Packet Switch Architecture

3. Output Queueing Architectures
4. input Queueing Architectures
5. Switching Fabrics
6. Flow and Congestion Control in Sw. Fabrics

Manolis Katevenis
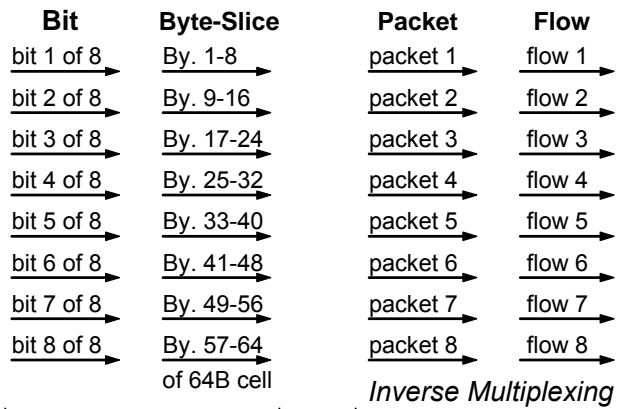
FORTH and Univ. of Crete, Greece

`http://archvlsi.ics.forth.gr/~kateveni/534`

---

5. Switching Fabrics

Table of Contents:

- **5.1 Inverse Multiplexing (Adaptive / Multipath Routing)**
  - byte-sliced switches, recursive definition of the Benes network
  - load distribution & balancing, packet ordering & resequencing
- **5.2 Scalable Non-Blocking Switching Fabrics**
  - banyan, Benes, Clos – $O(N \cdot \log N)$ cost & lower bound
  - fat trees – controlled blocking, locality of traffic
- **5.3 What about Scalable Scheduling?**
  - self-routing fabrics, sorting networks: bad solution
  - fabrics with small internal buffers and flow control: good solution

2

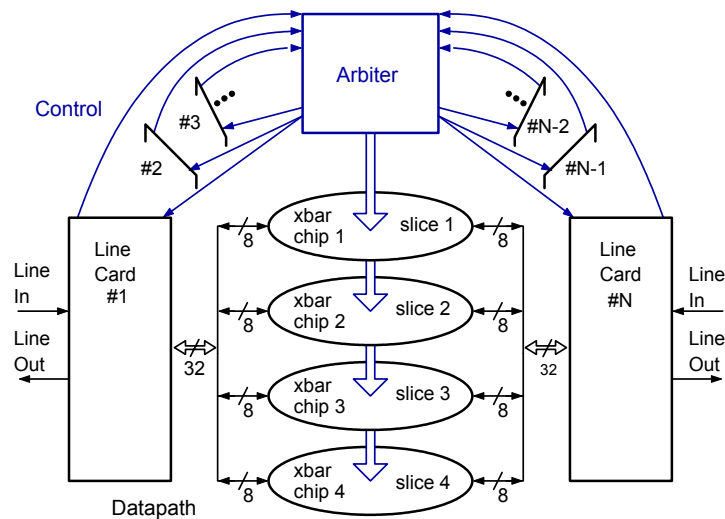## 5.1 Parallelism for High-Thruput: Inverse Multiplexing

Bit	Byte-Slice	Packet	Flow
bit 1 of 8	By. 1-8	packet 1	flow 1
bit 2 of 8	By. 9-16	packet 2	flow 2
bit 3 of 8	By. 17-24	packet 3	flow 3
bit 4 of 8	By. 25-32	packet 4	flow 4
bit 5 of 8	By. 33-40	packet 5	flow 5
bit 6 of 8	By. 41-48	packet 6	flow 6
bit 7 of 8	By. 49-56	packet 7	flow 7
bit 8 of 8	By. 57-64	packet 8	flow 8
	of 64B cell		

Inverse Multiplexing

same handling for all wires   different handling
(same time, same destination)   (diff. times & destinations)

- Parallel wires or network routes for scaling (virtual) "link" throughput up
- Easy: central control, synchronized; Difficult: distributed control, asynch.

5.1 - U.Crete - M. Katevenis - CS-534   3

## 5.1 Byte-Slicing: Tiny Tera & other commercial chips

Arbiter

Control

#3   #N-2

#2   #N-1

Line Card #1

Line In

Line Out

Datapath

xbar chip 1 — slice 1
xbar chip 2 — slice 2
xbar chip 3 — slice 3
xbar chip 4 — slice 4

8   8   32   32

Line Card #N

Line In

Line Out

Mckeown e.a.: "Tiny Tera: a Packet Switch Core", IEEE Micro, Jan.-Feb.'97
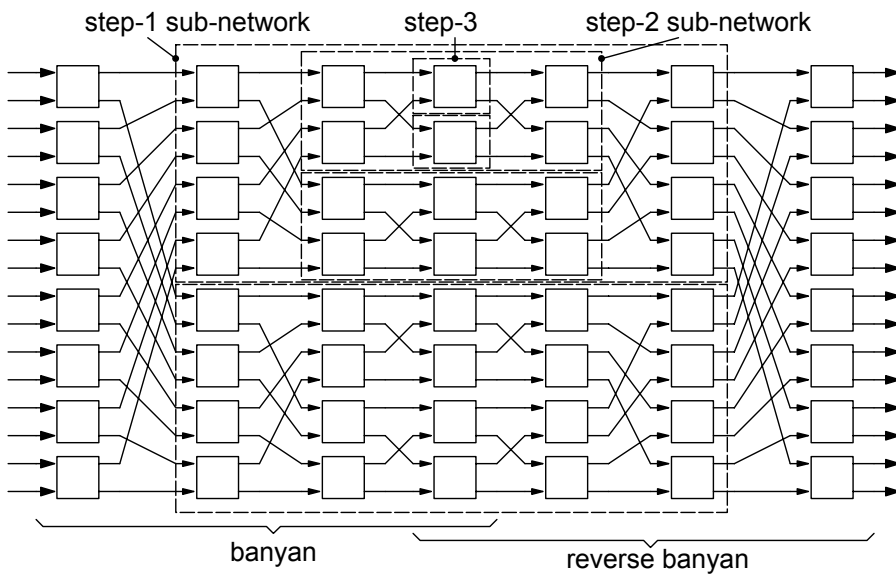
5.1 - U.Crete - M. Katevenis - CS-534   4
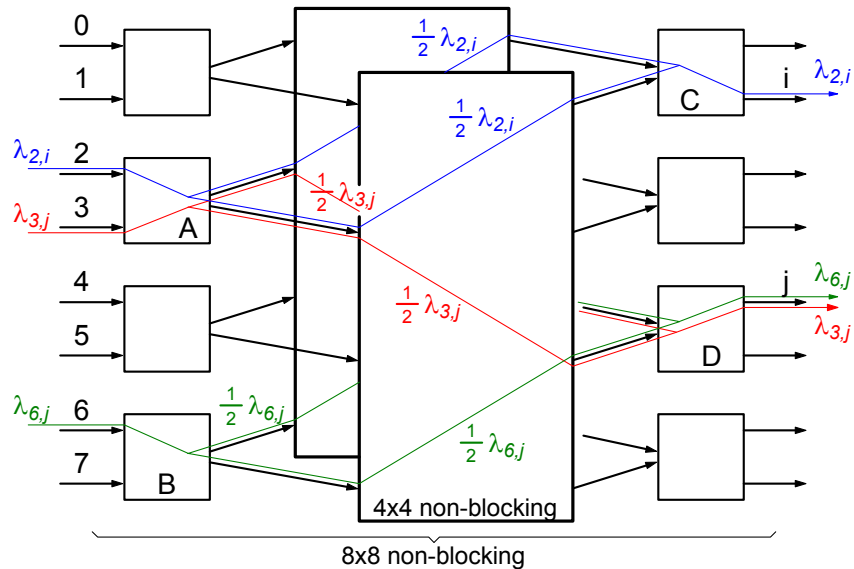
5.2.1 Benes Fabric: Recursive Definition

NxN Benes network: rearrangeably non-blocking
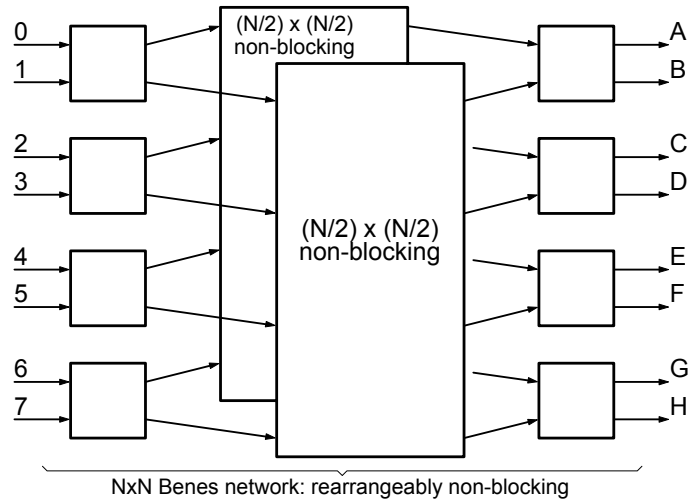
- Goal: reduce switch radix from *N×N* to *(N/2)×(N/2)*: combine ports in pairs
- Port-pairs require links of twice the throughput: use inverse multiplexing
⇒ Use two switches, of half the radix each, in parallel to provide req'd thruput

Full Construction of 16×16 Benes out of 2×2 Switches

Inverse Multiplexing for Non-Blocking Operation

$$\tfrac{1}{2}\lambda_{2,i}$$

$$\tfrac{1}{2}\lambda_{2,i}$$

0
1

C

i $\quad\lambda_{2,i}$

$\lambda_{2,i}$ 2

$\tfrac{1}{2}\lambda_{3,j}$

$\lambda_{3,j}$ 3

A

4
5

$\tfrac{1}{2}\lambda_{3,j}$

D

j $\quad\lambda_{6,j}$

$\lambda_{3,j}$

$\lambda_{6,j}$ 6 $\quad\tfrac{1}{2}\lambda_{6,j}$

7

B

$\tfrac{1}{2}\lambda_{6,j}$

4x4 non-blocking

8x8 non-blocking

5.2 - U.Crete - M. Katevenis - CS-534

7

Per-Flow Inverse Mux'ing for Non-Blocking Operation

- Prove that overall *N×N* network is non-blocking, i.e. *any* feasible external traffic $\Rightarrow$ feasible rates on all internal links
- All traffic entering switch A is feasible, hence of aggregate rate ≤ 1+1 = 2; it is split into two halves $\Rightarrow$ each of rate ≤ 1 $\Rightarrow$ traffic entering each *(N/2)×(N/2)* subnetwork is feasible
- It does <u>not suffice</u> to balance (equalize) the *aggregate* load out of switch A – must equally distribute *individual* (end-to-end) flows – *per-flow* inverse multiplexing

  $\Rightarrow$ each of $\lambda_{2,i}$; $\lambda_{3,j}$; $\lambda_{6,j}$ is individually split in two equal halves

  $\Rightarrow$ the sum of $\lambda_{3,j}+\lambda_{6,j}$ is also split in two equal halves

- All traffic exiting switch D is feasible, hence of aggregate rate ≤ 1+1 = 2; it enters D in two equal halves $\Rightarrow$ each of rate ≤ 1 $\Rightarrow$ traffic exiting each *(N/2)×(N/2)* subnetwork is also feasible

5.2 - U.Crete - M. Katevenis - CS-534

8

---

5. Switching Fabrics

4

Conceptual View of 8x8 Benes: Virtual Parallel Links using Inverse Multiplexing

multiplex, then inverse-multiplex   merge (and resequence), then demultiplex

Methods to implement (per-flow) Inverse Multiplexing

- *Per-Flow Round-Robin*, at packet granularity
  - for each flow, circularly and per-packet alternate among routes
  - requires maintaining per-flow state
  - danger of synchronized RR pointers: pck bursts to same route
  - alternative: arbitrary route selection, provided the (per-flow) imbalance counter has not exceeded upper bound value

5.2 - U.Crete - M. Katevenis - CS-534   9

Methods to implement (per-flow) inverse multiplexing (continued)

- *Adaptive Routing*, at packet granularity – usu. Indisciminate
  - chose the route with least-occupied buffer (max. credits)
  - + does not maintain or use per-flow state
  - − per-flow load balancing only "after-the-fact", when buffers fill up
- *Randomized* Route Selection, at packet granularity
  - + does not require maintaining per-flow state
  - − load balancing is approximate, and long-term
- **Packet Resequencing** (when needed): major cost of inv.mux'ng
  - Chiussi, Khotimsky, Krishnan: IEEE GLOBECOM'98
- Hashed Route Selection *at entire Flow Granularity*
  - route selection based on hash function of flow ID
  - + all packets of given flow through same route ⇒ *in-order delivery*
  - − poor load balancing when small number of flows

5.2 - U.Crete - M. Katevenis - CS-534   10

---

5. Switching Fabrics   5

5.2.2 The Banyan (Butterfly) Network

- Single route from given input to given output
- Each input is the root of a tree leading to all outputs
- Trees share nodes
- (Similarly, outputs are roots of trees feeding each from all inputs)
- for $N{\times}N$ network made of $2{\times}2$ sw.:
- $\log_2 N$ stages, of
- $N/2$ sw. per stage
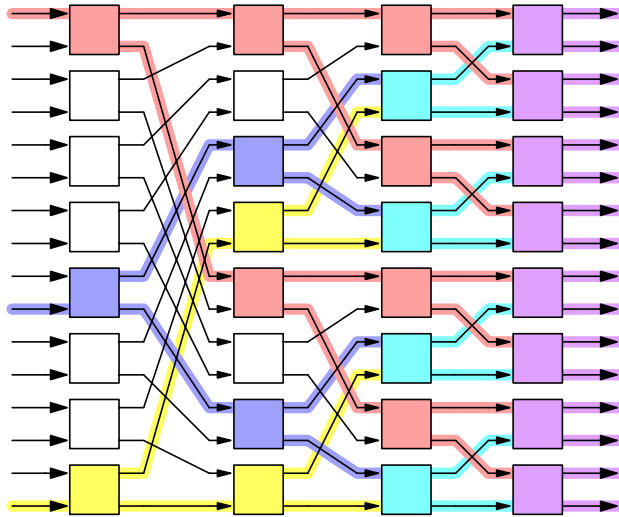
5.2 - U.Crete - M. Katevenis - CS-534   11

The banyan network is internally blocking

- Consider circuits: each $\lambda_{i,j}$ is either 1 or 0: single connection per port – "telephony" style
- There are $N!$ such circuit connection patterns for a $N{\times}N$ network – each is a permutation of the numbers $(1, 2, …, N)$

internal blocking

- Any network containing $(N/2){\cdot}\log_2 N$ or less $2{\times}2$ switches (like the banyan does) has to be internally blocking, because it can only be placed into less than $N!$ states, hence cannot route all $N!$ existing sets of con. req's
- Each $2{\times}2$ switch can be placed in 2 different states; a network containing $(N/2){\cdot}\log_2 N$ such switches can be placed into $2^{(N/2){\cdot}\log N} = N^{(N/2)}$ different states; $N^{(N/2)} = N \cdot (N/2)^{(N/2)-1} \cdot 2^{(N/2)-1} < N \cdot [(N-1) \cdot … \cdot (N/2+1)] \cdot [(N/2) \cdot … \cdot 2] = N! \Rightarrow$ not enough states
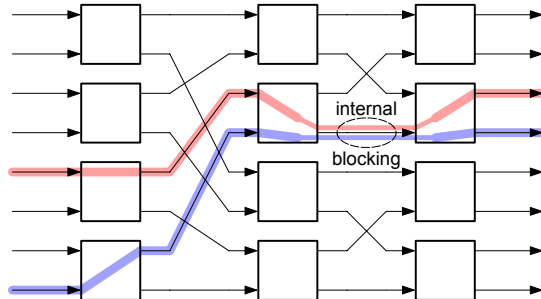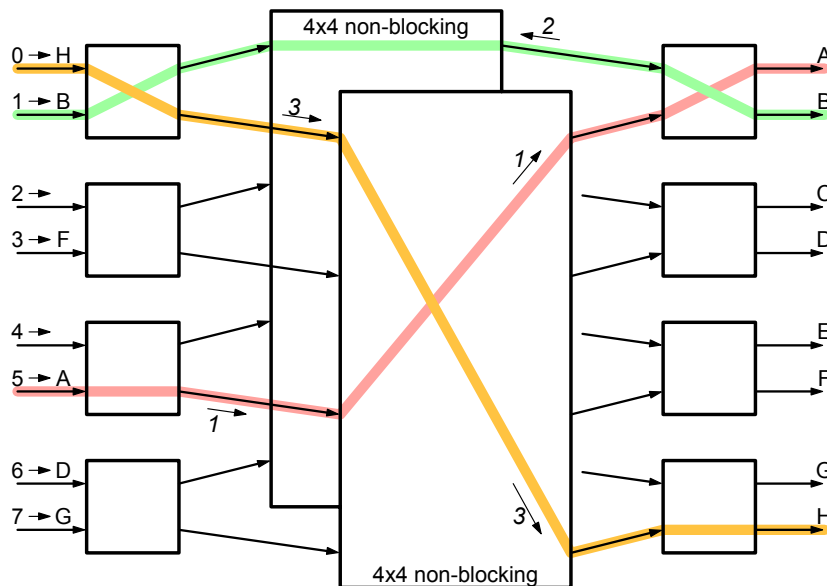
5.2 - U.Crete - M. Katevenis - CS-534   12

Benes Net under Telephony-Ckt Connection Requests

4x4 non-blocking

0 → H
1 → B
2 →
3 → F
4 →
5 → A
6 → D
7 → G

A
B
C
D
E
F
G
H

4x4 non-blocking
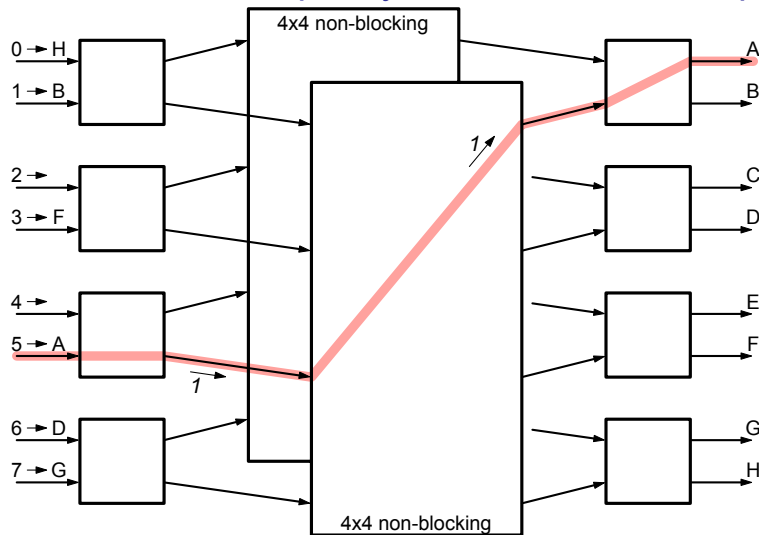
• Circuit Connections: Start from an input, use one of the subnets

13

4x4 non-blocking

0 → H
1 → B
2 →
3 → F
4 →
5 → A
6 → D
7 → G

A
B
C
D
E
F
G
H

4x4 non-blocking

• Continue from the brother port of the output, then the brother of the input

14

5. Switching Fabrics   7

4x4 non-blocking

0 → H
1 → B
2 →
3 → F
4 →
5 → A
6 → D
7 → G

A
B
C
D
E
F
G
H

4x4 non-blocking

- Keep "threading" output and input switches, till closing or no-connection

5.2 - U.Crete - M. Katevenis - CS-534

15

4x4 non-blocking

0 → H
1 → B
2 →
3 → F
4 →
5 → A
6 → D
7 → G

(a)

A
B
C
D
E
F
G
H

4x4 non-blocking

- Start a new "thread" (a) from an unconnected input, till completing all conn.

5.2 - U.Crete - M. Katevenis - CS-534

16
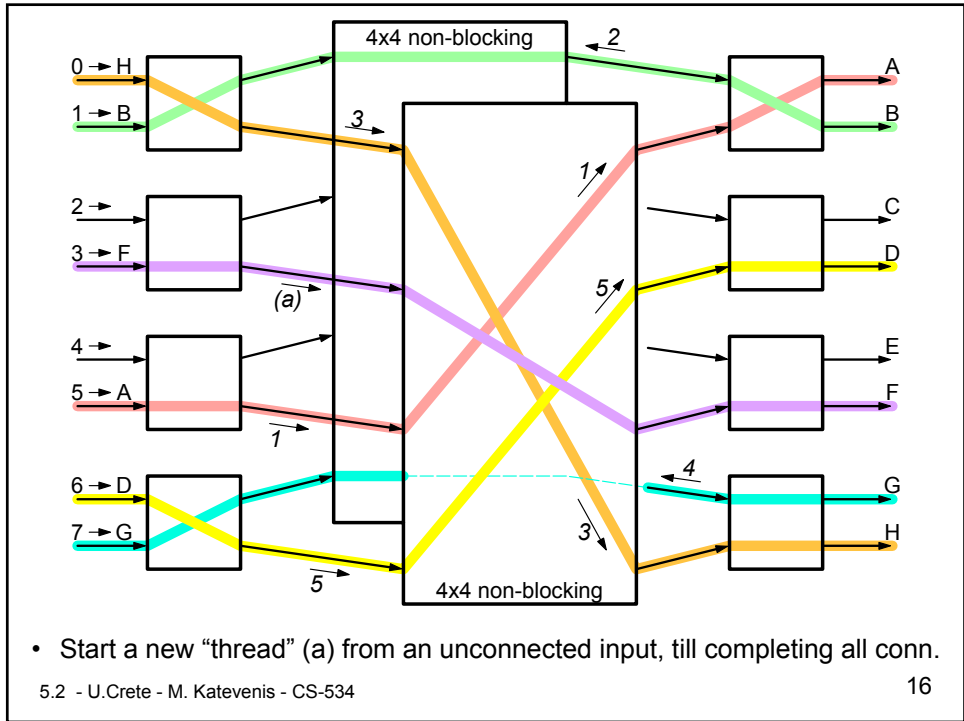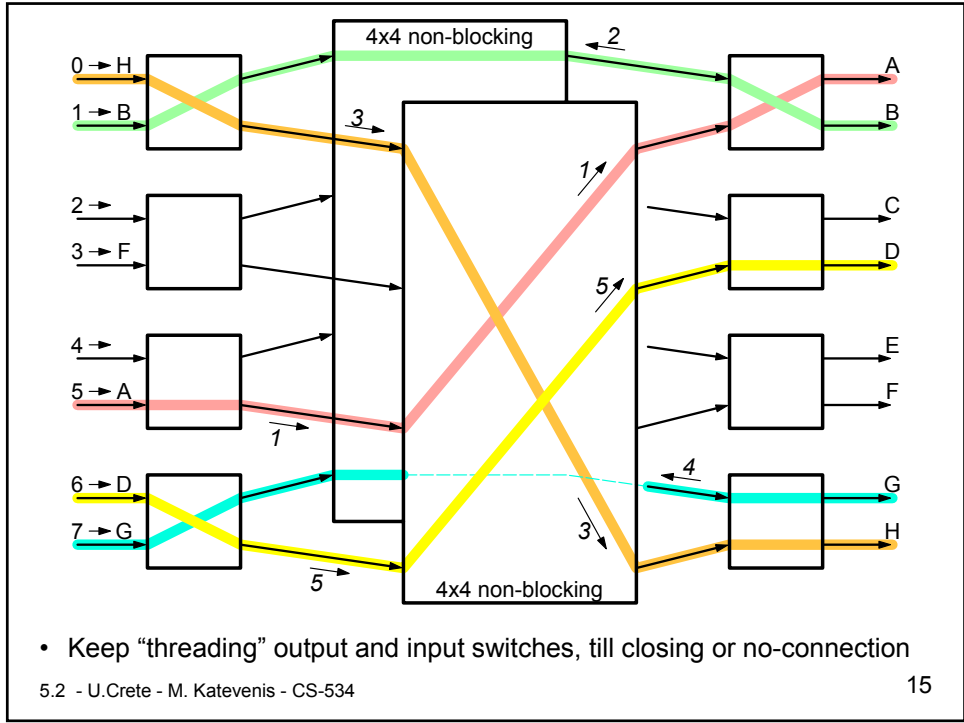
(A) Thread termination on input side (1 of 2)
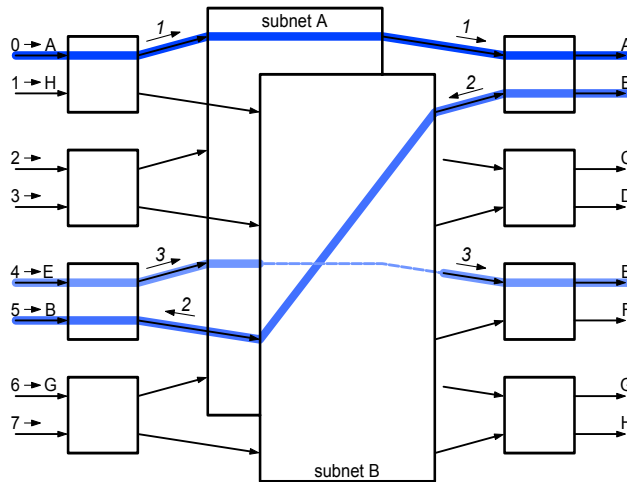

- Threads always start on the input side
- If a thread terminates on the input side:
  - all touched output switches are completely connected
  - concerning touched input switches:
    (1) if thread closes, all are complete,
    …

17

(A) Thread termination on input side (2 of 2)
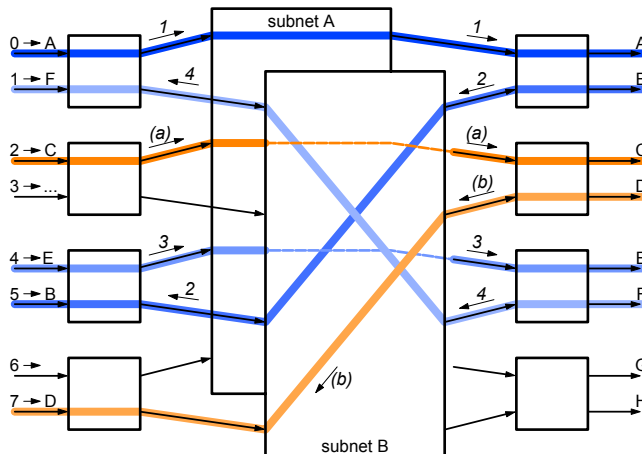
- Threads always start on the input side
- If a thread terminates on the input side:
  - all touched output switches are completely connected
  - concerning touched input switches:
    (1) if thread closes (4), all are complete,
    (2) if thread terminates on half-used input (b): all touched input switches are complete, except the first one, which is half-covered by this thread

18

5. Switching Fabrics
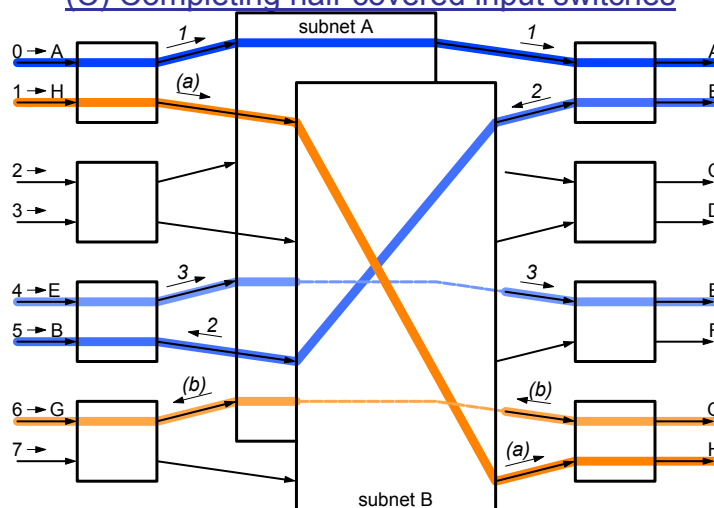
(B) Thread termination on output side

- Threads always start on the input side
- If a thread terminates on the output side:
  - all touched output switches are completely connected
  - the first touched input switch is half-covered
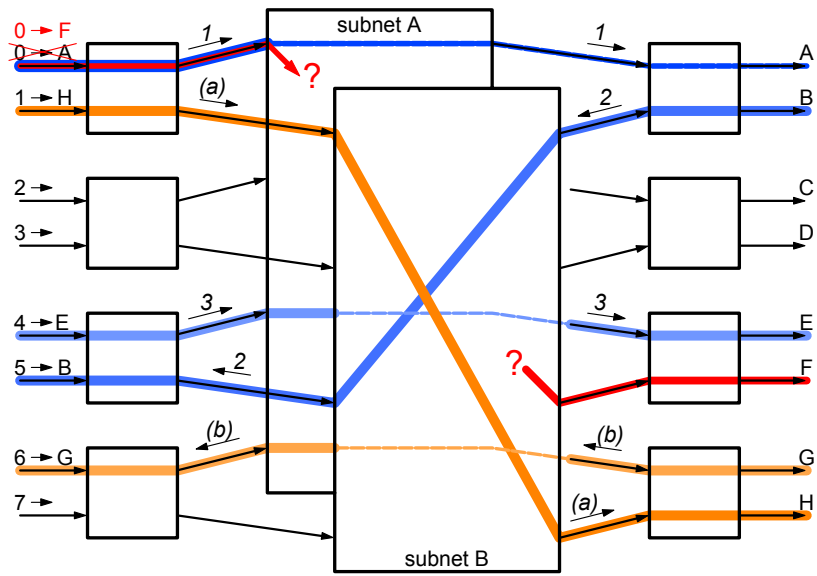
(C) Completing half-covered input switches

- New threads always start from a half-covered input switch, if there is one
  ⇒ all threads cover all out-sw's they touch, in-sw's are covered in sequence

Benes Fabric: *Rearrangeably* Non-Blocking

21

## Which is the lowest-cost non-blocking fabric?

- *N×N* Benes network, made of *2×2* switches:
  - *2·(log$_2$N)−1* stages (2 banyans back-to-back, 1 shared stage)
  - *N/2* switches per stage $\Rightarrow$ total switches = *N·(log$_2$N)−N/2*
  - number of states that the Benes network can be in = $2^{\text{#switches}}$ = $2^{N·(\log N)−N/2} = (2^{\log N})^N / 2^{N/2} = N^N / 2^{N/2} = [N·…·N] ·$ $[(N/2)·…·(N/2)] > N·(N-1)·…·2·1 = N!$ $\Rightarrow$ Benes has more states than the minimum required for a net to be non-blocking
  - Benes was seen to be non-blocking: *(i)* circuits and the "threading" algorithm, *(ii)* packets and inverse multiplexing
  - *"rearrangeably"* non-blocking: in a partially connected network, making a new connection may require re-routing existing ones
- Impossible for any network with about half the switches of the Benes (e.g. banyan) to be non-blocking (# of states)

$\Rightarrow$ Benes is probably the lowest-cost *practical* non-blocking fabric

22

5.2.3 Clos Networks (generalization of Benes nets)

IN inputs per switch

IN=3

IN=3

IN=3

OUT outputs per switch

OUT=3

OUT=3

OUT=3
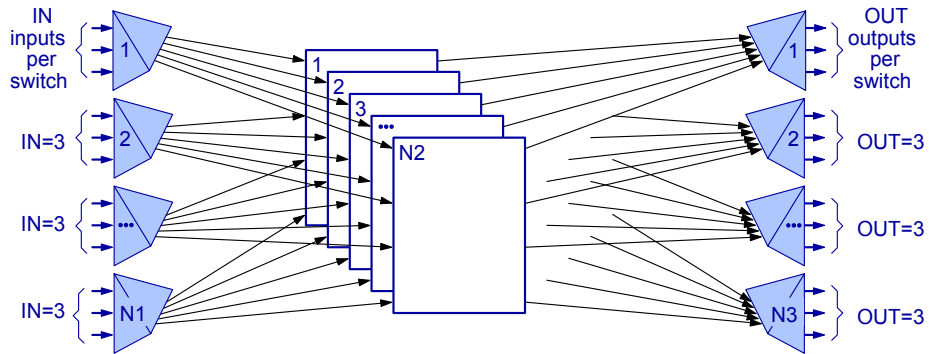
5-parameter Network: (IN, N1, N2, N3, OUT)
this example: the (3, 4, 5, 4, 3) Clos Network
usually: IN = OUT, and N1 = N3

5.2 - U.Crete - M. Katevenis - CS-534

23

Clos Networks

already connected elsewhere
IN-1

A

IN-1

Connect this last free input on A with the last free output on B

already connected elsewhere
OUT-1

B

OUT-1

• Strictly non-blocking
  if and only if N2 $\geq$ IN+OUT-1
• Rearrangeably non-blocking
  if N2 ≥ max{IN, OUT}

N2 switches >= IN +OUT - 1

5.2 - U.Crete - M. Katevenis - CS-534

24

5. Switching Fabrics

12

5.2.4 Fat Trees: customizable local versus global traffic
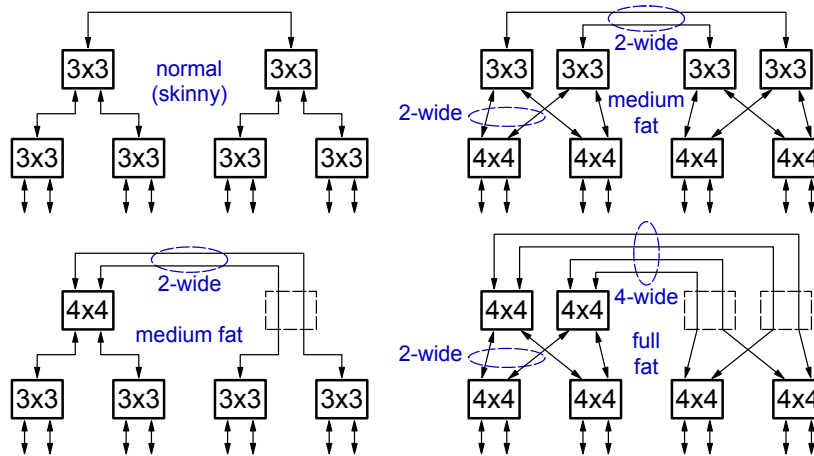
- Customizable percent fat – configurable amounts of internal blocking
- Bidirectional links, like most practical interconnects
- Skinny trees support local traffic – Full-fat tree is like folded Benes

5.2 - U.Crete - M. Katevenis - CS-534   25

Switch Radix, Hop Count, Network Diameter

- Most of our examples used unidirectional links – fig. *(a)*
  - "indirect" nets have ports at edges.
- Most practical interconnects use bidirectional links – fig. *(b)*
  - "direct" nets provide external ports on all switches.
- If some destinations are reachable at reduced hop count (P2 in *(b)*), that is at the expense of the total number of destinations reachable at a given hop count – or larger network diameter.
- Energy consumption to cross the net critically depends on the number of chip-to-chip hops, because chip power is dominated by I/O pin driver consum.

5.2 - U.Crete - M. Katevenis - CS-534   26

---

5. Switching Fabrics   13
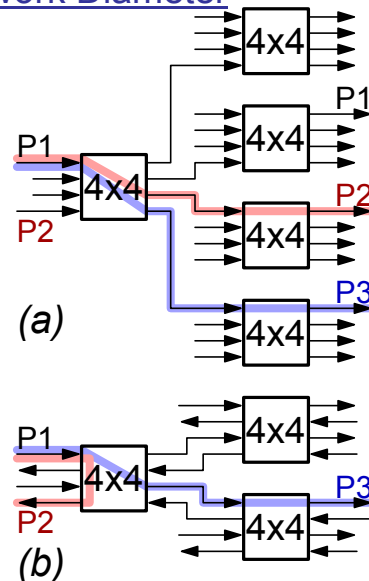
5.3 Towards Scalable Switches

- Buffer throughput limitation $\Rightarrow$ use input queueing or CIOQ
- Input queued crossbar scalability limited primarily by:
  - quadratic cost growth rate, $O(N^2)$, of crossbar
  - scheduler complexity & efficiency, i.e. solving the output contention (congestion management) problem
- To solve the crossbar cost $\Rightarrow$ use switching fabrics
- To solve the scheduler / contention / congestion problem:
  - (sorting / self-routing networks – bad solution)
  - Switching Fabrics with Small Internal Buffers, large input VOQ's, and Internal Backpressure (Flow Control)

5.3 - U.Crete - M. Katevenis - CS-534

27

[intentionally left blank]

5.3 - U.Crete - M. Katevenis - CS-534

28

Central Scheduler is Impractical for large N

Solution 1: Sorting Networks w. Distributed Control (see ch.5 for details)

- all incoming packets allowed in — no central scheduling
- conflicting packets appropriately steered — distributed control

$(N+k) \times (N+k)$ Sorting Network

$(N) \times (sN)$ Sorting Network

N inputs

K Conflicting packets recirculated

N outputs
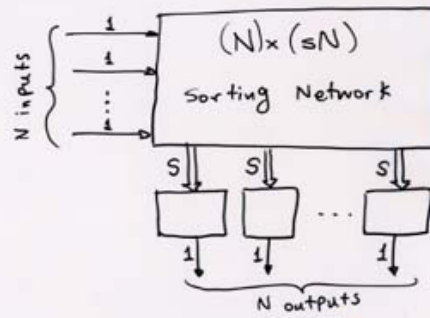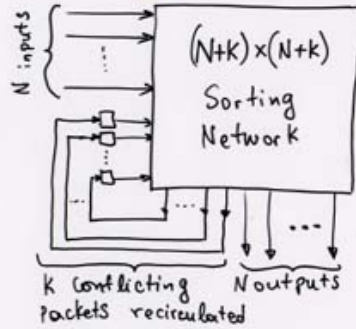
- uses communication paths as buffer memory ... too expensive

- Knock-Out Style but different Sw. Fabric

- Sorting Networks are quite large... not too practical

Central Scheduler is Impractical for large N

Solution 2: Switching Fabrics with Internal Buffering & Backpressure

Switching Fabric Distributed Scheduling

req-grant backpressure

req-grant backpressure
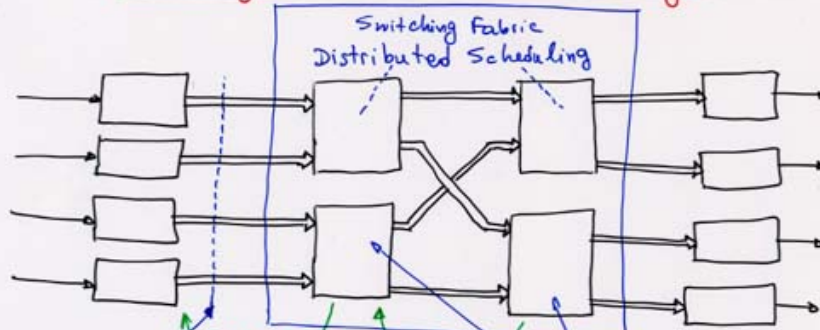
the traffic here may have packets that are short-term-conflicting in the switching fabric, but are long-term-non-conflicting in the fabric

handled by these

small internal buffers

owing to backpressure and distributed scheduling

5. Switching Fabrics   15