048 = 2^{16} ≈ 64 000

- Number of multiplexers = 64 000
- Number of demultiplexers = 64 000
- Number of Benes networks $m = log_2 N$ = 16
  => number of outputs in demultiplexers = 16
  => number of inputs in multiplexers = 16

- Number of stages in Benes networks is $2log_2 N - 1 = 2 \times 16 - 1 = 31$
- Number of 2x2 SBs in each Benes network is
  $[2log_2 N - 1] \times (N/2) \approx 31 \times 2^{15}$
- Number of 2x2 SBs in all the *m* Benes networks is 31×2^{19}

Control algorithms

- Control algorithms for networks, which are formed recursively by three stage factorization, can be applied recursively
  - works well for strict-sense non-blocking networks when setting up connections one at a time
  - for rearrangeable networks adding just one connection may cause the connection pattern to change dramatically
    => adding a connection to a Benes network can be as complicated as reconnecting all input-output pairs
- Let's examine control algorithm for a Benes network, formed by factoring recursively
  - $N = 2^m$ inputs and outputs
  - start with a totally disconnected network and establish requested connection patterns

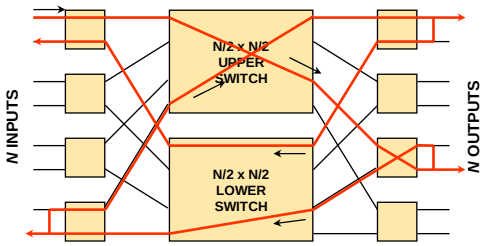
Looping algorithm

During the first factorization of an *N*x*N* switch each 2x2 input SB may be connected to a 2x2 output SB either via upper (***U***) or lower (***L***) *N*/2x*N*/2 switch (see figure)

1) *Initialization*
   Start with 2x2 input SB1 and mark it by ***S***
2) *Loop forward*
   Connect an unconnected input of ***S*** to desired output by upper switch ***U***. If no connection is required, go to 4.
3) *Loop backward*
   Connect the adjacent output of the output just visited to the desired input by the lower switch ***L***. If no connection is required, go to 4. Otherwise, the newly visited input SB becomes ***S***. Go to 2.

Looping algorithm (cont.)

4) *Start new loop*
Choose another SB, which has not been visited yet as ***S***. Go to 2.
If all connections for the *N*x*N* switch are made, the algorithm terminates at level *m*.



Looping algorithm (cont.)

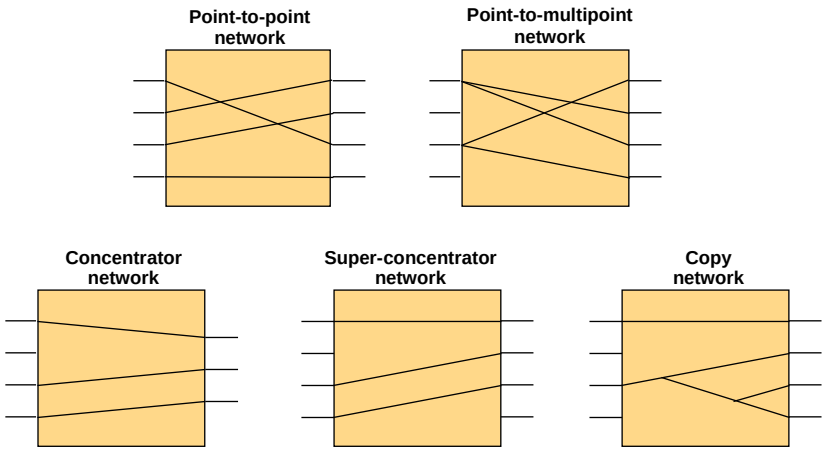
- Looping algorithm is applied recursively to establish connections for the upper switch ***U*** and lower switch ***L***
- Computation of paths is complex and time consuming and it can be shown that the total run time of the algorithm to compute paths for all inputs and outputs is proportional to $(log_2 N)^2$
- Looping algorithm
  - suits for circuit switching, because connections computed per call
  - not suitable for packet switching, because connections may have to be recomputed for all *N* input-output pairs within duration of a packet
  - dedicating a processor for each input and output SB
    => connection computations become faster, but exchange of path information between processors gets very complicated

=> Alternative switching architectures needed for packet switching

Switch fabrics

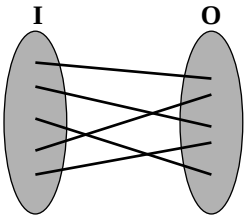
- **Multi-point switching**
- Self-routing networks
- Sorting networks
- Fabric implementation technologies
- Fault tolerance and reliability

Network types based on connection patterns



Graph presentation of connection patterns

Point-to-point switching



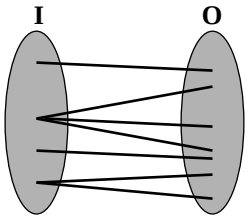
One-to-one connections

C = {(i,o)| i∈ I, o∈ O}

If (i,o) ∈ C and (i,o')∈ C => o=o'

If (i,o) ∈ C and (i',o) ∈ C => i=i'

Multi-cast switching



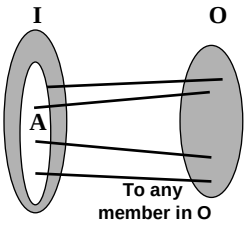
One-to-many connections

C = {(i,n_i)| i∈ I, n_i⊂O}

C - a logical mapping from inputs to outputs

Graph presentation of connection patterns (cont.)

Concentrator



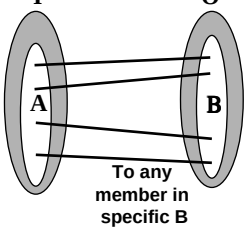
One-to-one connections

C = {(i,o)| i∈ A⊂I, o∈ O}

If (i,o)∈ C and (i,o')∈ C => o=o'

If (i,o)∈ C and (i',o) ∈ C => i=i'

Super-concentrator
(compact if B compact)



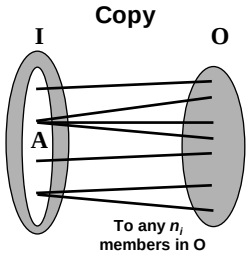
One-to-one connections

C = {(i,o)| i∈ A⊂I, o∈ B⊂O}

If (i,o)∈ C and (i,o')∈ C => o=o'

If (i,o)∈ C and (i',o)∈ C => i=i'

Graph presentation of connection patterns (cont.)



One-to-many connections

$C = \{(i,n_i) | \; i \in A \subset I, \; \Sigma n_i \in N\}$

Order and identity of outputs
n_i ignored (output unspecific)

Combinatorial bound

- $\zeta(G)$ is log_2 of the number of distinct and legitimate ***C*** realized by ***G***
- ζ measures combinatorial power of a graph (***G***), may bear no direct relationship to control complexity of finding a switch setting to realize a connection pattern
- ***R*** is number of cross-points in a switch fabric
- 2^R is the number of (on/off) states in a switch fabric of ***R*** cross-points => rough upper bound for the number of ***C***s in ***G*** is $\zeta \leq R$
- Better upper bound obtained by removing
  - all non-legitimate states, e.g., those in which two cross-points are feeding one output
  - one of states for which another state produces the same ***C***
- Such improvements not easily found

Combinatorial bound (cont.)

Let us look at the number of different ***C*** measured as ζ realized by different connection functions

- For each connection function, which defines a set of legitimate ***C***, we may compute the logarithm of the total number of distinct ***C***, marked by ζ
- A graph ***G*** is rearrangeably non-blocking if all such ***C*** can be realized by ***G***
  => we must have $\zeta \leq \zeta(G)$ for rearrangeably non-blocking ***G***

It follows that by observing the number of distinct ***C*** realized by different connection functions, we can find the lower bound of complexity for any rearrangeably non-blocking fabric.

Lower combinatorial bound for point-to-point connections

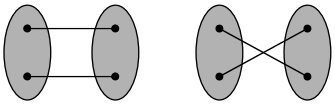
- *N*x*N* switch with full connectivity (any element in ***I*** can be connected to any distinct element in ***O***)
- Obviously network that can realize all maximal connection patterns can realize less than maximal patterns
- Number of ***C*** we want to realize equals to *N*!
- Sterling's approximation:
  => $N! \approx \sqrt{2\pi} N^{N+½} e^{-N} = \sqrt{2\pi} \exp_2(N\log_2 N - N\log_2 e + ½\log_2 N)$
  => $\zeta_{pt\text{-}pt} = \log_2 N! \approx N\log_2 N - 1.44N + ½\log_2 N = O(N\log N)$
- If 2x2 SBs are used, at least $N\log_2 N$ such SBs are needed to realize the *N*! possible maximal connection patterns

=> a point-to-point interconnection network has a complexity of $O(N\log N)$

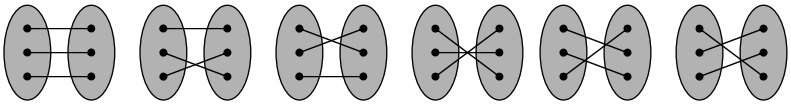
Visualization of point-to-point mappings

Number of connection patterns in point-to-point switching $L_c = N!$

N = 2 => $L_c = 2$



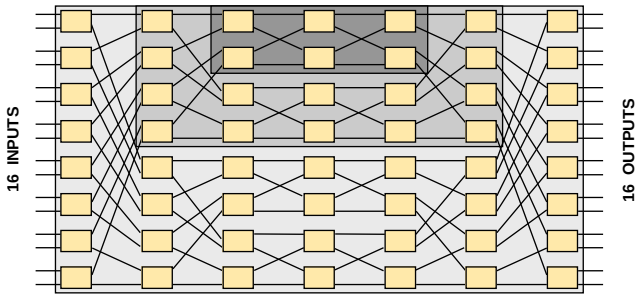
N = 3 => $L_c = 6$



Construction of C: Number inputs and mix them in an arbitrary order.

Lower bound of Benes network

- Number of 2x2 SBs in a Benes network:
  => $2log_2N - 1$ stages and each stage has $N/2$ SBs of size 2x2
  => total number of 2x2 SBs is $Nlog_2N - N/2$, which is close to $\zeta_{pt\text{-}pt}$
  => total number of cross-points $4(Nlog_2N - N/2) \approx 4Nlog_2N$



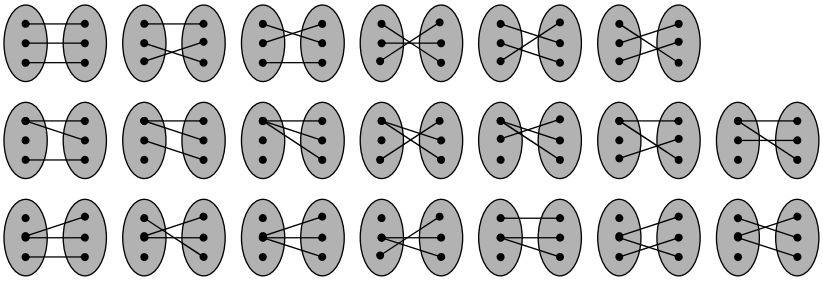
Lower combinatorial bound for multi-point connections

- Let's suppose that any input can be connected to any output, i.e., each element in ***O*** may choose any one of the ***N*** inputs
  => total number of connection patterns ***C*** is $N^N = \exp_2(N \log_2 N)$
  => $\zeta_{mcast} = N \log_2 N$
  => $\zeta_{mcast} - \zeta_{pt\text{-}pt} = 1.44N$
- A fabric architecture that would implement multi-casting and would be close to the lower bound of complexity is not known yet
- It is known that Benes network implements multi-cast if the number of 2x2 SBs is doubled compared to the pt-to-pt case

Visualization of multi-cast mappings

Number of connection patterns in multi-cast switching $L_c = N^N$

N = 3 => L_c = 27



etc. …

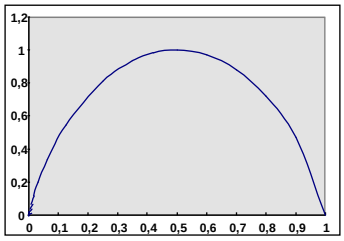
Lower combinatorial bound for concentrator

- A concentrator with ***M*** inputs and ***N*** outputs (*M*>*N*)
- Connection pattern ***C*** defined to be a set of any ***N*** of the ***M*** inputs
- Number of these sets = $\binom{M}{N} = \frac{M!}{N!\,(M-N)!}$
- Sterling's approximation:

$$\zeta_{concentrator} = \log_2 \frac{M!}{N!(M-N)!}$$

$$\approx \log_2 \sqrt{\frac{M}{2\pi N(M-N)}} \left(\frac{M^M}{N^N (M-N)^{M-N}} \right)$$

$$\approx MH(c)$$



Entropy function: $H(c) = -c\log_2 c - (1-c)\log_2(1-c), \quad c = N/M$

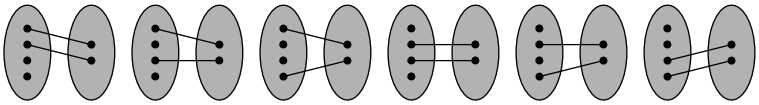
Lower combinatorial bound for concentrator (cont.)

- For given ***C*** => $\zeta_{concentrator} = O(M)$
- This lower bound is smaller by a factor of log*M* than that of point-to-point or multi-point networks
- Although concentrators with linear complexity (linear to number of inputs) can be shown to exist, there are no known practical solutions due to complicated control algorithms
- It can be shown that a strict-sense non-blocking concentrator is as complex as a point-to-point non-blocking concentrator - *M logM*
- => *MlogM*-fabrics are used for concentration

Visualization of concentrator mappings

Number of connection patterns in concentrator $L_c = \binom{M}{N} = \frac{M!}{N!(N-M)!}$

Concentrator MxN = 4x2 => $L_c = 6$



Lower combinatorial bound for super-concentrator

- A super-concentrator with ***M*** inputs, ***N*** outputs and ***K*** elements ($K \leq M,N$)
- Connection pattern ***C*** defined to be a legitimate set of ***C*** = ***(A, B)*** by all ***A*** and ***B*** with ***K*** elements
- Total number of these sets = $\binom{M}{K}\binom{N}{K}$

$$\zeta_{super-con} \approx M\,H\left(\frac{K}{M}\right) + N\,H\left(\frac{K}{N}\right)$$

- Super concentrators more complex than concentrators
- Compact super-concentrator specifies output set ***B*** once the starting position of the compact sequence is specified => there are *N* possible starting positions and hence

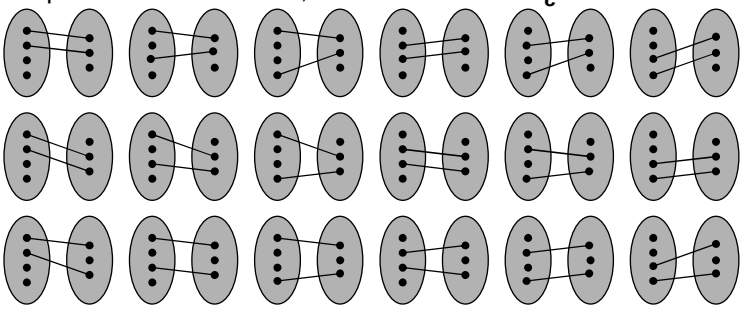
$$\zeta_{super-con} \approx M\,H\left(\frac{K}{M}\right) + \log_2 N$$

Visualization of super-concentrator mappings

Number of connection patterns in super-concentrator

$$L_c = \binom{M}{K}\binom{N}{K} = \frac{M!N!}{K!(M-K)!K!(N-K)!}$$

Super concentrator M=4, N=3 and K=2 => $L_c = 18$



Lower combinatorial bound for copy network

- A copy network with M inputs and N outputs
- Connection requests n_i over all inputs i is equal to N

  => Number of connection patterns C equals to $\binom{M-1+N}{M-1}$

  => $\zeta_{copy} \approx (M-1+N)H\left(\frac{M-1}{M-1+N}\right)$

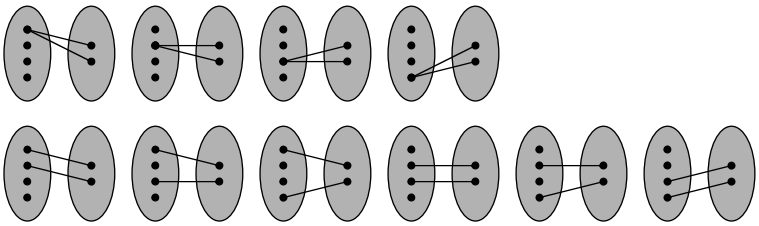
Complexity lower bound is liner in M and N

Visualization of copy mappings

Number of connection patterns in copy network

$$L_c = \binom{M-1+N}{M-1} = \frac{(M-1+N)!}{(M-1)!N!}$$

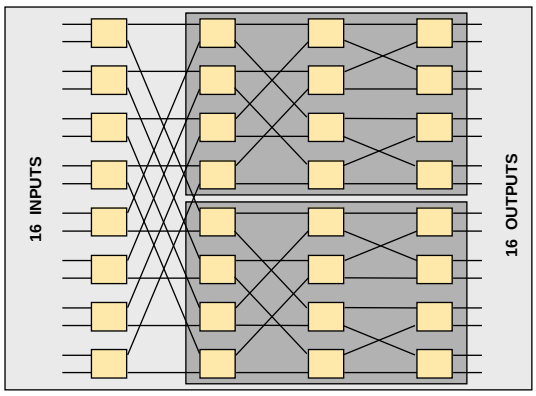
Copy MxN = 4x2 => $L_c = 10$



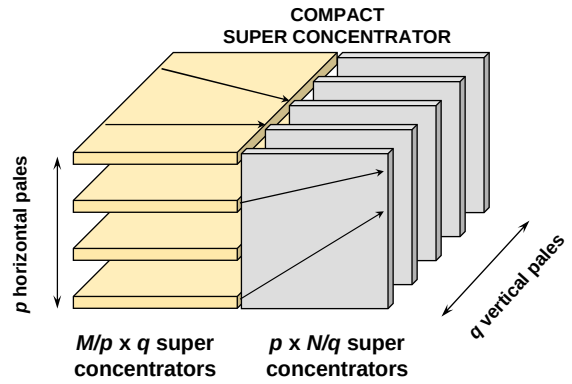
L_c is the number of ways *N* objects can be put into *M* bins.

Compact super-concentrator example

Inverse Banyan network formed by recursive 2-stage factoring using super-concentrators



Two stage factoring



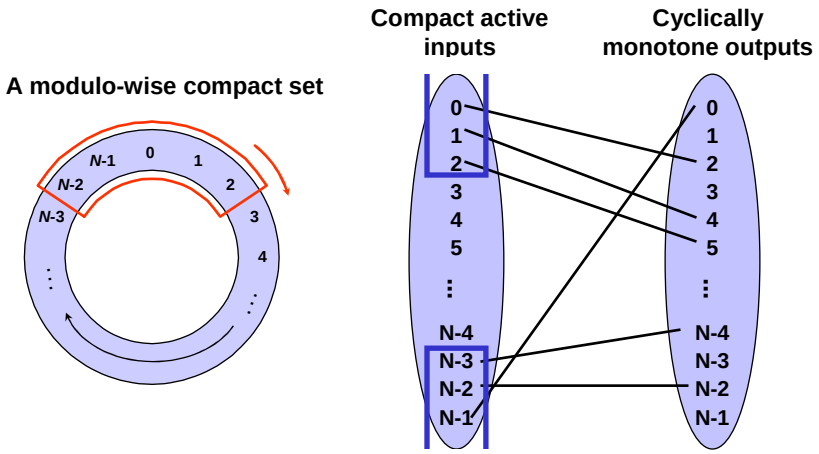
2-stage factoring can be used to construct compact super-concentrators

Distribution network

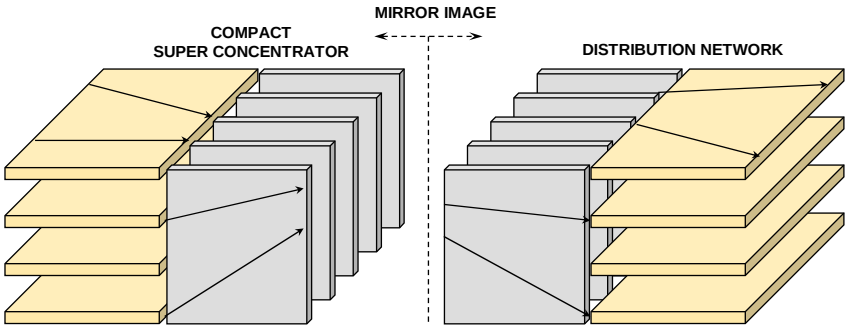
- Mirror image of a compact super-concentrator is called a **distribution network**
- Provided that an input-output connection pattern $\boldsymbol{C}$ = { (i,o_i) } satisfies:
  - **Compactness condition** - active inputs i for the pair in $\boldsymbol{C}$ are compact in modulo fashion
  - **Monotone condition** - outputs o_i to be connected to each active input are strictly increasing in i in modulo fashion

  a 2-stage network can be made a non-blocking one if the connection requests arrive in sorted order - one way to achieve this is to put a sorting network in front of a 2-stage network
- All point-to-point connections satisfying the above two conditions can be connected using the distribution network

Compact and monotone connection pattern

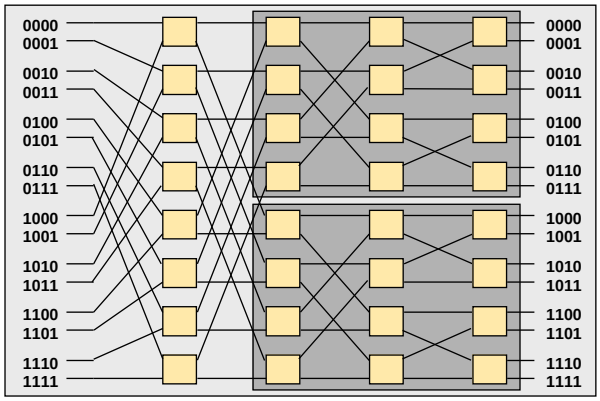


Construction of a distribution network



Example of a distribution network

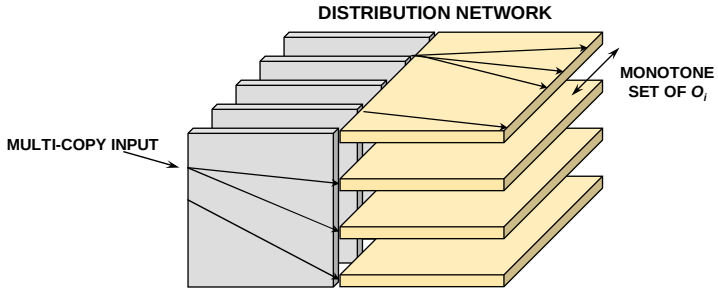
Distribution network based on inverse Banyan network



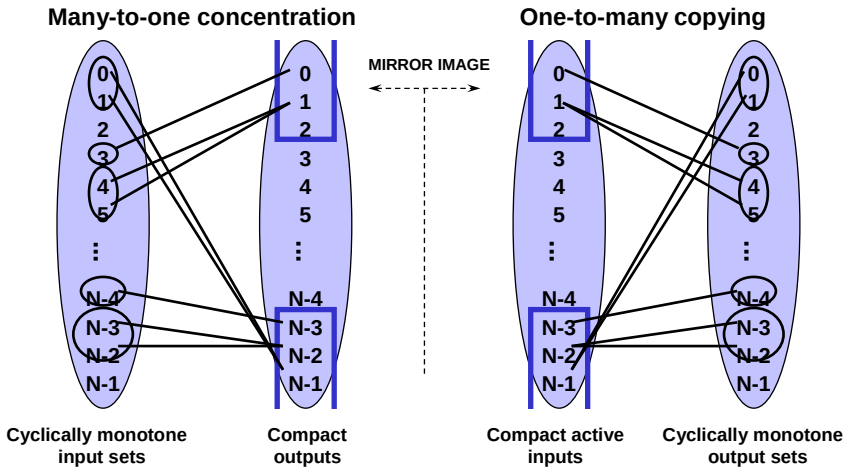
Construction of copy networks

- Distribution network, which allows multiple connections between an input and outputs, is called a copy distribution network
- An input can make extra connections to outputs if the outputs connected remain monotonically increasing with respect to the inputs
  - **Compactness condition** - active inputs ***i*** for the pair in ***C*** are compact in modulo fashion
  - **Monotone condition** - each element in O_i is greater than each element in $O_{i'}$ if ***i*** >***i'*** in modulo fashion
- Inverse of many-to-one concentrator performs the copy functions

Copy distribution network



Compact and monotone connection pattern



Construction of multi-cast networks

- Multi-cast networks can be constructed, e.g. by concatenating a copy network and a point-to-point network
  - 3-stage factorization can be applied to get a point-to-point network
  - resulting network consists of a concentrator, copy distribution network and Benes network
  => number of stages increases
  => total number of 2x2 SBs is $2N\log_2 N$
- There are alternative ways to construct multi-cast networks, but they encounter the above mentioned problems
  - number of stages increases
- Difficult to calculate connections through the fabric
- Complicated fabric control algorithms
- One way to solve the control problem is to use self-routing

Switch fabrics

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

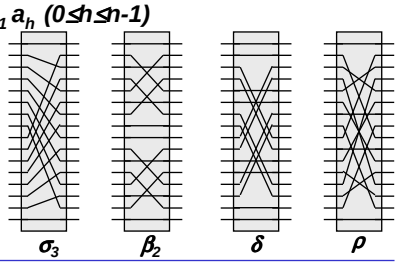
Self-routing

- Self-routing is a popular principle in fast packet switching
- Header of each packet contains all information needed to route a packet through a switch fabric
- One or more paths may exist from an input to an output
- Interconnection network has the **unique path property** - if the sequence of nodes connecting an input to an output is unique for all input-output pairs
- An important class of networks having the unique property is the generic banyan network
  - *N*log*N* complexity
  - only one path connecting each input-output pair

Banyan network

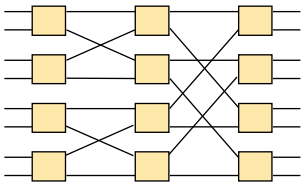
- Different permutations (mappings) have been defined between inlets and outlets of a generic *N*x*N* network that are used as the basic building blocks of self-routing networks
- Examples of commonly used permutations are
  - $\sigma_h(a_{n-1}\ldots a_0) = a_{n-1}\ldots a_{h+1}\, a_{h-1}\ldots a_0\, a_h$ $(0 \le h \le n-1)$
  - $\sigma_h^{-1}(a_{n-1}\ldots a_0) = a_{n-1}\ldots a_{h+1}\, a_0\, a_h \ldots a_1 (0 \le h \le n-1)$
  - $\beta_h(a_{n-1}\ldots a_0) = a_{n-1}\ldots a_{h+1}\, a_0\, a_{h-1}\ldots a_1\, a_h$ $(0 \le h \le n-1)$
  - $j(a_{n-1}\ldots a_0) = a_{n-1}\ldots a_0$
  - $\delta(a_{n-1}\ldots a_0) = a_1\, a_2 \ldots a_{n-1}\, a_0$
  - $\rho(a_{n-1}\ldots a_0) = a_0\, a_1 \ldots a_{n-2}\, a_{n-1}$

where $a = a_{n-1}\ldots a_0$ represents a generic address with base-2 digits a_i $(n=\log_2 N)$

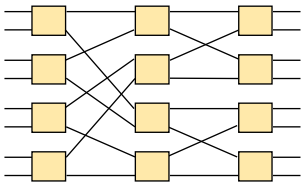


Examples of unique route network

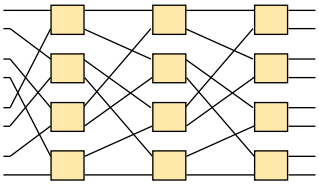
Banyan network



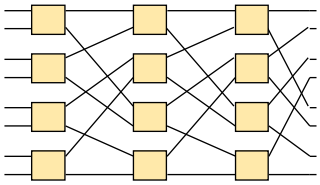
Baseline network



Shuffle exchange (Omega) network

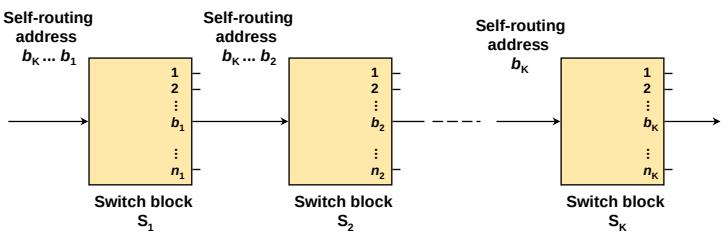


Flip network
(inverse shuffle exchange)



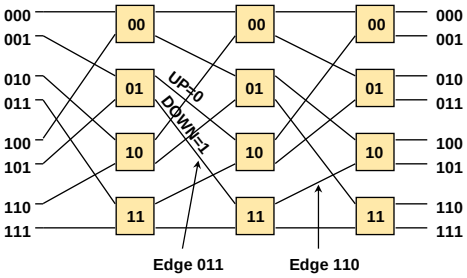
Self-routing principle

- S_1, S_2 , … S_K is a sequence of switch blocks, which have n_1, n_2, … n_K outputs respectively
- Route of a packet uses the b_k-th output of switch block k
  => route given by the sequence b_1 b_2 … b_k … b_K
- At switch block k, packet routed to output b_k and address b_k is removed from the self-routing header



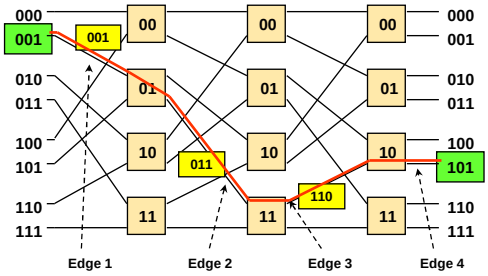
Self-routing principle (cont.)

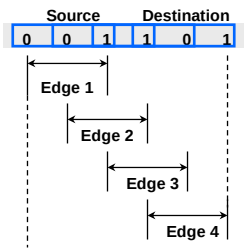
- In a self-routing shuffle exchange (*N*x*N*) network
  - $N = \mathbf{2}^{K}$ inputs and outputs interconnected by $\boldsymbol{K}$ stages of 2^{K-1} nodes
  - nodes numbered in each stage from 0 to 2^{K-1}-1 (binary $\boldsymbol{K}$-1 format)
  - links (= edges) in each stage numbered from 0 to 2^{K}-1 (binary $\boldsymbol{K}$ format)
  - outgoing links numbered by appending "0" (up going links) or "1" (down going links) to the node's number



Self-routing in a shuffle exchange network

- Self-routing shuffle exchange scheme
  - a packet at input $\boldsymbol{a_1a_2}$ … $\boldsymbol{a_K}$ is destined to output $\boldsymbol{b_1b_2}$ … $\boldsymbol{b_K}$
  - $\boldsymbol{b_1b_2}$ … $\boldsymbol{b_K}$ is used as the self-routing address (up link chosen if $\boldsymbol{b_k}$=0 and down link if $\boldsymbol{b_k}$=1)
  - packet visits first node $\boldsymbol{a_2}$…$\boldsymbol{a_K}$ => travels along edge $\boldsymbol{a_2}$…$\boldsymbol{a_K b_1}$ => visits node $\boldsymbol{a_3}$…$\boldsymbol{a_K b_1}$ =>…=> finally after visiting node $\boldsymbol{b_1}$…$\boldsymbol{b_{n-1}}$ arrives at output edge $\boldsymbol{b_1}$…$\boldsymbol{b_n}$





Monotone and compact addresses

- Top-down numbering of nodes and links can be used also for other self-routing networks having the unique property
- If self-routing addresses of packets at the inputs satisfy conditions:
  - addresses are strictly **monotone** in the sense that destination addresses are strictly increasing in top-down manner at the inputs
  - packets are **compact** in the sense that there is no idle input between any two inputs with packets

=> self-routing paths used by these packets do not share any link within the shuffle exchange network

=> no need for buffering at inputs of the internal nodes

Limitations of banyan networks

- Banyan network can realize $\mathbf{exp_2}(\frac{1}{2}N\log_2 N) = (N^N)^{\frac{1}{2}}$ input-output permutations (connection patterns)
- Full connectivity requires $N!$ connection patterns
  => Banyan network is a blocking one
- Combinatorial power of banyan network can be increased significantly by implementing
  - multiple links between nodes
  - duplicated switch
  - appended switch with random routing (shuffle)
  - buffering at intermediate nodes (=> undesirable random delay)

Example of a multi-cast copy network