

# Packet Routing in hypercubic networks

From Leighton Chapter 3.4

# Routing models and Definitions

- *On Line*. Based on local control and information carried in the packets.
- *Store and forward* (circuit switching)
- *static routing* (dynamic)
- *one to one* (one to many, many to one)

# Greedy Routing and worst case instances

- On the Butterfly there is a unique path from  $\langle u, 0 \rangle$  to any  $\langle v, \log N \rangle$  known as the *greedy path*.
- One packet needs  $\log N$  steps.
- What about  $N$  parallel requests for routing?

# Greedy Routing and worst case instances

- Problem instance is coded as a permutation  $\pi : [1, N] \rightarrow [1, N]$
- We want to send packets from level 0 to level  $\log N$ .  
 $\langle u, 0 \rangle \rightarrow \langle \pi(u), \log N \rangle$
- Bit reversal permutation :  
 $\pi(u_1, u_2, \dots, u_{\log N}) = u_{\log N}, u_{\log N-1}, \dots, u_1$
- $\sqrt{N}/2$  packets will use edge  $\langle 0 \dots 0, (\log N - 1)/2 \rangle$ ,  
 $\langle 0 \dots 0, (\log N - 1)/2 \rangle$

# Greedy Routing path of bit reversal

$$\langle u_1 \cdots u_{\log N - 1/2} 00 \cdots 0, 0 \rangle \rightarrow \langle 0u_2 \cdots u_{\log N - 1/2} 00 \cdots 0, 1 \rangle$$

$\rightarrow \dots$

$$\rightarrow \langle 0 \cdots 0u_{\log N - 1/2} 00 \cdots 0, \frac{\log N - 3}{2} \rangle$$

$$\rightarrow \langle 0 \cdots 000 \cdots 0, \frac{\log N - 1}{2} \rangle$$

$$\rightarrow \langle 0 \cdots 000 \cdots 0, \frac{\log N + 1}{2} \rangle$$

$$\rightarrow \langle 0 \cdots 00u_{\log N - 1/2} 0 \cdots 0, \frac{\log N + 3}{2} \rangle$$

$\rightarrow \dots$

$$\rightarrow \langle 0 \cdots 00u_{\log N - 1/2} \cdots u_1, \log N \rangle$$

# Greedy Routing and worst case instances

- bit reversal permutation congestion:

$$2^{\frac{\log N - 1}{2}} = \sqrt{N/2} = \Theta(\sqrt{N})$$

- Total steps needed to route bit reversal  $\sqrt{N/2} + \log N - 1 = \Theta(\sqrt{N})$
- Another “natural” problem with  $\Theta(\sqrt{N})$  steps running time is the *transpose permutation*

$$\pi(u_1 \cdots u_{\frac{\log N}{2}} u_{\frac{\log N}{2}+1} \cdots u_{\log N}) = u_{\frac{\log N}{2}+1} \cdots u_{\log N} u_1 \cdots u_{\frac{\log N}{2}}$$

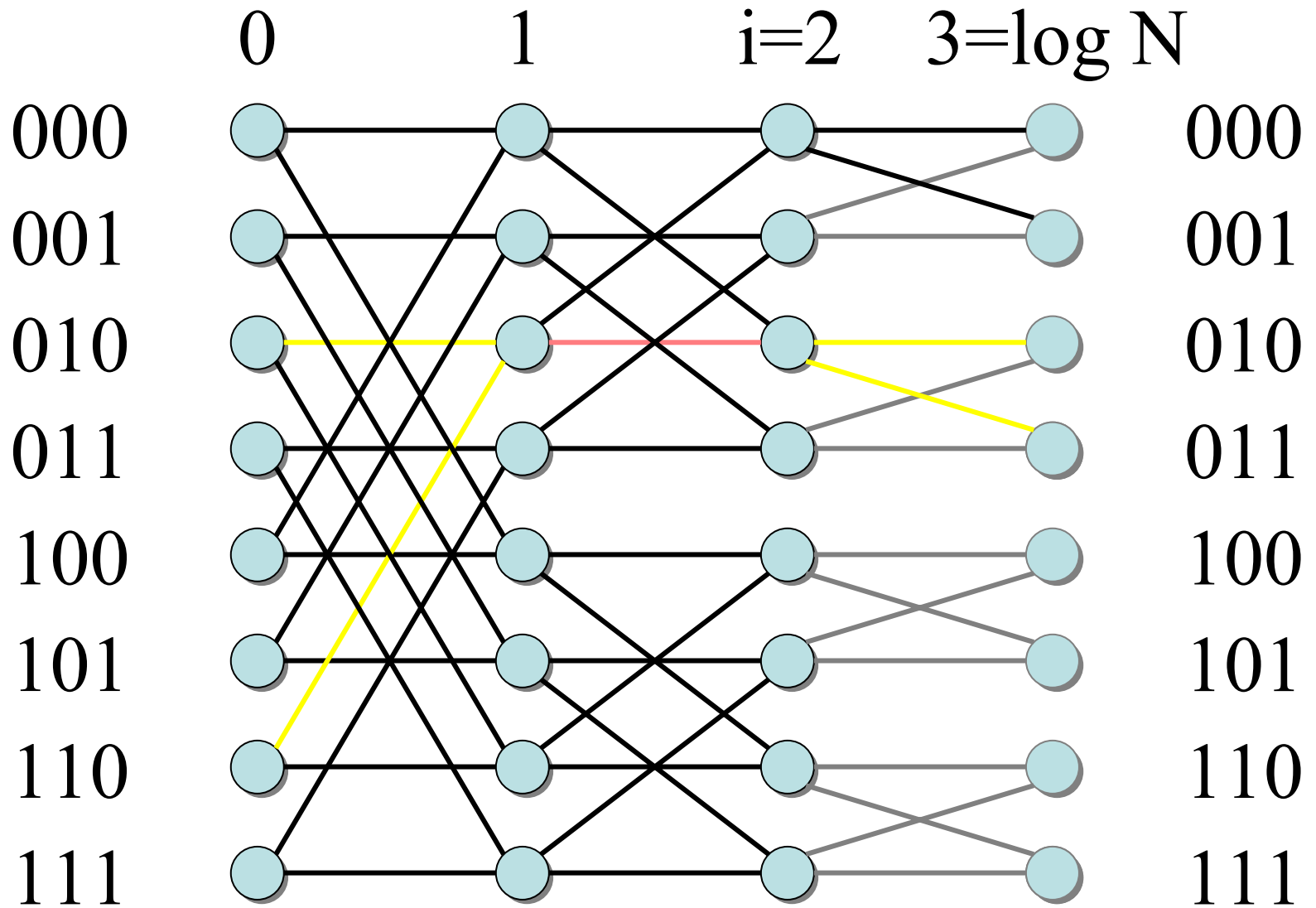
# Theorem 3.22

Any routing problem on a  $\log N$  dimensional butterfly for which at most one packet starts at each level 0 node and at most one is destined for each level  $\log N$  node will route all the packets in  $O(\sqrt{N})$  steps!

**Proof:** Consider an edge  $e$  at the  $i$ -th level. Let  $n_i$  be the # greedy paths passing through  $e$ .

$n_i < \min(2^{i-1}, 2^{\log N - i})$  Reason: How many nodes from level 0 ( $\log N$ ) can (be) reach(ed) from  $e$ ?

# Reachable nodes in Butterfly Network





- **Proof:** Consider an edge  $e$  at the  $i$ -th level. Let  $n_i$  be the # greedy paths passing through  $e$ .
- Total delay is sum of  $n_i$

$$\begin{aligned}
\sum_{i=1}^{\log N} (n_i - 1) &\leq \sum_{i=1}^{\frac{\log N+1}{2}} 2^{i-1} + \sum_{i=\frac{\log N+1}{2}}^{\log N} 2^{\log N-i} - \log N \\
&= 2^{\frac{\log N+1}{2}} + 2^{\frac{\log N-1}{2}} - \log N - 2 \\
&= \frac{3\sqrt{N}}{\sqrt{2}} - \log N - 2
\end{aligned}$$

# Why study end to end greedy routing?

Used often in practice.

If every node in the wrapped butterfly has a packet to send, then we can

1. Route each packet to the correct row (using the same direction).
2. Route the packet to its column.

The  $N \log N$  packets need  $\Theta(N \log N)$  time steps (worst case).

# Why study end to end greedy routing?

## II

Algorithm for arbitrary routing (not e2e) problem:

1. Route each packet to level 0 in its row.
2. Route the packet to the  $\log N$  level node in its destination row. (e2e)
3. Route the packet to its correct destination.

The “computationally” hard part is e2e, which dominates the  $O(\log N)$  of the other parts.

# Oblivious routing

- An algorithm for routing is *oblivious* if the path traveled by each packet does **only** depend on origin and destination.
- Greedy routing is oblivious.
- In a  $N$  node  $d$ -degree network there is a routing permutation problem for which an oblivious routing strategy will need  $\Omega(\sqrt{N}/d)$  steps.
- For the butterfly worst case is  $\Omega(\sqrt{N})$ . Compare this to  $O(\log N)$ . For the hypercube worst case bound is  $\Omega(\sqrt{N}/\log N)$

# Oblivious routing heroes



Krizanc

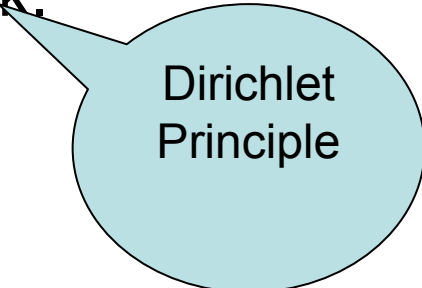
Tsantilas

# Proof

The oblivious algorithm defines a path  $P_{u,v}$  based on the starting  $u$  and ending  $v$  nodes. For any node  $u$  there are  $N-1$   $P_{u,v}$  paths. Let  $S_k(v)$  denote the set of edges that have more than  $k$  paths ending in  $v$  using them. Let  $S_k^*(v)$  be the nodes incident to an edge in  $S_k(v)$ .

Then:  $|S_k^*(v)| \leq 2 |S_k(v)|$ , because every edge is incident to two nodes.

Further: If  $k \leq (n-1)/d$  then  $|S_k^*(v)| \leq 2$  because  $k$  of the  $N-1$  paths ending in  $v$  must follow the same link.



Dirichlet  
Principle

For  $k \leq (n-1)/d$   $|V - S_k^*(v)| \leq (k-1)(d-1) |S_k^*(v)|$

Consider node  $u$  not in  $S_k^*(v)$  and his path  $P_{u,v}$ . The path must eventually enter  $S_k^*(v)$ . Consider the entering node  $(w, w')$   $w \in S_k^*(v)$   $w' \notin S_k^*(v)$ .

$(w, w') \notin S_k(v)$  thus there are at most  $k-1$  nodes  $t$ , for which  $P_{t,v}$  enters  $S_k^*(v)$ . Additionally for each of the  $|S_k^*(v)|$  choices for  $w'$  there are at most  $d-1$  choices for  $w$ . Hence there are at most  $(k-1)(d-1) |S_k^*(v)|$  nodes  $t$  for which  $P_{t,v}$  enters  $S_k^*(v)$  from outside.

$$\begin{aligned}
N &= |V - S_k^*(v)| + |S_k^*(v)| \\
&\leq (k-1)(d-1)|S_k^*(v)| + |S_k^*(v)| \\
&\leq 2[1 + (k-1)(d-1)]|S_k(v)| \\
&\leq 2kd|S_k(v)|
\end{aligned}$$

Ergo for  $k \leq \frac{N-1}{d}$

$$|S_k(v)| \geq \frac{N}{2kd} \text{ and for } k = \frac{\sqrt{N}}{d}$$

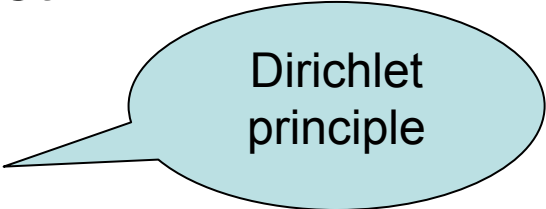
$$\sum_{v \in V} |S_k(v)| \geq \frac{N^2}{2kd} = \frac{N^{3/2}}{2}$$



Up until now we have bounded the sum of the cardinality of the  $k$ -congested edge sets by an expression of  $N$ .

There are at most  $N d/2$  edges in  $G$ . So there is an edge such  $e$  that  $e \in S_k(v)$  for at least

$$\frac{N^{3/2} / 2}{Nd / 2} = \frac{\sqrt{N}}{d} = k \quad \text{nodes}$$



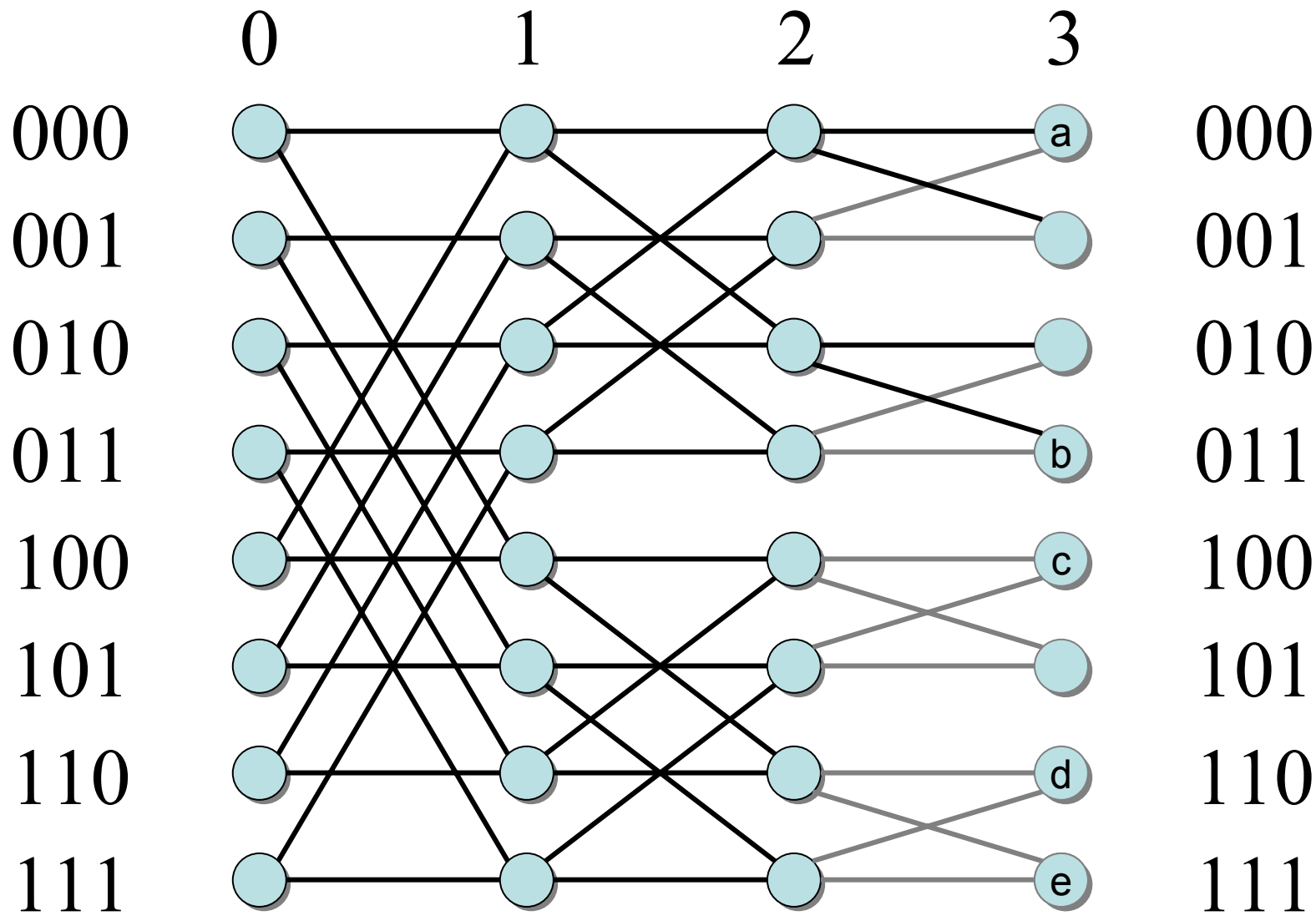
Dirichlet  
principle

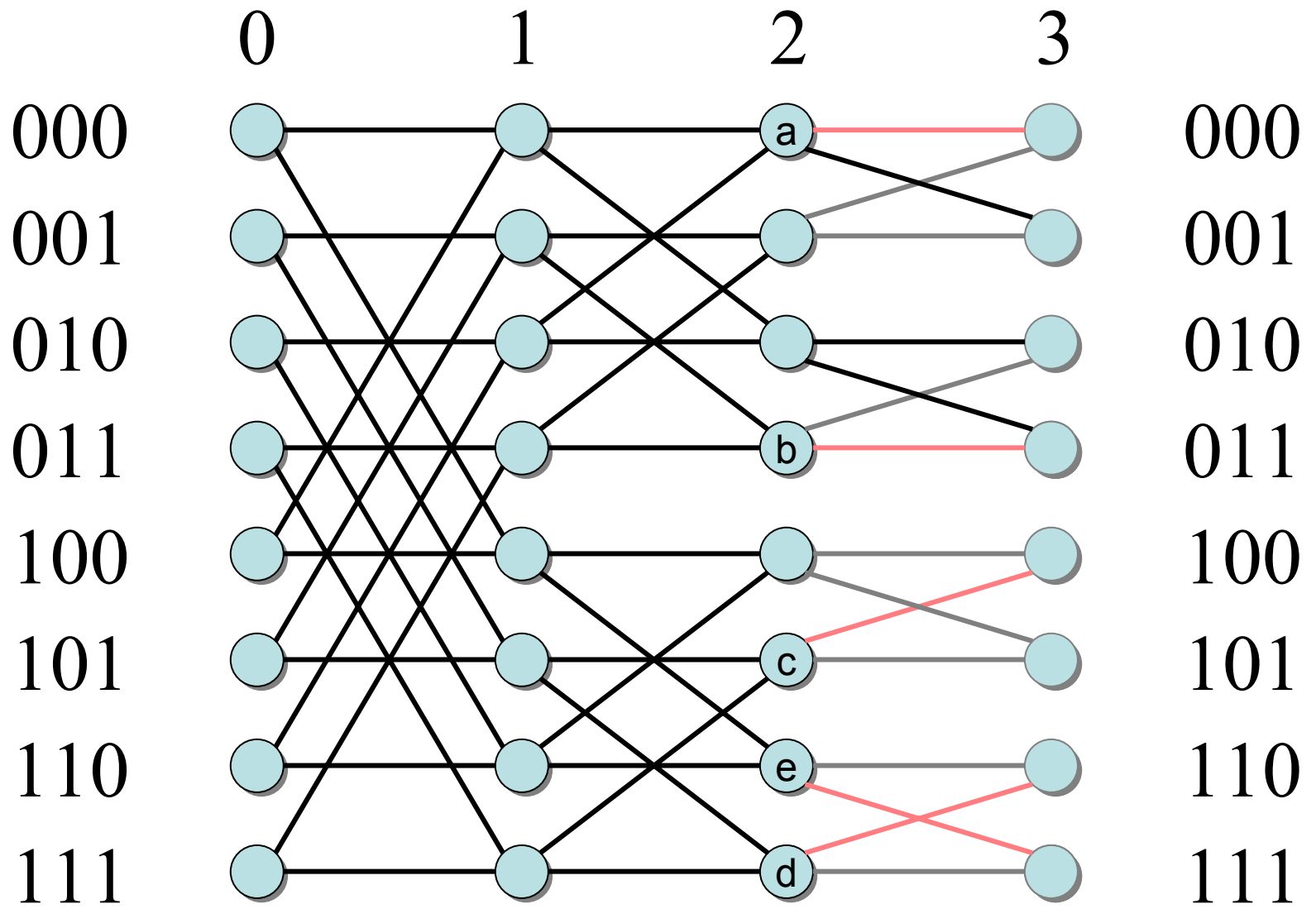
Choose now for all the destination of the paths going through  $e$  a source not chosen by a previous destination results in a partial permutation with overlap bounded as above.

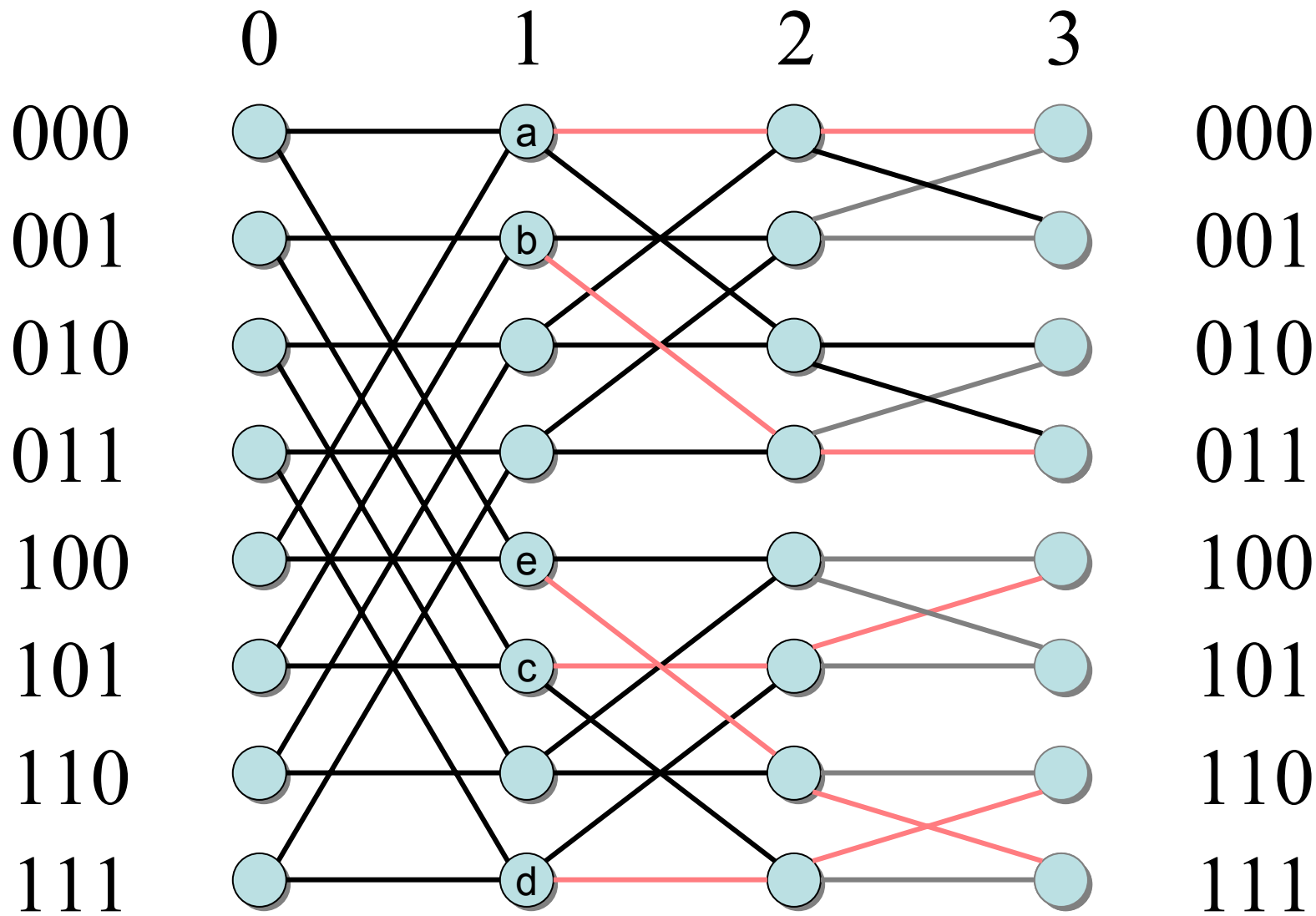
# Packing, Spreading, Monotone Routing

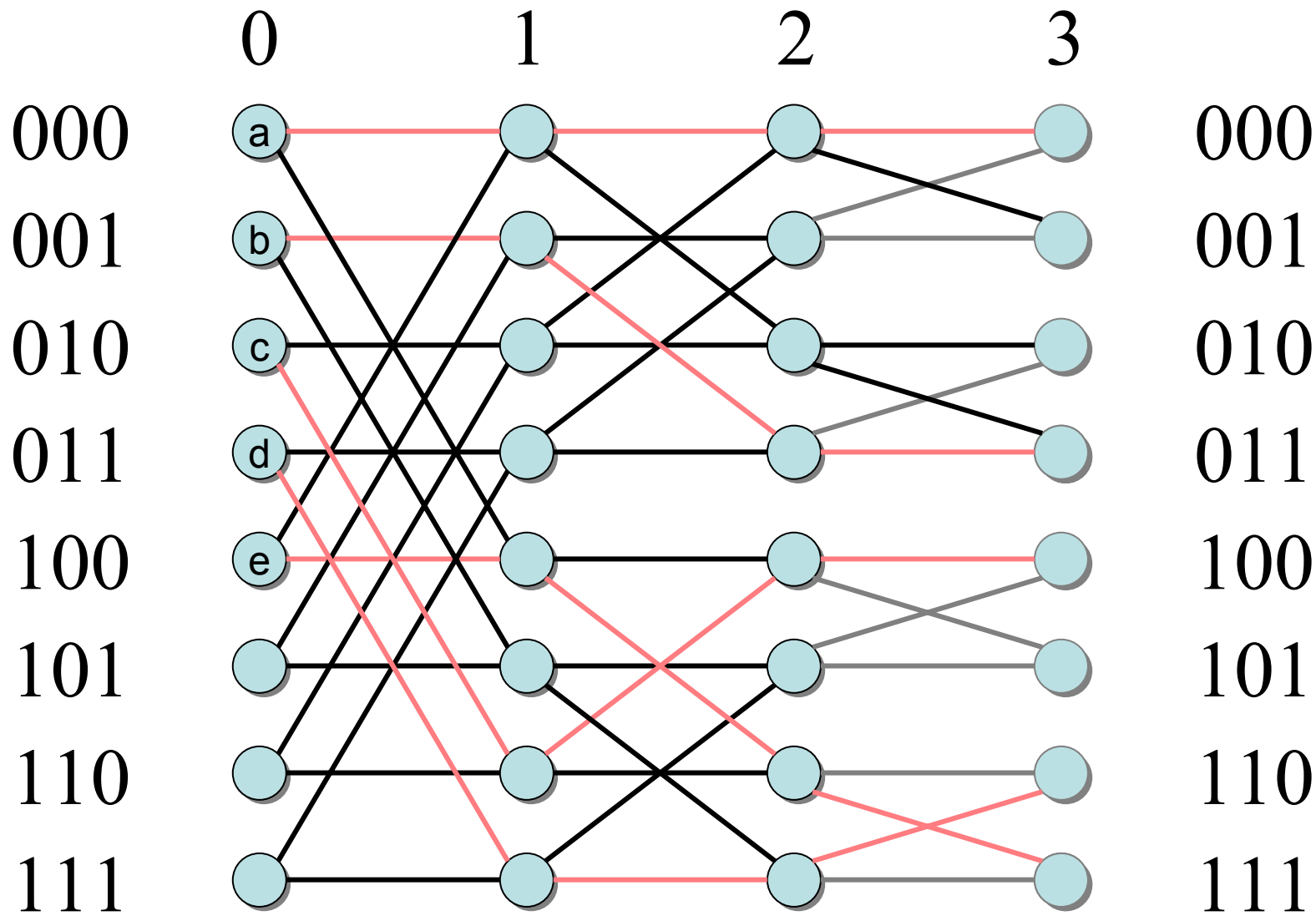
Many natural routing problems need  $O(\log N)$  steps.

*Packing problem* consists of routing  $M \leq N$  packets in level  $\log N$  of an  $N$  input butterfly into the first  $M$  processors in level 0, so that the relative order is unchanged.









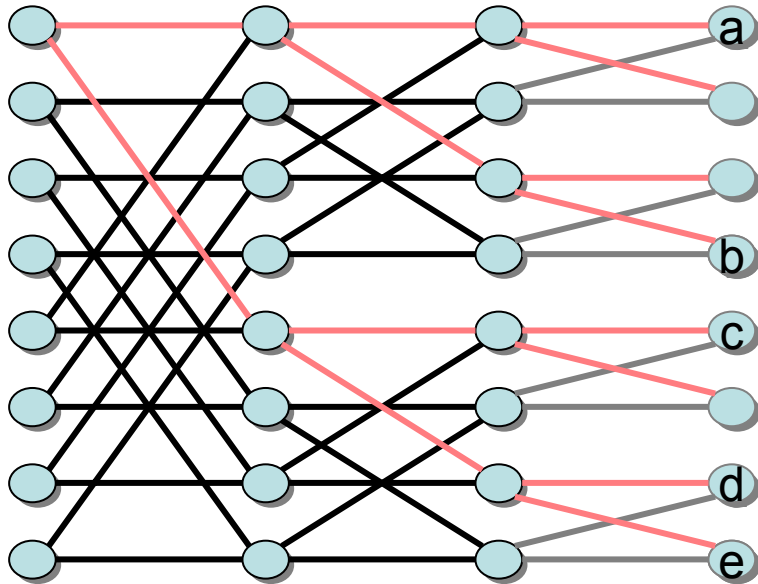
# Packing

Correct destinations of packets can be computed using the parallel prefix algorithm on a  $N$  leaf complete binary tree contained in the  $\log N$  dimensional butterfly.

We use the indicator function of the set with processors having a packet and addition as the associative prefix operator.

This can be done in  $2 \log N = O(\log N)$  steps using algorithm from 1.2.2

# Packing



$$x_j = \begin{cases} 1 & \text{if processor } \langle bin(j), \log N \rangle \text{ has a packet} \\ 0 & \text{if processor } \langle bin(j), \log N \rangle \text{ has no packet} \end{cases}$$

$$y_j = x_0 + x_1 + \dots + x_j$$



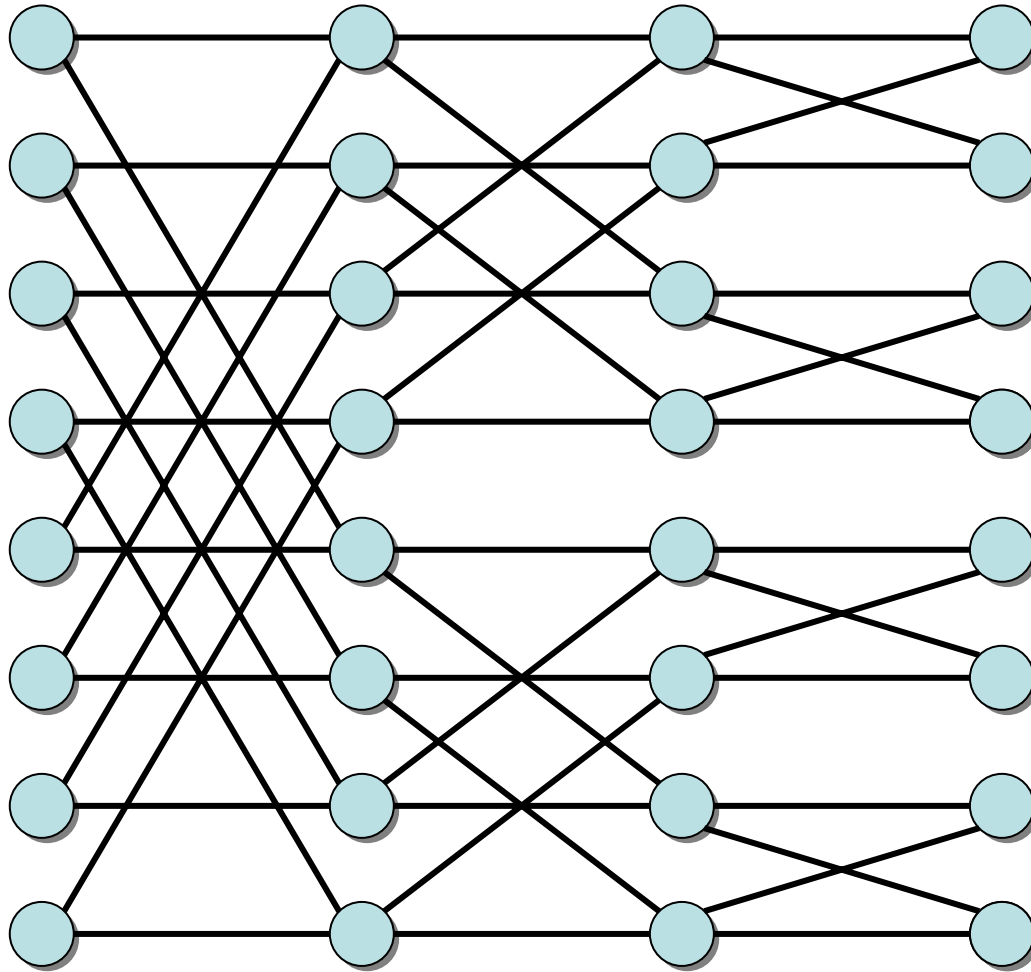
# Packing uses node disjoint paths

All packets need  $\log N$  steps to reach their destination. We can observe that the paths are node disjoint. The result holds only when packets depart from level  $\log N$ .

At the first step packets could collide only if they are from consecutive nodes. This can not happen since they will have consecutive destinations ergo they will differ in the last bit of the  $\log N - 1$  node.

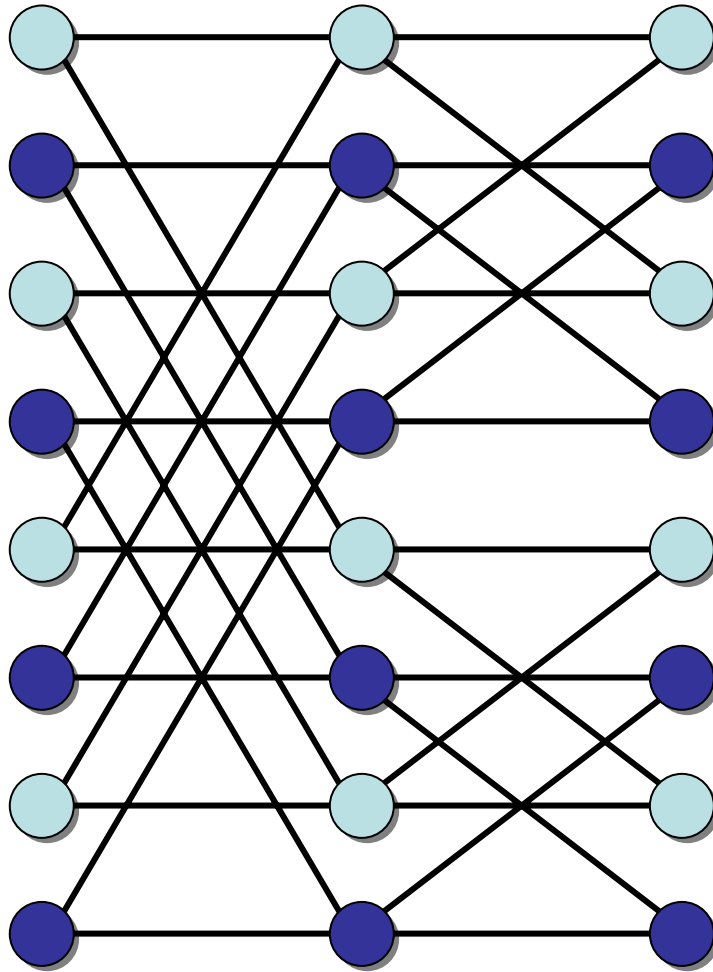
At the next steps the packets contained in the even (odd) rows move in the sub-butterfly of the even (odd) rows.

# Packing uses node disjoint paths (how to decompose a butterfly)

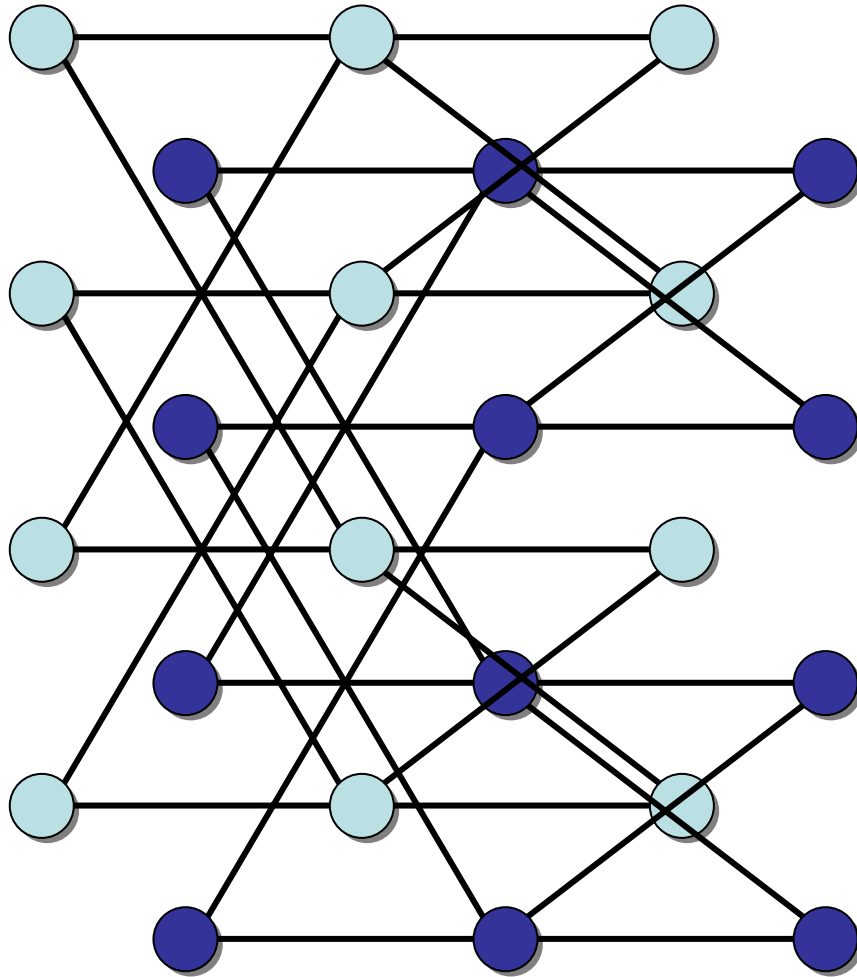




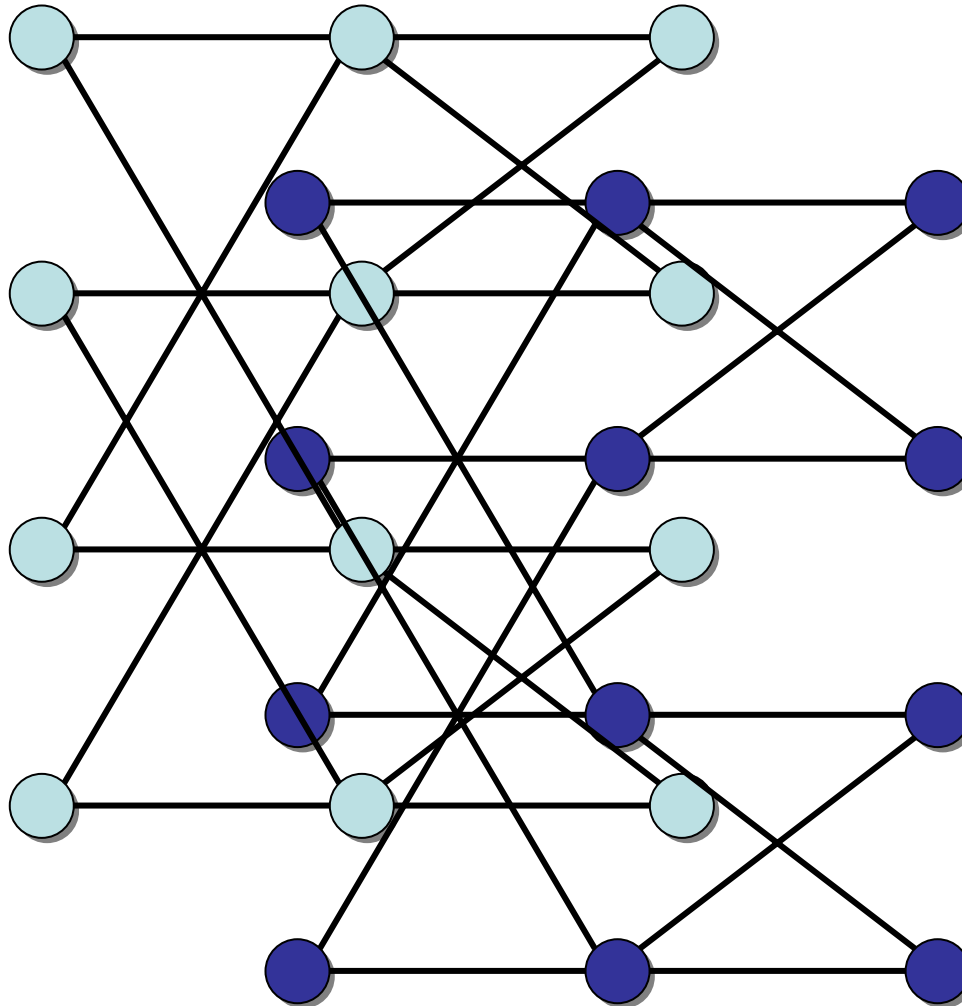
Packing uses node disjoint paths  
(how to decompose a butterfly)



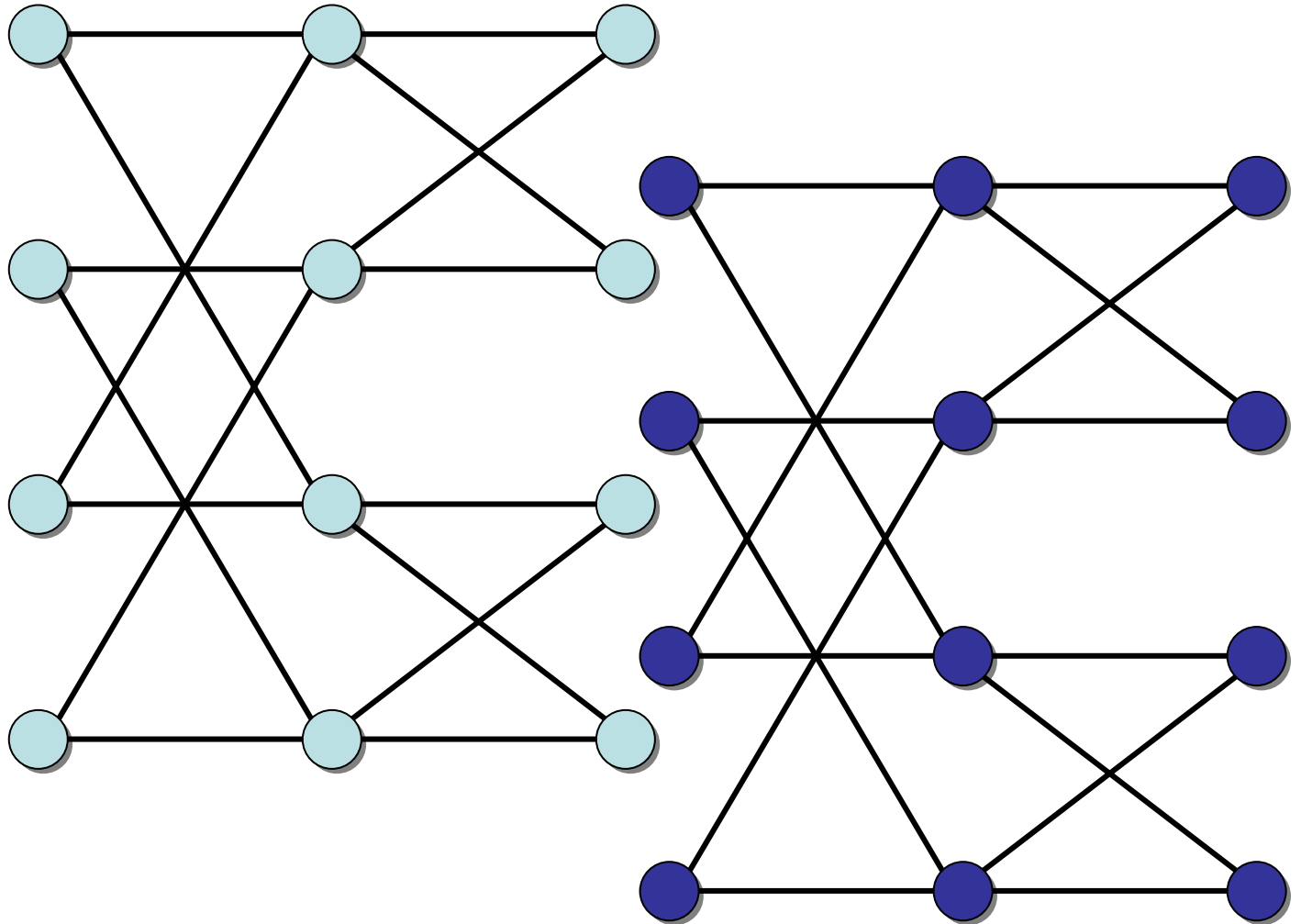
# Packing uses node disjoint paths (how to decompose a butterfly)



Packing uses node disjoint paths  
(how to decompose a butterfly)



# Packing uses node disjoint paths (how to decompose a butterfly)



# Generalizing Packing

- The destination of the first packet need not be at position 0. Any interval will do. Wrap around works also. The key assumption is that uppermost nodes at level 0 are contiguous.
- K-cyclic shift is a special case of packing
- Packing arises in binary switches.
- Algorithms discussed here are normal (one level – dimension is used at each step, consecutive levels are used at consecutive steps)

These algorithms can be used in a hypercube or in butterflies or N node shuffle exchange graph in  $2 \log N$  steps



# Spreading

- Spreading is the reverse of packing. (routing of  $M \leq N$  packets from level 0 to level  $\log N$  to a given destination constrain: relative order unchanged)
- With greedy routing needs  $\log N$ .

# Monotone Routing

- Monotone Routing is a routing problem where the relative order is unchanged.
- Examples: Packing and spreading.
- Algorithm:

Pack the packets to be routed. (from  $\log N$ ,  
 $\log N-1, \dots, 0$ )

Spread the packets (level  $0, 1, \dots, \log N$ )

- The algorithm needs  $O(\log N)$  steps and is normal.

# One 2 many Routing

- One 2 many routing is the problem in which some packets need to reach multiple destinations. Yet every destination node is the target of one packet.
- Algorithm:
- The packet at the  $i$ -th node will be sent to the  $y_{i-1}$  node.
- $x_i$  # destinations the packet at node  $i$  has.
- Calculate via parallel prefix operation.
- After the monotone routing use *segmented prefix computation* to make copies.
- A one 2 one problem remains.

# Many 2 many && many 2 one Routing

- We saw that one 2 many routing can be reduced to a one 2 one.
- Similarly a many 2 many can be reduced to many to one.
- With  $O(\log N)$  overhead we can solve problems where packets have multiple destinations to problems where they have one.

# Reducing routing to sorting

## Algorithm:

```
if M = N
```

```
sort().
```

```
If M < N
```

```
    sort(). /* T time steps*/
```

```
    apply spreading. /*O(log N)*/
```

- We need  $T + O(\log N)$  steps
- Since  $T = \Omega(\log N)$  we need  $O(T)$  steps.

# Reducing routing to sorting II

We can handle many to one routing by a modification of the previous algorithm in  $T + O(\log N)$  if we can use combining.

```
Sort(); /* T steps*/
```

```
Segmented_Prefix_Combine(); /*  $O(\log N)$  */
```

```
Monotone_route(); /* $O(\log N)$ */
```

$T + O(\log N)$  steps are needed.

# Average Case Behavior of greedy Routing

A big gap exists between worst and best case.

We will show: Average case is “near” the best case.

# What is Average Case?

Each packet has a random destination.

Packets start at level 0.

Each input node has  $p$  packets.

Queues will grow as needed.

Each node can send one packet at each step!



# Average Congestion

Congestion: Maximum number of packets passing through a node.

Bounded congestion implies:

- bounded queues,
- bounded running times for running time of the greedy algorithm.  $\log N + O(p) + o(\log N)$

# Bounding Average Congestion

$P_r(v)$  probability that  $r$  or more packets will pass through node  $v$ .  $v$  is on level  $i$ .

$p2^i$  packets can reach  $v$  from level 0.

$2^{\log N - i}$  destinations would cause one of those packets to go through  $v$ . ( $2^{\log N - i} = N2^{-i}$ ).

$N$  are all the destinations, probability of causing the packet to go through  $v$  is  $2^{-i}$ .

# Bounding Average Congestion II

$$\begin{aligned} P_r(v) &\leq \binom{p 2^i}{r} (2^{-i})^r \\ &\leq \left( \frac{p 2^i e}{r} \right)^r 2^{-ir} \\ &= \left( \frac{pe}{r} \right)^r \end{aligned}$$

# Bounding Average Congestion III

$P_r(v)$  does not depend on  $v$  or  $i$ ! So the probability that no node will be congested by more than  $r$  packets is  $N \log N (pe/r)^r$ .

By choosing  $r$  suitably large we can make this expression very low.

$$p \geq \frac{\log N}{2}, \text{ choose } r = 2ep \Rightarrow$$

$$N \log N \left( \frac{pe}{r} \right)^r \leq 1 / N^{3/2}$$

$$p \leq \frac{\log N}{2}, \text{ choose } r = \frac{2e \log N}{\log(\log N / p)} \Rightarrow$$

$$N \log N \left( \frac{pe}{r} \right)^r \leq 1 / N^2$$

# Theorem 3.24

For all but at most  $N^{-3/2}$  fraction of the possible routing problems with  $p$  packets per input at most  $C$  packets pass through a node where

$$C = \begin{cases} 2ep & \text{if } p \geq \frac{\log N}{2} \\ \frac{2e \log N}{\log(\log N / p)} & \text{if } p \leq \frac{\log N}{2} \end{cases}$$

# Theorem 3.25

For any  $\alpha$  all but at most  $N^{-\alpha}$  fraction of the possible routing problems with  $p$  packets per input at most  $C$  packets pass through a node where

$$C = O(\alpha p) + o(\alpha \log N)$$

Proof: is along the lines of 3.24 (adjust  $r$ )

# Random rank contention resolution

Random rank contention resolution. Each packet  $P$  is assigned a random priority key  $r(P) \in [1, K]$ .

Further ranking of the packets  $t(P)$  can be based on their origin resulting in a total order.

Definition of a total order

$$\underline{r}(P) = (r(P), t(P))$$

$$\underline{r}(P) < \underline{r}(P') \text{ iff } r(P) < r(P') \text{ or} \\ r(P) = r(P') \text{ and } t(P) < t(P')$$

The packet with the lowest rank in a queue is forwarded first.

# Theorem 3.26

Given any routing problem with contention  $C$  on a  $\log N$  dimensional butterfly the greedy algorithm will complete routing in  $T$  steps with probability at least  $1 - 1/N^7$ , when random rank protocol is used, where

$$T = \begin{cases} O(C) & \text{if } C \geq \frac{\log N}{2} \\ \log N + O(\log N / \log(\log(N) / C)) & \text{if } C \leq \frac{\log N}{2} \end{cases}$$



# Delay Sequence I

Let  $P_0$  be the last packet to reach its destination at time  $T$  and its destination is some  $v_0$ . Its rank is  $r(P_0)$

That packet was delayed last time during step  $T - l_0$  at some node  $v_1$  on level  $\log N - l_0$ .

Let  $P_1$  be the packet that moved forward at time  $T - l_0$ .

Then  $r(P_1) \leq r(P_0)$

That packet was delayed last time during step  $T - l_0 - l_1 - 1$  at some node  $v_2$  on level  $\log N - l_0 - l_1$ .

# Delay Sequence II

Let  $P_2$  be the packet that moved forward at time  $T - l_0 - l_1 - 1$ .

Then  $r(P_2) \leq r(P_1)$

That packet was delayed last time during step  $T - l_0 - l_1 - l_2 - 2$  at some node  $v_3$  on level  $\log N - l_0 - l_1 - l_2$ .

...

We can continue in that fashion, defining  $P_i, l_i, v_i$  until some level  $s-1$ . The process ends when we have defined a packet  $P_{s-1}$ , which was not delayed previously to delaying  $P_{s-2}$  at  $v_{s-1}$ . Additionally we define  $v_s$  the origin node of  $P_{s-1}$ , and  $l_{s-1}$  the distance of  $v_s$  and  $v_{s-1}$ .

# Delay Sequence III

By the construction we know:

- $\log N = l_0 + l_1 + l_2 + \dots + l_{s-1}$
- $P_{s-1}$  departs at step  $T - l_0 - l_1 - l_2 - \dots - l_{s-1} - (s-2) = 1$  ergo

$$T = \log N - s - 1$$

- $r(P_i) \leq r(P_{i-1})$  for  $1 \leq i \leq s-1$

We call the sequence of nodes  $v_0 \rightarrow v_1 \rightarrow v_2 \dots \rightarrow v_s$  the *delay path P*.

# Delay Sequence Definition

A delay sequence consists of

1. A delay path
2.  $s$  integers  $l_0 \geq 1, l_1 \geq 0, l_2 \geq 0, \dots, l_{s-1} \geq 0$  that sum up to  $\log N$ . They are lengths of sub-paths.
3.  $s+1$  nodes  $v_i$  such that  $v_i$  is on level  $\log N - l_0 - l_1 - \dots - l_{i-1}$  of  $P$ .
4.  $s$  different packets such that the greedy path of  $P_i$  contains  $v_i$ .
5. Keys  $k_i \in [1, K]$  for the packets such that the  $k_{i+1} \leq k_i$ .

# Counting Delay Paths

A delay sequence is *active* if  $r(P_i) = k_i$ .

Lots of delay sequences for a routing problem but probability that one is active is small

The probability that this happens is  $K^{-s}$  since  $r(P_i)$  are uniformly distributed in  $[1, K]$ .

The number of paths is  $N^2$ . A path is defined by its begin and end.

The list of  $s$  sub-path lengths that sum up to  $\log N$  is

$$\binom{s + \log N - 2}{s - 1}$$

# Counting Delay Sequences

A delay sequence is *active* if  $r(P_i) = k_i$ .

After fixing the path and the sub-path lengths the nodes  $v_i$  where delay happened are also fixed.

At most  $C$  packets will pass through any of them. Hence there are at most  $C^s$  ways to choose those  $s$  packets.

Since for the keys we know that they must be ordered  $k_{i+1} \leq k_i$  and within  $[1, K]$ . This can be done in

$$\binom{s + K - 1}{s}$$

ways.

# Counting Delay Sequences II

The number of possible delay sequences with  $s$  packets is at most

$$\mathcal{N}_s \leq N^2 \binom{s + \log N - 1}{s - 1} C^s \binom{s + K - 1}{s}$$

# Counting Delay Sequences II

The probability that there is an active delay sequence with  $s$  packets

$$\begin{aligned} & N^2 \binom{s + \log N - 2}{s - 1} C^s \binom{s + K - 1}{s} K^{-s} \\ & \leq N^2 2^{s + \log N - 2} C^s \left( \frac{s + K - 1}{s} \right)^s e^s K^{-s} \\ & \leq N^3 \left[ \frac{2Ce(s + K - 1)}{sK} \right]^s \end{aligned}$$



# Counting Delay Sequences III

If we take:

$$K > s \Rightarrow N^3 \left[ \frac{2Ce(s + K - 1)}{sK} \right]^s \leq N^3 \left( \frac{4eC}{s} \right)^s$$

After some algebra the probability is  $=o(N^{-7})$  for

$$s = \begin{cases} 8eC & \text{if } C \geq \log N / 2 \\ 8e \log N / \log(\log N / C) & \text{if } C \leq \log N / 2 \end{cases}$$

# Running times

With probability  $o(N^{-7})$  there is no active delay path  $s$ . This also bounds corresponding running times  $T = \log N + s - 1$

Bigger running times will not be probable because packets leaving times constitute a stream.

$$T = \begin{cases} \log N + 8eC - 1 & \text{if } C \geq \log N / 2 \\ \log N + 8e \log N / \log(\log N / C) - 1 & \text{if } C \leq \log N / 2 \end{cases}$$

# Beyond Random rank

A contention resolution protocol is called non-predictive if it is deterministic and contention is resolved based on the history of the contending packets only.

Example: FIFO.

Same number of problem instances that require the same running time.

# Hashing

Average case running time favorable. Some interesting instances not! What can we do.

Use hashing to make problems look enough random. That way  $\log N + o(\log N)$  complexity will be exhibited.

*Hot spots* can also be avoided in a parallel machine using the butterfly as its interconnection network.

Let  $q$  # data units stored in each memory block.

We seek  $h:[1, qN] \rightarrow [1, qN]$

# r-wise independence

When is hashing function random enough?

We seek  $h:[1,qN] \rightarrow [1,qN]!$

A function  $h$  is said to be  $r$ -wise independent if

$$\Pr[h(x_1) = y_1, \dots, h(x_r) = y_r] = \Pr[h(x_1) = y_1] \cdots \Pr[h(x_r) = y_r]$$

# Theorem 3.33

Consider a random routing problem for which there are  $p$  packets per input in a  $N$  input butterfly, and for which each packet function has destination  $h(x)$  where  $h()$  is  $r$ -wise independent, such that  $\text{Prob}[h(x)=i]=1/N$  where  $r = O(p)+o(\log N)$ .

For all but at most  $N^{3/2}$  fraction of the possible routing problems with  $p$  packets per input at most  $C$  packets pass through a node where

$$T = \begin{cases} O(C) & \text{if } C \geq \frac{\log N}{2} \\ \log N + O(\log N / \log(\log(N) / C)) & \text{if } C \leq \frac{\log N}{2} \end{cases}$$

# Randomized Routing

The algorithm: Do two passes of the butterfly. One to a random destination and one greedy to the correct.

With high probability the problem will be solved in  $2 \log N + o(\log N) + O(p)$  steps.

Analysis assumes random rank and no barriers.

# Bounded Queues

We assume that there are two input queues at each node, each with a bounded size.

Each link can forward one packet at each step, and each node can receive one packet at each input each step.

Nodes can sense if a forward queue is full.

Each node will forward one real packet at each step.

Complex dynamics due to *backpressure*.

Analysis of the vanilla greedy routing with bounded queues yielded no elegant and general results.



# Ranades Algorithm

A variation of greedy routing with

- easy analysis
- use in combining
- usable in other networks
- Guarantees use of bounded buffers.

# Ranade's Algorithm

Ranade algorithm makes use of the random rank contention resolution labels.

New invariant will be that packets passing through a queue, do so according to the labels.

Packets are entered into the network in sorted order.

Each node can additionally forward one ghost packet at each step. They signal that no packet with a lower rank will follow from that node over that link.

Ghost messages contain as information the rank of a packet. They can be discarded if a real packet needs the buffers. They are also discarded when they stop moving forward.

# Ranade's Algorithm

Nodes are in active or quiescent state.

A packet is in quiescent state

- If no packet has reached the node or
- it has sent all packets.

Active nodes have two packets (real or ghost) at the head of their queues.

# Ranade's Algorithm

## Send ( )

If node is\_active() :

At each step the packet with the lowest rank is chosen.

If it is a ghost message it is sent via both links.

If it is a real packet the queue ahead is sensed.

If the queue has room it is sent via the corresponding link and a ghost with that label is sent via the other link.

If there is no room the ghost message is sent over both links.

# Theorem 3.34 on Ranade's Algorithm

Given any greedy routing problem with congestion  $C$  of a  $\log N$  dimensional butterfly with queues of maximum size  $Q$  Ranade's algorithm will complete the routing of all the packets in  $T$  steps with probability at least  $1 - O(N^{-\alpha})$  where

$$T = \begin{cases} O(C) & \text{if } C \geq \log N / 2 \\ \log N + O(\log N / \log N(\log N / C)) & \text{if } C \geq \log N / 2 \end{cases}$$