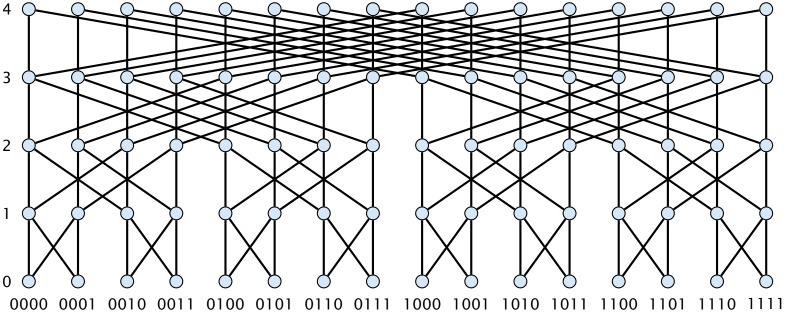
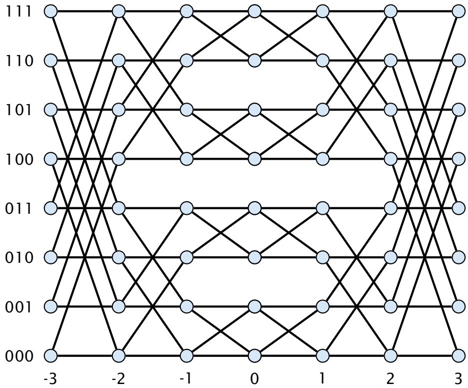
Bufferfly Network BF(d)



- node set $V = \{(\ell, \bar{x}) \mid \bar{x} \in [2]^d, \ell \in [d+1]\}$, where $\bar{x} = x_0 x_1 \ldots x_{d-1}$ is a bit-string of length d
- edge set $E = \{\{(\ell, \bar{x}), (\ell+1, \bar{x}')\} \mid \ell \in [d], \bar{x} \in [2]^d, x'_i = x_i \text{ for } i \neq \ell\}$

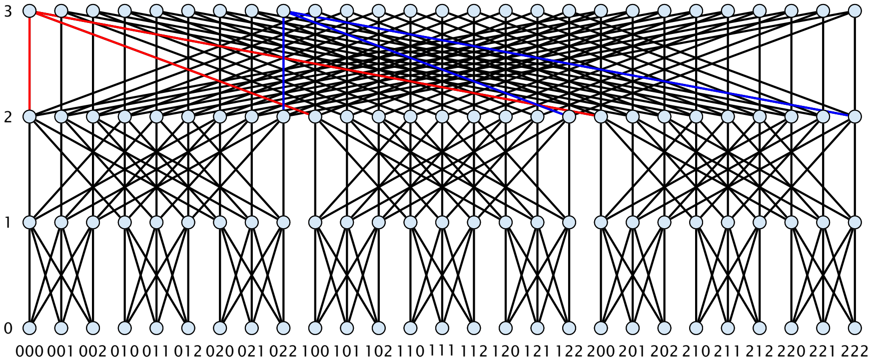
Sometimes the first and last level are identified.

Beneš Network



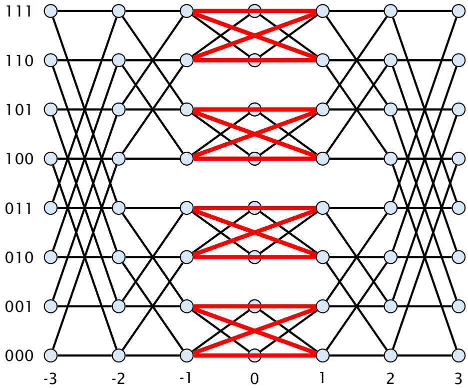
- node set $V = \{(\ell, \bar{x}) \mid \bar{x} \in [2]^d, \ell \in \{-d, \dots, d\}\}$
- edge set
  $E = \{\{(\ell, \bar{x}), (\ell + 1, \bar{x}')\} \mid \ell \in [d], \bar{x} \in [2]^d, x_i' = x_i \text{ for } i \neq \ell\}$
  $\cup \{\{(-\ell, \bar{x}), (\ell - 1, \bar{x}')\} \mid \ell \in [d], \bar{x} \in [2]^d, x_i' = x_i \text{ for } i \neq \ell\}$

n-ary Bufferfly Network BF(n, d)



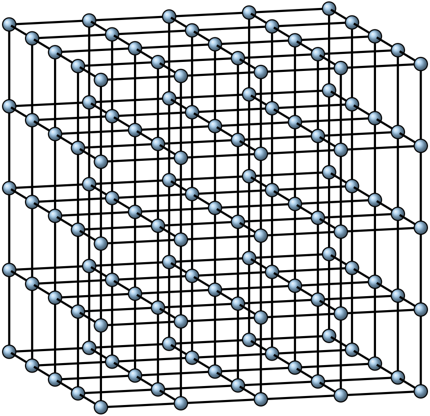
- node set $V = \{(\ell, \bar{x}) \mid \bar{x} \in [n]^d, \ell \in [d+1]\}$, where $\bar{x} = x_0 \, x_1 \, \ldots x_{d-1}$ is a bit-string of length d
- edge set $E = \{\{(\ell, \bar{x}), (\ell + 1, \bar{x}')\} \mid \ell \in [d], \bar{x} \in [n]^d, x'_i = x_i \text{ for } i \neq \ell\}$

Permutation Network $\mathrm{PN}(n, d)$



- There is an n-ary version of the Benes network (2 n-ary butterflies glued at level 0).
- identifying levels 0 and 1 (or 0 and -1) gives $\mathrm{PN}(n, d)$.

The d-dimensional mesh $M(n, d)$



- node set $V = [n]^d$
- edge set $E = \{\{(x_0, \dots, x_i, \dots, x_{d-1}), (x_0, \dots, x_i + 1, \dots, x_{d-1})\} \mid x_s \in [n] \text{ for } s \in [d] \setminus \{i\}, x_i \in [n-1]\}$

Remarks

$M(2,d)$ is also called d-dimensional hypercube.

$M(n,1)$ is also called linear array of length n.

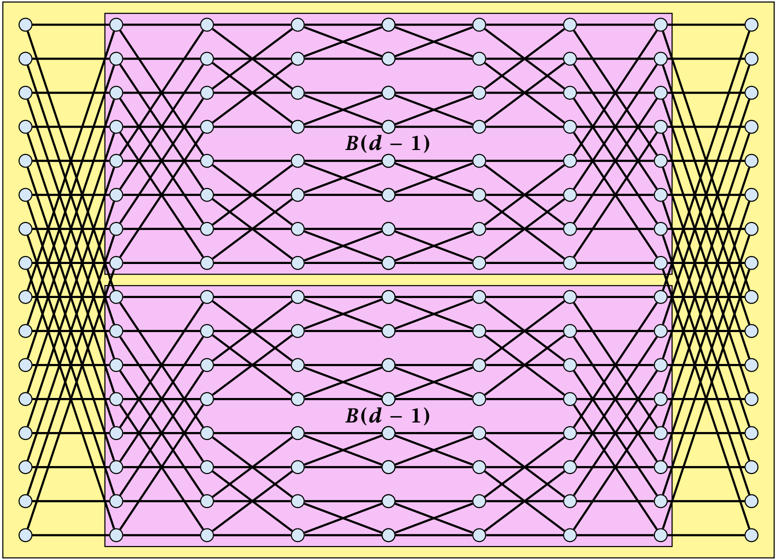
Permutation Routing

Lemma 1
On the linear array $M(n,1)$ any permutation can be routed online in $2n$ steps with buffersize 3.

Permutation Routing

Lemma 2
On the Beneš network any permutation can be routed offline in $2d$ steps between the sources level $(+d)$ and target level $(-d)$.

Recursive Beneš Network



Permutation Routing

base case $d = 0$
trivial

induction step $d \to d + 1$

- The packets that start at (α, d) and $(\alpha(\bar{d}), d)$ have to be sent into different sub-networks.
- The packets that end at $(\alpha, -d)$ and $(\alpha(\bar{d}), -d)$ have to come out of different sub-networks.

We can generate a graph on the set of packets.

- Every packet has an incident source edge (connecting it to the conflicting start packet)
- Every packet has an incident target edge (connecting it to the conflicting packet at its target)
- This clearly gives a bipartite graph; Coloring this graph tells us which packet to send into which sub-network.

Permutation Routing

base case $d = 0$

trivial

induction step $d \to d + 1$

- The packets that start at $(\bar{a}, d)$ and $(\bar{a}(d), d)$ have to be sent into different sub-networks.
- The packets that end at $(\bar{a}, -d)$ and $(\bar{a}(d), -d)$ have to come out of different sub-networks.

We can generate a graph on the set of packets.

- Every packet has an incident source edge (connecting it to the conflicting start packet)
- Every packet has an incident target edge (connecting it to the conflicting packet at its target)
- This clearly gives a bipartite graph; Coloring this graph tells us which packet to send into which sub-network.

Permutation Routing

base case $d = 0$

trivial

induction step $d \to d + 1$

- The packets that start at $(\bar{a}, d)$ and $(\bar{a}(d), d)$ have to be sent into different sub-networks.
- The packets that end at $(\bar{a}, -d)$ and $(\bar{a}(d), -d)$ have to come out of different sub-networks.

We can generate a graph on the set of packets.

- Every packet has an incident source edge (connecting it to the conflicting start packet)
- Every packet has an incident target edge (connecting it to the conflicting packet at its target)
- This clearly gives a bipartite graph; Coloring this graph tells us which packet to send into which sub-network.

Permutation Routing

base case $d = 0$

trivial

induction step $d \to d + 1$

- The packets that start at $(\bar{a}, d)$ and $(\bar{a}(d), d)$ have to be sent into different sub-networks.
- The packets that end at $(\bar{a}, -d)$ and $(\bar{a}(d), -d)$ have to come out of different sub-networks.

We can generate a graph on the set of packets.

Permutation Routing

base case $d = 0$

trivial

induction step $d \to d + 1$

- The packets that start at $(\bar{a}, d)$ and $(\bar{a}(d), d)$ have to be sent into different sub-networks.
- The packets that end at $(\bar{a}, -d)$ and $(\bar{a}(d), -d)$ have to come out of different sub-networks.

We can generate a graph on the set of packets.

- Every packet has an incident source edge (connecting it to the conflicting start packet)
- Every packet has an incident target edge (connecting it to the conflicting packet at its target)
- This clearly gives a bipartite graph; Coloring this graph tells us which packet to send into which sub-network.

Permutation Routing

base case $d = 0$

trivial

induction step $d \to d + 1$

- The packets that start at $(\bar{a}, d)$ and $(\bar{a}(d), d)$ have to be sent into different sub-networks.
- The packets that end at $(\bar{a}, -d)$ and $(\bar{a}(d), -d)$ have to come out of different sub-networks.

We can generate a graph on the set of packets.

- Every packet has an incident source edge (connecting it to the conflicting start packet)
- Every packet has an incident target edge (connecting it to the conflicting packet at its target)
- This clearly gives a bipartite graph; Coloring this graph tells us which packet to send into which sub-network.

Permutation Routing

base case $d = 0$

trivial

induction step $d \to d + 1$

- The packets that start at $(\bar{a}, d)$ and $(\bar{a}(d), d)$ have to be sent into different sub-networks.
- The packets that end at $(\bar{a}, -d)$ and $(\bar{a}(d), -d)$ have to come out of different sub-networks.

We can generate a graph on the set of packets.

- Every packet has an incident source edge (connecting it to the conflicting start packet)
- Every packet has an incident target edge (connecting it to the conflicting packet at its target)
- This clearly gives a bipartite graph; Coloring this graph tells us which packet to send into which sub-network.

Permutation Routing on the n-ary Beneš Network

Instead of two we have n sub-networks $B(n, d-1)$.

All packets starting at positions $\{(x_0, \ldots, x_i, \ldots, x_{d-1}, d) \mid x_i \in [n]\}$ have to be send to different sub-networks.

All packets ending at positions $\{(x_0, \ldots, x_i, \ldots, x_{d-1}, d) \mid x_i \in [n]\}$ have to come from different sub-networks.

The conflict graph is a n-uniform 2-regular hypergraph.

We can color such a graph with n colors such that no two nodes in a hyperedge share a color.

This gives the routing.

Permutation Routing on the n-ary Beneš Network

Instead of two we have n sub-networks $B(n, d-1)$.

All packets starting at positions $\{(x_0, \dots, x_i, \dots, x_{d-1}, d) \mid x_i \in [n]\}$ have to be send to different sub-networks.

All packets ending at positions $\{(x_0, \dots, x_i, \dots, x_{d-1}, d) \mid x_i \in [n]\}$ have to come from different sub-networks.

The conflict graph is a n-uniform 2-regular hypergraph.

We can color such a graph with n colors such that no two nodes in a hyperedge share a color.

This gives the routing.

Permutation Routing on the n-ary Beneš Network

Instead of two we have n sub-networks $B(n, d-1)$.

All packets starting at positions $\{(x_0, \ldots, x_i, \ldots, x_{d-1}, d) \mid x_i \in [n]\}$ have to be send to different sub-networks.

All packets ending at positions $\{(x_0, \ldots, x_i, \ldots, x_{d-1}, d) \mid x_i \in [n]\}$ have to come from different sub-networks.

The conflict graph is a n-uniform 2-regular hypergraph.

We can color such a graph with n colors such that no two nodes in a hyperedge share a color.

This gives the routing.

Permutation Routing on the n-ary Beneš Network

Instead of two we have n sub-networks $B(n, d-1)$.

All packets starting at positions $\{(x_0, \dots, x_i, \dots, x_{d-1}, d) \mid x_i \in [n]\}$ have to be send to different sub-networks.

All packets ending at positions $\{(x_0, \dots, x_i, \dots, x_{d-1}, d) \mid x_i \in [n]\}$ have to come from different sub-networks.

The conflict graph is a n-uniform 2-regular hypergraph.

We can color such a graph with n colors such that no two nodes in a hyperedge share a color.

This gives the routing.

Permutation Routing on the n-ary Beneš Network

Instead of two we have n sub-networks $B(n, d-1)$.

All packets starting at positions $\{(x_0, \ldots, x_i, \ldots, x_{d-1}, d) \mid x_i \in [n]\}$ have to be send to different sub-networks.

All packets ending at positions $\{(x_0, \ldots, x_i, \ldots, x_{d-1}, d) \mid x_i \in [n]\}$ have to come from different sub-networks.

The conflict graph is a n-uniform 2-regular hypergraph.

We can color such a graph with n colors such that no two nodes in a hyperedge share a color.

This gives the routing.

Permutation Routing on the n-ary Beneš Network

Instead of two we have n sub-networks $B(n, d-1)$.

All packets starting at positions $\{(x_0, \ldots, x_i, \ldots, x_{d-1}, d) \mid x_i \in [n]\}$ have to be send to different sub-networks.

All packets ending at positions $\{(x_0, \ldots, x_i, \ldots, x_{d-1}, d) \mid x_i \in [n]\}$ have to come from different sub-networks.

The conflict graph is a n-uniform 2-regular hypergraph.

We can color such a graph with n colors such that no two nodes in a hyperedge share a color.

This gives the routing.

Permutation Routing on the n-ary Beneš Network

Instead of two we have n sub-networks $B(n, d-1)$.

All packets starting at positions
$\{(x_0, \ldots, x_i, \ldots, x_{d-1}, d) \mid x_i \in [n]\}$ have to be send to different sub-networks.

All packets ending at positions
$\{(x_0, \ldots, x_i, \ldots, x_{d-1}, d) \mid x_i \in [n]\}$ have to come from different sub-networks.

The conflict graph is a n-uniform 2-regular hypergraph.

We can color such a graph with n colors such that no two nodes in a hyperedge share a color.

This gives the routing.

Lemma 3

On a d-dimensional mesh with sidelength n we can route any permutation (offline) in $4dn$ steps.

We can simulate the algorithm for the n-ary Beneš Network.

Each step can be simulated by routing on disjoint linear arrays. This takes at most $2n$ steps.

We can simulate the algorithm for the n-ary Beneš Network.

Each step can be simulated by routing on disjoint linear arrays. This takes at most $2n$ steps.

We simulate the behaviour of the Beneš network on the n-dimensional mesh.

In round $r \in \{-d, \ldots, -1, 0, 1, \ldots, d-1\}$ we simulate the step of sending from level r of the Beneš network to level $r+1$.

Each node $\bar{x} \in [n]^d$ of the mesh simulates the node $(r, \bar{x})$.

Hence, if in the Beneš network we send from $(r, \bar{x})$ to $(r+1, \bar{x}')$ we have to send from $\bar{x}$ to $\bar{x}'$ in the mesh.

All communication is performed along linear arrays. In round $r < 0$ the linear arrays along dimension $-r-1$ (recall that dimensions are numbered from 0 to $d-1$) are used

$$\bar{x}_{d-1} \ldots \bar{x}_{-r} \alpha \bar{x}_{-r-2} \ldots \bar{x}_0$$

In rounds $r \geq 0$ linear arrays along dimension r are used.

Hence, we can perform a round in $\mathcal{O}(n)$ steps.

We simulate the behaviour of the Beneš network on the n-dimensional mesh.

In round $r \in \{-d, \ldots, -1, 0, 1, \ldots, d-1\}$ we simulate the step of sending from level r of the Beneš network to level $r+1$.

Each node $\bar{x} \in [n]^d$ of the mesh simulates the node $(r, \bar{x})$.

Hence, if in the Beneš network we send from $(r, \bar{x})$ to $(r+1, \bar{x}')$ we have to send from $\bar{x}$ to $\bar{x}'$ in the mesh.

All communication is performed along linear arrays. In round $r < 0$ the linear arrays along dimension $-r-1$ (recall that dimensions are numbered from 0 to $d-1$) are used

$$\bar{x}_{d-1} \ldots \bar{x}_{-r} \alpha \bar{x}_{-r-2} \ldots \bar{x}_0$$

In rounds $r \geq 0$ linear arrays along dimension r are used.

Hence, we can perform a round in $\mathcal{O}(n)$ steps.

We simulate the behaviour of the Beneš network on the n-dimensional mesh.

In round $r \in \{-d, \dots, -1, 0, 1, \dots, d-1\}$ we simulate the step of sending from level r of the Beneš network to level $r+1$.

Each node $\bar{x} \in [n]^d$ of the mesh simulates the node $(r, \bar{x})$.

Hence, if in the Beneš network we send from $(r, \bar{x})$ to $(r+1, \bar{x}')$ we have to send from $\bar{x}$ to $\bar{x}'$ in the mesh.

All communication is performed along linear arrays. In round $r < 0$ the linear arrays along dimension $-r-1$ (recall that dimensions are numbered from 0 to $d-1$) are used

$$\bar{x}_{d-1} \dots \bar{x}_{-r} \alpha \bar{x}_{-r-2} \dots \bar{x}_0$$

In rounds $r \geq 0$ linear arrays along dimension r are used.

Hence, we can perform a round in $\mathcal{O}(n)$ steps.

We simulate the behaviour of the Beneš network on the n-dimensional mesh.

In round $r \in \{-d, \ldots, -1, 0, 1, \ldots, d-1\}$ we simulate the step of sending from level r of the Beneš network to level $r+1$.

Each node $\bar{x} \in [n]^d$ of the mesh simulates the node $(r, \bar{x})$.

Hence, if in the Beneš network we send from $(r, \bar{x})$ to $(r+1, \bar{x}')$ we have to send from $\bar{x}$ to $\bar{x}'$ in the mesh.

All communication is performed along linear arrays. In round $r < 0$ the linear arrays along dimension $-r-1$ (recall that dimensions are numbered from 0 to $d-1$) are used

$$\bar{x}_{d-1} \ldots \bar{x}_{-r} \alpha \bar{x}_{-r-2} \ldots \bar{x}_0$$

In rounds $r \geq 0$ linear arrays along dimension r are used.

Hence, we can perform a round in $\mathcal{O}(n)$ steps.

We simulate the behaviour of the Beneš network on the n-dimensional mesh.

In round $r \in \{-d, \dots, -1, 0, 1, \dots, d-1\}$ we simulate the step of sending from level r of the Beneš network to level $r+1$.

Each node $\bar{x} \in [n]^d$ of the mesh simulates the node $(r, \bar{x})$.

Hence, if in the Beneš network we send from $(r, \bar{x})$ to $(r+1, \bar{x}')$ we have to send from $\bar{x}$ to $\bar{x}'$ in the mesh.

All communication is performed along linear arrays. In round $r < 0$ the linear arrays along dimension $-r-1$ (recall that dimensions are numbered from 0 to $d-1$) are used

$$\bar{x}_{d-1} \dots \bar{x}_{-r} \alpha \bar{x}_{-r-2} \dots \bar{x}_0$$

In rounds $r \geq 0$ linear arrays along dimension r are used.

Hence, we can perform a round in $\mathcal{O}(n)$ steps.

We simulate the behaviour of the Beneš network on the n-dimensional mesh.

In round $r \in \{-d, \ldots, -1, 0, 1, \ldots, d-1\}$ we simulate the step of sending from level r of the Beneš network to level $r+1$.

Each node $\bar{x} \in [n]^d$ of the mesh simulates the node $(r, \bar{x})$.

Hence, if in the Beneš network we send from $(r, \bar{x})$ to $(r+1, \bar{x}')$ we have to send from $\bar{x}$ to $\bar{x}'$ in the mesh.

All communication is performed along linear arrays. In round $r < 0$ the linear arrays along dimension $-r-1$ (recall that dimensions are numbered from 0 to $d-1$) are used

$$\bar{x}_{d-1} \ldots \bar{x}_{-r} \alpha \bar{x}_{-r-2} \ldots \bar{x}_0$$

In rounds $r \geq 0$ linear arrays along dimension r are used.

Hence, we can perform a round in $\mathcal{O}(n)$ steps.

We simulate the behaviour of the Beneš network on the n-dimensional mesh.

In round $r \in \{-d, \ldots, -1, 0, 1, \ldots, d-1\}$ we simulate the step of sending from level r of the Beneš network to level $r+1$.

Each node $\bar{x} \in [n]^d$ of the mesh simulates the node $(r, \bar{x})$.

Hence, if in the Beneš network we send from $(r, \bar{x})$ to $(r+1, \bar{x}')$ we have to send from $\bar{x}$ to $\bar{x}'$ in the mesh.

All communication is performed along linear arrays. In round $r < 0$ the linear arrays along dimension $-r-1$ (recall that dimensions are numbered from 0 to $d-1$) are used

$$\bar{x}_{d-1} \ldots \bar{x}_{-r} \alpha \bar{x}_{-r-2} \ldots \bar{x}_0$$

In rounds $r \geq 0$ linear arrays along dimension r are used.

Hence, we can perform a round in $\mathcal{O}(n)$ steps.

Lemma 4

We can route any permutation on the Beneš network in $\mathcal{O}(d)$ steps with constant buffer size.

The same is true for the butterfly network.

Lemma 4

We can route any permutation on the Beneš network in $\mathcal{O}(d)$ steps with constant buffer size.

The same is true for the butterfly network.

The nodes are of the form $(\ell, \bar{x})$, $\bar{x} \in [n]^d, \ell \in -d, \ldots, d$.

We can view nodes with same first coordinate forming columns and nodes with the same second coordinate as forming rows. This gives rows of length $2d + 1$ and columns of length n^d.

We route in 3 phases:

1. Permute packets along the rows such that afterwards no column contains packets that have the same target row. $O(d)$ steps.
2. We can use pipeling to permute every column, so that afterwards every packet is in its target row. $O(2d + 2d)$ steps.
3. Every packet is in its target row. Permute packets to their right destinations. $O(d)$ steps.

The nodes are of the form $(\ell, \bar{x})$, $\bar{x} \in [n]^d, \ell \in -d, \dots, d$.

We can view nodes with same first coordinate forming columns and nodes with the same second coordinate as forming rows. This gives rows of length $2d + 1$ and columns of length n^d.

We route in 3 phases:

1. Permute packets along the rows such that afterwards no column contains packets that have the same target row. $\mathcal{O}(d)$ steps.
2. We can use pipeling to permute every column, so that afterwards every packet is in its target row. $\mathcal{O}(2d + 2d)$ steps.
3. Every packet is in its target row. Permute packets to their right destinations. $\mathcal{O}(d)$ steps.

The nodes are of the form $(\ell, \bar{x})$, $\bar{x} \in [n]^d, \ell \in -d, \dots, d$.

We can view nodes with same first coordinate forming columns and nodes with the same second coordinate as forming rows. This gives rows of length $2d + 1$ and columns of length n^d.

We route in 3 phases:

1. Permute packets along the rows such that afterwards no column contains packets that have the same target row. $\mathcal{O}(d)$ steps.
2. We can use pipeling to permute **every** column, so that afterwards every packet is in its target row. $\mathcal{O}(2d + 2d)$ steps.
3. Every packet is in its target row. Permute packets to their right destinations. $\mathcal{O}(d)$ steps.

The nodes are of the form $(\ell, \bar{x})$, $\bar{x} \in [n]^d, \ell \in -d, \dots, d$.

We can view nodes with same first coordinate forming columns and nodes with the same second coordinate as forming rows. This gives rows of length $2d + 1$ and columns of length n^d.

We route in 3 phases:

1. Permute packets along the rows such that afterwards no column contains packets that have the same target row. $\mathcal{O}(d)$ steps.
2. We can use pipeling to permute **every** column, so that afterwards every packet is in its target row. $\mathcal{O}(2d + 2d)$ steps.
3. Every packet is in its target row. Permute packets to their right destinations. $\mathcal{O}(d)$ steps.

The nodes are of the form $(\ell, \bar{x})$, $\bar{x} \in [n]^d, \ell \in -d, \ldots, d$.

We can view nodes with same first coordinate forming columns and nodes with the same second coordinate as forming rows. This gives rows of length $2d + 1$ and columns of length n^d.

We route in 3 phases:

1. Permute packets along the rows such that afterwards no column contains packets that have the same target row. $\mathcal{O}(d)$ steps.
2. We can use pipeling to permute **every** column, so that afterwards every packet is in its target row. $\mathcal{O}(2d + 2d)$ steps.
3. Every packet is in its target row. Permute packets to their right destinations. $\mathcal{O}(d)$ steps.

Lemma 5

We can do offline permutation routing of (partial) permutations in $2d$ steps on the hypercube.

Lemma 6

We can sort on the hypercube $M(2,d)$ in $\mathcal{O}(d^2)$ steps.

Lemma 7

We can do online permutation routing of permutations in $\mathcal{O}(d^2)$ steps on the hypercube.

Lemma 5

We can do offline permutation routing of (partial) permutations in $2d$ steps on the hypercube.

Lemma 6

We can sort on the hypercube $M(2,d)$ in $\mathcal{O}(d^2)$ steps.

Lemma 7

We can do online permutation routing of permutations in $\mathcal{O}(d^2)$ steps on the hypercube.

Lemma 5

We can do offline permutation routing of (partial) permutations in $2d$ steps on the hypercube.

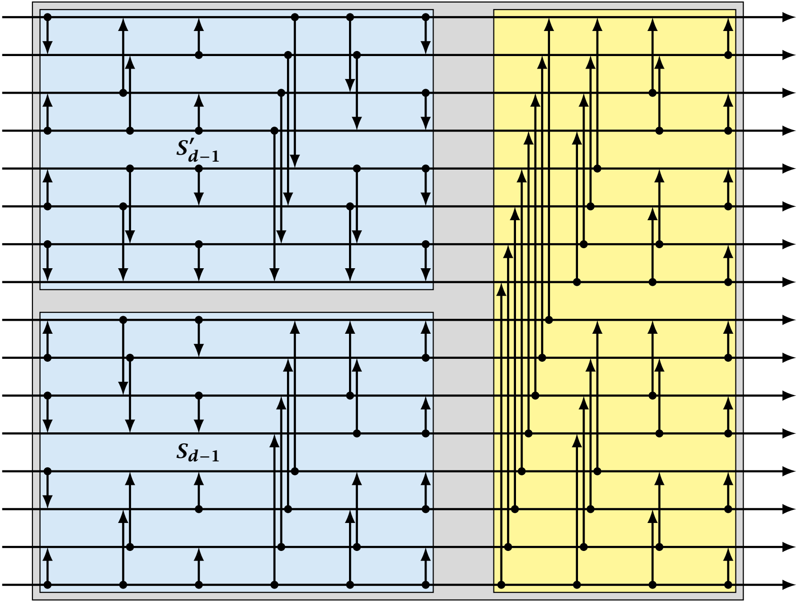
Lemma 6

We can sort on the hypercube $M(2,d)$ in $\mathcal{O}(d^2)$ steps.

Lemma 7

We can do online permutation routing of permutations in $\mathcal{O}(d^2)$ steps on the hypercube.

Bitonic Sorter S_d



ASCEND/DESCEND Programs

Algorithm 11 ASCEND(procedure *oper*)

```
1: for dim = 0 to d − 1
2:     for all ā ∈ [2]^d pardo
3:         oper(ā, ā(dim), dim)
```

Algorithm 11 DESCEND(procedure *oper*)

```
1: for dim = d − 1 to 0
2:     for all ā ∈ [2]^d pardo
3:         oper(ā, ā(dim), dim)
```

oper should only depend on the dimension and on values stored in the respective processor pair $(\bar{a}, \bar{a}(dim), V[\bar{a}], V[\bar{a}(dim)])$.

oper should take constant time.

Algorithm 11 oper$(a, a', dim, T_a, T_{a'})$

1: **if** $a_{dim}, \ldots, a_0 = 0^{dim+1}$ **then**
2: $\quad T_a = \min\{T_a, T_{a'}\}$

Performing an ASCEND run with this operation computes the minimum in processor 0.

We can sort on $M(2,d)$ by using d DESCEND runs.

We can do offline permutation routing by using a DESCEND run followed by an ASCEND run.

Algorithm 11 oper$(a, a', dim, T_a, T_{a'})$

1: **if** $a_{dim}, \dots, a_0 = 0^{dim+1}$ **then**
2: $\quad T_a = \min\{T_a, T_{a'}\}$

Performing an ASCEND run with this operation computes the minimum in processor 0.

We can sort on $M(2, d)$ by using d DESCEND runs.

We can do offline permutation routing by using a DESCEND run followed by an ASCEND run.

Algorithm 11 oper$(a, a', dim, T_a, T_{a'})$

if $a_{dim}, \ldots, a_0 = 0^{dim+1}$ **then**
    $T_a = \min\{T_a, T_{a'}\}$

Performing an ASCEND run with this operation computes the minimum in processor 0.

We can sort on $M(2, d)$ by using d DESCEND runs.

We can do offline permutation routing by using a DESCEND run followed by an ASCEND run.

Algorithm 11 oper$(a, a', dim, T_a, T_{a'})$

---

if $a_{dim}, \ldots, a_0 = 0^{dim+1}$ **then**

  $T_a = \min\{T_a, T_{a'}\}$

Performing an ASCEND run with this operation computes the minimum in processor 0.

We can sort on $M(2,d)$ by using d DESCEND runs.

We can do offline permutation routing by using a DESCEND run followed by an ASCEND run.

We can perform an ASCEND/DESCEND run on a linear array $M(2^d, 1)$ in $\mathcal{O}(2^d)$ steps.

The CCC network is obtained from a hypercube by replacing every node by a cycle of degree d.

- nodes $\{(\ell, \bar{x}) \mid \bar{x} \in [2]^d, \ell \in [d]\}$
- edges $\{\{(\ell, \bar{x}), (\ell, \bar{x}(\ell)\} \mid x \in [2]^d, \ell \in [d]\}$

constand degree

Lemma 8

Let $d = 2^k$. An ASCEND run of a hypercube $M(2, d+k)$ can be simulated on CCC(d) in $\mathcal{O}(d)$ steps.

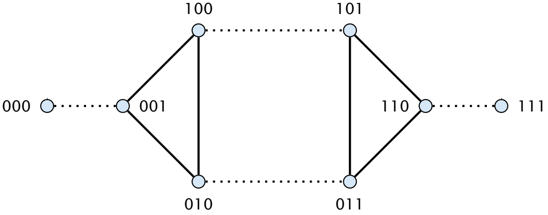
The shuffle exchange network SE(d) is defined as follows

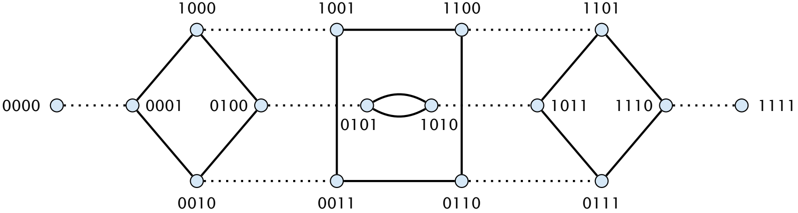
- nodes: $V = [2]^d$
- edges:
  $E = \Big\{\{x\bar{\alpha}, \bar{\alpha}x\} \mid x \in [2], \bar{\alpha} \in [2]^{d-1}\Big\} \cup \Big\{\{\bar{\alpha}0, \bar{\alpha}1\} \mid \bar{\alpha} \in [2]^{d-1}\Big\}$

constand degree

Edges of the first type are called shuffle edges. Edges of the second type are called exchange edges

Shuffle Exchange Networks





Lemma 9

We can perform an ASCEND run of $M(2,d)$ on $SE(d)$ in $\mathcal{O}(d)$ steps.

Simulations between Networks

For the following observations we need to make the definition of parallel computer networks more precise.

Each node of a given network corresponds to a processor/RAM.

In addition each processor has a read register and a write register.

In one (synchronous) step each neighbour of a processor P_i can write into P_i's write register or can read from P_i's read register.

Usually we assume that proper care has to be taken to avoid concurrent reads and concurrent writes from/to the same register.

Simulations between Networks

For the following observations we need to make the definition of parallel computer networks more precise.

Each node of a given network corresponds to a processor/RAM.

In addition each processor has a read register and a write register.

In one (synchronous) step each neighbour of a processor P_i can write into P_i's write register or can read from P_i's read register.

Usually we assume that proper care has to be taken to avoid concurrent reads and concurrent writes from/to the same register.

Simulations between Networks

For the following observations we need to make the definition of parallel computer networks more precise.

Each node of a given network corresponds to a processor/RAM.

In addition each processor has a read register and a write register.

In one (synchronous) step each neighbour of a processor P_i can write into P_i's write register or can read from P_i's read register.

Usually we assume that proper care has to be taken to avoid concurrent reads and concurrent writes from/to the same register.

Simulations between Networks

For the following observations we need to make the definition of parallel computer networks more precise.

Each node of a given network corresponds to a processor/RAM.

In addition each processor has a read register and a write register.

In one (synchronous) step each neighbour of a processor P_i can write into P_i's write register or can read from P_i's read register.

Usually we assume that proper care has to be taken to avoid concurrent reads and concurrent writes from/to the same register.

Simulations between Networks

For the following observations we need to make the definition of parallel computer networks more precise.

Each node of a given network corresponds to a processor/RAM.

In addition each processor has a read register and a write register.

In one (synchronous) step each neighbour of a processor P_i can write into P_i's write register or can read from P_i's read register.

Usually we assume that proper care has to be taken to avoid concurrent reads and concurrent writes from/to the same register.

Simulations between Networks

For the following observations we need to make the definition of parallel computer networks more precise.

Each node of a given network corresponds to a processor/RAM.

In addition each processor has a read register and a write register.

In one (synchronous) step each neighbour of a processor P_i can write into P_i's write register or can read from P_i's read register.

Usually we assume that proper care has to be taken to avoid concurrent reads and concurrent writes from/to the same register.

Simulations between Networks

Definition 10
A configuration C_i of processor P_i is the complete description of the state of P_i including local memory, program counter, read-register, write-register, etc.

Suppose a machine M is in configuration $(C_0, \ldots, C_{p-1})$, performs t synchronous steps, and is then in configuration $C = (C'_0, \ldots, C'_{p-1})$.

C'_i is called the t-th successor configuration of C for processor i.

Simulations between Networks

Definition 10

A configuration C_i of processor P_i is the complete description of the state of P_i including local memory, program counter, read-register, write-register, etc.

Suppose a machine M is in configuration $(C_0, \ldots, C_{p-1})$, performs t synchronous steps, and is then in configuration $C = (C_0', \ldots, C_{p-1}')$.

C_i' is called the t-th successor configuration of C for processor i.

Simulations between Networks

Definition 10
A configuration C_i of processor P_i is the complete description of the state of P_i including local memory, program counter, read-register, write-register, etc.

Suppose a machine M is in configuration $(C_0, \dots, C_{p-1})$, performs t synchronous steps, and is then in configuration $C = (C'_0, \dots, C'_{p-1})$.

C'_i is called the t-th successor configuration of C for processor i.

Simulations between Networks

Definition 11
Let $C = (C_0, \ldots, C_{p-1})$ a configuration of M. A machine M' with $q \geq p$ processors weakly simulates t steps of M with slowdown k if

- in the beginning there are p non-empty processors sets $A_0, \ldots, A_{p-1} \subseteq M'$ so that all processors in A_i know C_i;
- after at most $k \cdot t$ steps of M' there is a processor $Q^{(i)}$ that knows the t-th successors configuration of C for processor P_i.

Simulations between Networks

Definition 12

M' simulates M with slowdown k if

- M' weakly simulates machine M with slowdown k
- and **every** processor in A_i knows the t-th successor configuration of C for processor P_i.

We have seen how to simulate an ASCEND/DESCEND run of the hypercube $M(2, d+k)$ on CCC(d) with $d = 2^k$ in $\mathcal{O}(d)$ steps.

Hence, we can simulate $d + k$ steps (one ASCEND run) of the hypercube in $\mathcal{O}(d)$ steps. This means slowdown $\mathcal{O}(1)$.

Lemma 13

Suppose a network S with n processors can route any permutation in time $\mathcal{O}(t(n))$. Then S can simulate any ***constant degree*** *network M with at most n vertices with slowdown $\mathcal{O}(t(n))$.*

Map the vertices of M to vertices of S in an arbitrary way.

Color the edges of M with $\Delta + 1$ colors, where $\Delta = \mathcal{O}(1)$ denotes the maximum degree.

Each color gives rise to a permutation.

We can route this permutation in S in $t(n)$ steps.

Hence, we can perform the required communication for one step of M by routing $\Delta + 1$ permutations in S. This takes time $t(n)$.

A processor of M is simulated by the same processor of S throughout the simulation.

Map the vertices of M to vertices of S in an arbitrary way.

Color the edges of M with $\Delta + 1$ colors, where $\Delta = \mathcal{O}(1)$ denotes the maximum degree.

Each color gives rise to a permutation.

We can route this permutation in S in $t(n)$ steps.

Hence, we can perform the required communication for one step of M by routing $\Delta + 1$ permutations in S. This takes time $t(n)$.

A processor of M is simulated by the same processor of S throughout the simulation.

Map the vertices of M to vertices of S in an arbitrary way.

Color the edges of M with $\Delta + 1$ colors, where $\Delta = \mathcal{O}(1)$ denotes the maximum degree.

Each color gives rise to a permutation.

We can route this permutation in S in $t(n)$ steps.

Hence, we can perform the required communication for one step of M by routing $\Delta + 1$ permutations in S. This takes time $t(n)$.

A processor of M is simulated by the same processor of S throughout the simulation.

Map the vertices of M to vertices of S in an arbitrary way.

Color the edges of M with $\Delta + 1$ colors, where $\Delta = \mathcal{O}(1)$ denotes the maximum degree.

Each color gives rise to a permutation.

We can route this permutation in S in $t(n)$ steps.

Hence, we can perform the required communication for one step of M by routing $\Delta + 1$ permutations in S. This takes time $t(n)$.

A processor of M is simulated by the same processor of S throughout the simulation.

Map the vertices of M to vertices of S in an arbitrary way.

Color the edges of M with $\Delta + 1$ colors, where $\Delta = \mathcal{O}(1)$ denotes the maximum degree.

Each color gives rise to a permutation.

We can route this permutation in S in $t(n)$ steps.

Hence, we can perform the required communication for one step of M by routing $\Delta + 1$ permutations in S. This takes time $t(n)$.

A processor of M is simulated by the same processor of S throughout the simulation.

Map the vertices of M to vertices of S in an arbitrary way.

Color the edges of M with $\Delta + 1$ colors, where $\Delta = \mathcal{O}(1)$ denotes the maximum degree.

Each color gives rise to a permutation.

We can route this permutation in S in $t(n)$ steps.

Hence, we can perform the required communication for one step of M by routing $\Delta + 1$ permutations in S. This takes time $t(n)$.

A processor of M is simulated by the same processor of S throughout the simulation.

Lemma 14

Suppose a network S with n processors can sort n numbers in time $\mathcal{O}(t(n))$. Then S can simulate any network M with at most n vertices with slowdown $\mathcal{O}(t(n))$.

Lemma 15

There is a constant degree network on $\mathcal{O}(n^{1+\epsilon})$ *nodes that can simulate any constant degree network with slowdown* $\mathcal{O}(1)$*.*

Suppose we allow concurrent reads, this means in every step all neighbours of a processor P_i can read P_i's read register.

Lemma 16

A constant degree network M that can simulate any n-node network has slowdown $\mathcal{O}(\log n)$ (independent of the size of M).

Suppose we allow concurrent reads, this means in every step all neighbours of a processor P_i can read P_i's read register.

Lemma 16

A constant degree network M that can simulate any n-node network has slowdown $\mathcal{O}(\log n)$ (independent of the size of M).

We show the lemma for the following type of simulation.

- There are representative sets A_i^t for every step t that specify which processors of M simulate processor P_i in step t (know the configuration of P_i after the t-th step).
- The representative sets for different processors are disjoint.
- for all $i \in \{1, \ldots, n\}$ and steps t, $A_i^t \neq \emptyset$.

This is a step-by-step simulation.

We show the lemma for the following type of simulation.

- There are representative sets A_i^t for every step t that specify which processors of M simulate processor P_i in step t (know the configuration of P_i after the t-th step).
- The representative sets for different processors are disjoint.
- for all $i \in \{1, \dots, n\}$ and steps t, $A_i^t \neq \emptyset$.

This is a step-by-step simulation.

Suppose processor P_i reads from processor P_{j_i} in step t.

Every processor $Q \in M$ with $Q \in A_i^{t+1}$ must have a path to a processor $Q' \in A_i^t$ and to $Q'' \in A_{j_i}^t$.

Let k_t be the largest distance (maximized over all i, j_i).

Then the simulation of step t takes time at least k_t.

The slowdown is at least

$$k = \frac{1}{\ell}\sum_{t=1}^{\ell} k_t$$

Suppose processor P_i reads from processor P_{j_i} in step t.

Every processor $Q \in M$ with $Q \in A_i^{t+1}$ must have a path to a processor $Q' \in A_i^t$ and to $Q'' \in A_{j_i}^t$.

Let k_t be the largest distance (maximized over all i, j_i).

Then the simulation of step t takes time at least k_t.

The slowdown is at least

$$k = \frac{1}{\ell} \sum_{t=1}^{\ell} k_t$$

Suppose processor P_i reads from processor P_{j_i} in step t.

Every processor $Q \in M$ with $Q \in A_i^{t+1}$ must have a path to a processor $Q' \in A_i^t$ and to $Q'' \in A_{j_i}^t$.

Let k_t be the largest distance (maximized over all i, j_i).

Then the simulation of step t takes time at least k_t.

The slowdown is at least

$$k = \frac{1}{\ell} \sum_{t=1}^{\ell} k_t$$

Suppose processor P_i reads from processor P_{j_i} in step t.

Every processor $Q \in M$ with $Q \in A_i^{t+1}$ must have a path to a processor $Q' \in A_i^t$ and to $Q'' \in A_{j_i}^t$.

Let k_t be the largest distance (maximized over all i, j_i).

Then the simulation of step t takes time at least k_t.

The slowdown is at least

$$k = \frac{1}{\ell}\sum_{t=1}^{\ell} k_t$$

Suppose processor P_i reads from processor P_{j_i} in step t.

Every processor $Q \in M$ with $Q \in A_i^{t+1}$ must have a path to a processor $Q' \in A_i^t$ and to $Q'' \in A_{j_i}^t$.

Let k_t be the largest distance (maximized over all i, j_i).

Then the simulation of step t takes time at least k_t.

The slowdown is at least

$$k = \frac{1}{\ell} \sum_{t=1}^{\ell} k_t$$

We show

- The simulation of a step takes at least time $\gamma \log n$, or
- the size of the representative sets shrinks by a lot

$$\sum_i |A_i^{t+1}| \leq \frac{1}{n^\epsilon} \sum_i |A_i^t|$$

We show

- The simulation of a step takes at least time $\gamma \log n$, or
- the size of the representative sets shrinks by a lot

$$\sum_i |A_i^{t+1}| \leq \frac{1}{n^\epsilon} \sum_i |A_i^t|$$

Suppose there is no pair (i, j) such that i reading from j requires time $\gamma \log n$.

Suppose there is no pair (i, j) such that i reading from j requires time $\gamma \log n$.

- For every i the set $\Gamma_{2k}(A_i)$ contains a node from A_j.
- Hence, there must exist a j_i such that $\Gamma_{2k}(A_i)$ contains at most

$$|C_{j_i}| := \frac{|A_i| \cdot c^{2k}}{n-1} \leq \frac{|A_i| \cdot c^{3k}}{n} \enspace .$$

processors from $|A_{j_i}|$

Suppose there is no pair (i, j) such that i reading from j requires time $\gamma \log n$.

- For every i the set $\Gamma_{2k}(A_i)$ contains a node from A_j.
- Hence, there must exist a j_i such that $\Gamma_{2k}(A_i)$ contains at most

  $$|C_{j_i}| := \frac{|A_i| \cdot c^{2k}}{n-1} \leq \frac{|A_i| \cdot c^{3k}}{n} \quad .$$

  processors from $|A_{j_i}|$

If we choose that i reads from j_i we get

$$|A_i'|$$

If we choose that i reads from j_i we get

$$|A_i'| \leq |C_{j_i}| \cdot c^k$$

If we choose that i reads from j_i we get

$$\begin{aligned}|A_i'| &\leq |C_{j_i}| \cdot c^k \\ &\leq c^k \cdot \frac{|A_i| \cdot c^{3k}}{n}\end{aligned}$$

If we choose that i reads from j_i we get

$$\begin{aligned} |A_i'| &\le |C_{j_i}| \cdot c^k \\ &\le c^k \cdot \frac{|A_i| \cdot c^{3k}}{n} \\ &= \frac{1}{n}|A_i| \cdot c^{4k} \end{aligned}$$

If we choose that i reads from j_i we get

$$\begin{aligned}|A_i'| &\leq |C_{j_i}| \cdot c^k \\ &\leq c^k \cdot \frac{|A_i| \cdot c^{3k}}{n} \\ &= \frac{1}{n}|A_i| \cdot c^{4k}\end{aligned}$$

If we choose that i reads from j_i we get

$$\begin{aligned}|A_i'| &\le |C_{j_i}| \cdot c^k \\ &\le c^k \cdot \frac{|A_i| \cdot c^{3k}}{n} \\ &= \frac{1}{n}|A_i| \cdot c^{4k}\end{aligned}$$

Choosing $k = \Theta(\log n)$ gives that this is at most $|A_i|/n^{\epsilon}$.

Let ℓ be the total number of steps and s be the number of short steps when $k_t < \gamma \log n$.

In a step of time k_t a representative set can at most increase by c^{k_t+1}.

Let h_ℓ denote the number of representatives after step ℓ.

Let ℓ be the total number of steps and s be the number of short steps when $k_t < \gamma \log n$.

In a step of time k_t a representative set can at most increase by c^{k_t+1}.

Let h_ℓ denote the number of representatives after step ℓ.

Let ℓ be the total number of steps and s be the number of short steps when $k_t < \gamma \log n$.

In a step of time k_t a representative set can at most increase by c^{k_t+1}.

Let h_ℓ denote the number of representatives after step ℓ.

$$n \le h_\ell \le h_0\Big(\frac{1}{n^\epsilon}\Big)^s \prod_{t\in\text{long}} c^{k_t+1} \le \frac{n}{n^{\epsilon s}} \cdot c^{\ell+\sum_t k_t}$$

If $\sum_t k_t \ge \ell(\frac{\epsilon}{2}\log_c n - 1)$, we are done. Otw.

$$n \le n^{1-\epsilon s+\ell\frac{\epsilon}{2}}$$

This gives $s \le \ell/2$.

Hence, at most 50% of the steps are short.

$$n \leq h_\ell \leq h_0\Big(\frac{1}{n^\epsilon}\Big)^s \prod_{t \in \text{long}} c^{k_t+1} \leq \frac{n}{n^{\epsilon s}} \cdot c^{\ell + \sum_t k_t}$$

If $\sum_t k_t \geq \ell(\frac{\epsilon}{2}\log_c n - 1)$, we are done. Otw.

$$n \leq n^{1-\epsilon s + \ell\frac{\epsilon}{2}}$$

This gives $s \leq \ell/2$.

Hence, at most 50% of the steps are short.

$$n \le h_\ell \le h_0\left(\frac{1}{n^\epsilon}\right)^s \prod_{t\in\text{long}} c^{k_t+1} \le \frac{n}{n^{\epsilon s}} \cdot c^{\ell+\sum_t k_t}$$

If $\sum_t k_t \ge \ell(\frac{\epsilon}{2}\log_c n - 1)$, we are done. Otw.

$$n \le n^{1-\epsilon s+\ell\frac{\epsilon}{2}}$$

This gives $s \le \ell/2$.

Hence, at most 50% of the steps are short.

$$n \le h_\ell \le h_0 \Big(\frac{1}{n^\epsilon}\Big)^s \prod_{t \in \text{long}} c^{k_t+1} \le \frac{n}{n^{\epsilon s}} \cdot c^{\ell + \sum_t k_t}$$

If $\sum_t k_t \ge \ell(\frac{\epsilon}{2}\log_c n - 1)$, we are done. Otw.

$$n \le n^{1-\epsilon s + \ell \frac{\epsilon}{2}}$$

This gives $s \le \ell/2$.

Hence, at most 50% of the steps are short.

Deterministic Online Routing

Lemma 17

A permutation on an $n \times n$-mesh can be routed online in $\mathcal{O}(n)$ steps.

Deterministic Online Routing

Definition 18 (Oblivious Routing)

Specify a path-system $\mathcal{W}$ with a path $P_{u,v}$ between u and v for every pair $\{u,v\} \in V \times V$.

A packet with source u and destination v moves along path $P_{u,v}$.

Deterministic Online Routing

Definition 19 (Oblivious Routing)

Specify a path-system $\mathcal{W}$ with a path $P_{u,v}$ between u and v for every pair $\{u,v\} \in V \times V$.

Definition 20 (node congestion)

For a given path-system the node congestion is the maximum number of path that go through any node $v \in V$.

Definition 21 (edge congestion)

For a given path-system the edge congestion is the maximum number of path that go through any edge $e \in E$.

Deterministic Online Routing

Definition 22 (dilation)

For a given path system the dilation is the maximum length of a path.

Lemma 23
Any oblivious routing protocol requires at least $\max\{C_f, D_f\}$ *steps, where* C_f *and* D_f*, are the congestion and dilation, respectively, of the path-system used. (node congestion or edge congestion depending on the communication model)*

Lemma 24
Any reasonable oblivious routing protocol requires at most $\mathcal{O}(D_f \cdot C_f)$ *steps (unbounded buffers).*

Theorem 25 (Borodin, Hopcroft)

For any path system $\mathcal{W}$ there exists a permutation $\pi : V \to V$ and an edge $e \in E$ such that at least $\Omega(\sqrt{n}/\Delta)$ of the paths go through e.

Let $\mathcal{W}_v = \{P_{v,u} \mid u \in V\}$.

We say that an edge e is z-popular for v if at least z paths from $\mathcal{W}_v$ contain e.

For any node v there are many edges that are are quite popular for v.

$|V| \times |E|$-matrix $A(z)$:

$$A_{v,e}(z) = \begin{cases} 1 & e \text{ is } z\text{-popular for } v \\ 0 & \text{otherwise} \end{cases}$$

Define

- $$A_v(z) = \sum_e A_{v,e}(z)$$
- $$A_e(z) = \sum_v A_{v,e}(z)$$

Lemma 26

Let $z \le \frac{n-1}{\Delta}$.

For every node $v \in V$ *there exist at least* $\frac{n}{2\Delta z}$ *edges that are* z *popular for* v. *This means*

$$A_v(z) \ge \frac{n}{2\Delta z}$$

Lemma 27

There exists an edge e' that is z-popular for at least z nodes with $z = \Omega(\sqrt{n}\Delta)$.

$$\sum_e A_e(z) = \sum_v A_v(z) \geq \frac{n^2}{2\Delta z}$$

There must exist an edge e'

$$A_{e'}(z) \geq \left\lceil \frac{n^2}{|E| \cdot 2\Delta z} \right\rceil \geq \left\lceil \frac{n}{2\Delta^2 z} \right\rceil$$

where the last step follows from $|E| \leq \Delta n$.

We choose z such that $z = \frac{n}{2\Delta^2 z}$ (i.e., $z = \sqrt{n}/(\sqrt{2}\Delta)$).

This means e' is $\lceil z \rceil$-popular for $\lceil z \rceil$ nodes.

We can construct a permutation such that z paths go through e'.

Deterministic oblivious routing may perform very poorly.

What happens if we have a random routing problem in a butterfly?

Suppose every source on level 0 has p packets, that are routed to random destinations.

How many packets go over node v on level i?

From v we can reach $2^d/2^i$ different targets.

Hence,

$$\Pr[\text{packet goes over } v] \le \frac{2^{d-i}}{2^d} = \frac{1}{2^i}$$

Suppose every source on level 0 has p packets, that are routed to random destinations.

How many packets go over node v on level i?

From v we can reach $2^d/2^i$ different targets.

Hence,

$$\Pr[\text{packet goes over } v] \leq \frac{2^{d-i}}{2^d} = \frac{1}{2^i}$$

Suppose every source on level 0 has p packets, that are routed to random destinations.

How many packets go over node v on level i?

From v we can reach $2^d/2^i$ different targets.

Hence,

$$\Pr[\text{packet goes over } v] \leq \frac{2^{d-i}}{2^d} = \frac{1}{2^i}$$

Suppose every source on level 0 has p packets, that are routed to random destinations.

How many packets go over node v on level i?

From v we can reach $2^d/2^i$ different targets.

Hence,

$$\Pr[\text{packet goes over } v] \leq \frac{2^{d-i}}{2^d} = \frac{1}{2^i}$$

Expected number of packets:

$$\mathrm{E}[\text{packets over } v] = p \cdot 2^i \cdot \frac{1}{2^i} = p$$

since only $p2^i$ packets can reach v.

But this is trivial.

Expected number of packets:

$$\mathrm{E}[\text{packets over } v] = p \cdot 2^i \cdot \frac{1}{2^i} = p$$

since only $p2^i$ packets can reach v.

But this is trivial.

What is the probability that at least r packets go through v.

$$\begin{aligned}\Pr[\text{at least } r \text{ path through } v] &\le \binom{p\cdot 2^i}{r}\cdot\left(\frac{1}{2^i}\right)^r\\ &\le \left(\frac{p2^i\cdot e}{r}\right)^r\cdot\left(\frac{1}{2^i}\right)\\ &= \left(\frac{pe}{r}\right)^r\end{aligned}$$

$$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v]$$

$$\le d2^d\cdot\left(\frac{pe}{r}\right)^r$$

What is the probability that at least r packets go through v.

$$\Pr[\text{at least } r \text{ path through } v] \le \binom{p \cdot 2^i}{r} \cdot \left(\frac{1}{2^i}\right)^r$$

$$\le \left(\frac{p2^i \cdot e}{r}\right)^r \cdot \left(\frac{1}{2^i}\right)$$

$$= \left(\frac{pe}{r}\right)^r$$

$$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v]$$

$$\le d2^d \cdot \left(\frac{pe}{r}\right)^r$$

What is the probability that at least r packets go through v.

$$\Pr[\text{at least } r \text{ path through } v] \leq \binom{p \cdot 2^i}{r} \cdot \left(\frac{1}{2^i}\right)^r$$

$$\leq \left(\frac{p2^i \cdot e}{r}\right)^r \cdot \left(\frac{1}{2^i}\right)$$

$$= \left(\frac{pe}{r}\right)^r$$

$$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v]$$

$$\leq d2^d \cdot \left(\frac{pe}{r}\right)^r$$

What is the probability that at least r packets go through v.

$$\begin{aligned}\Pr[\text{at least } r \text{ path through } v] &\le \binom{p\cdot 2^i}{r}\cdot\left(\frac{1}{2^i}\right)^r\\ &\le \left(\frac{p2^i\cdot e}{r}\right)^r\cdot\left(\frac{1}{2^i}\right)\\ &= \left(\frac{pe}{r}\right)^r\end{aligned}$$

$$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v]$$

$$\le d2^d\cdot\left(\frac{pe}{r}\right)^r$$

What is the probability that at least r packets go through v.

$$\begin{aligned}
\Pr[\text{at least } r \text{ path through } v] &\le \binom{p\cdot 2^i}{r}\cdot\left(\frac{1}{2^i}\right)^r \\
&\le \left(\frac{p2^i\cdot e}{r}\right)^r\cdot\left(\frac{1}{2^i}\right) \\
&= \left(\frac{pe}{r}\right)^r
\end{aligned}$$

$$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v]$$

$$\le d2^d\cdot\left(\frac{pe}{r}\right)^r$$

What is the probability that at least r packets go through v.

$$\Pr[\text{at least } r \text{ path through } v] \le \binom{p \cdot 2^i}{r} \cdot \left(\frac{1}{2^i}\right)^r$$
$$\le \left(\frac{p2^i \cdot e}{r}\right)^r \cdot \left(\frac{1}{2^i}\right)$$
$$= \left(\frac{pe}{r}\right)^r$$

$$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v]$$
$$\le d2^d \cdot \left(\frac{pe}{r}\right)^r$$

$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v]$

$$\leq d2^d \cdot \left(\frac{pe}{r}\right)^r$$

Choose r as $2ep + (\ell + 1)d + \log d = \mathcal{O}(p + \log N)$, where N is number of sources in BF(d).

$$\Pr[\text{exists node } v \text{ with more than } r \text{ paths over } v] \leq \frac{1}{N^\ell}$$

$$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v] \le d2^d \cdot \left(\frac{pe}{r}\right)^r$$

Choose r as $2ep + (\ell + 1)d + \log d = \mathcal{O}(p + \log N)$, where N is number of sources in BF(d).

$$\Pr[\text{exists node } v \text{ with more than } r \text{ paths over } v] \le \frac{1}{N^\ell}$$

$$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v]$$
$$\leq d2^d \cdot \left(\frac{pe}{r}\right)^r$$

Choose r as $2ep + (\ell + 1)d + \log d = \mathcal{O}(p + \log N)$, where N is number of sources in BF(d).

$$\Pr[\text{exists node } v \text{ with more than } r \text{ paths over } v] \leq \frac{1}{N^\ell}$$

$$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v] \le d2^d \cdot \left(\frac{pe}{r}\right)^r$$

Choose r as $2ep + (\ell + 1)d + \log d = \mathcal{O}(p + \log N)$, where N is number of sources in $\mathrm{BF}(d)$.

$$\Pr[\text{exists node } v \text{ with more than } r \text{ paths over } v] \le \frac{1}{N^\ell}$$

$$\Pr[\text{there exists a node } v \text{ sucht that at least } r \text{ path through } v] \le d2^d \cdot \left(\frac{pe}{r}\right)^r$$

Choose r as $2ep + (\ell + 1)d + \log d = \mathcal{O}(p + \log N)$, where N is number of sources in BF(d).

$$\Pr[\text{exists node } v \text{ with more than } r \text{ paths over } v] \le \frac{1}{N^\ell}$$

Scheduling Packets

Assume that in every round a node may forward at most one packet but may receive up to two.

We select a random rank $R_p \in [k]$. Whenever, we forward a packet we choose the packet with smaller rank. Ties are broken according to packet id.

Random Rank Protocol

Scheduling Packets

Assume that in every round a node may forward at most one packet but may receive up to two.

We select a random rank $R_p \in [k]$. Whenever, we forward a packet we choose the packet with smaller rank. Ties are broken according to packet id.

Random Rank Protocol

Scheduling Packets

Assume that in every round a node may forward at most one packet but may receive up to two.

We select a random rank $R_p \in [k]$. Whenever, we forward a packet we choose the packet with smaller rank. Ties are broken according to packet id.

Random Rank Protocol

Scheduling Packets

Assume that in every round a node may forward at most one packet but may receive up to two.

We select a random rank $R_p \in [k]$. Whenever, we forward a packet we choose the packet with smaller rank. Ties are broken according to packet id.

Random Rank Protocol

Definition 28 (Delay Sequence of length s)

- delay path $\mathcal{W}$
- lengths $\ell_0, \ell_1, \ldots, \ell_s$, with $\ell_0 \geq 1, \ell_1, \ldots, \ell_s \geq 0$ lengths of delay-free sub-paths
- collision nodes $v_0, v_1, \ldots, v_s, v_{s+1}$
- collision packets $P_0, \ldots, P_s$

Definition 28 (Delay Sequence of length s)

- delay path $\mathcal{W}$
- lengths $\ell_0, \ell_1, \dots, \ell_s$, with $\ell_0 \geq 1$, $\ell_1, \dots, \ell_s \geq 0$ lengths of delay-free sub-paths
- collision nodes $v_0, v_1, \dots, v_s, v_{s+1}$
- collision packets $P_0, \dots, P_s$

Definition 28 (Delay Sequence of length s)

- delay path $\mathcal{W}$
- lengths $\ell_0, \ell_1, \ldots, \ell_s$, with $\ell_0 \geq 1$, $\ell_1, \ldots, \ell_s \geq 0$ lengths of delay-free sub-paths
- collision nodes $v_0, v_1, \ldots, v_s, v_{s+1}$
- collision packets $P_0, \ldots, P_s$

Definition 28 (Delay Sequence of length s)

- delay path $\mathcal{W}$
- lengths $\ell_0, \ell_1, \dots, \ell_s$, with $\ell_0 \geq 1$, $\ell_1, \dots, \ell_s \geq 0$ lengths of delay-free sub-paths
- collision nodes $v_0, v_1, \dots, v_s, v_{s+1}$
- collision packets $P_0, \dots, P_s$

Properties

- $\text{rank}(P_0) \geq \text{rank}(P_1) \geq \cdots \geq \text{rank}(P_s)$
- $\sum_{i=0}^{s} \ell_i = d$
- if the routing takes $d + s$ steps than the delay sequence has length s

Properties

- $\mathrm{rank}(P_0) \geq \mathrm{rank}(P_1) \geq \cdots \geq \mathrm{rank}(P_s)$
- $\sum_{i=0}^{s} \ell_i = d$
- if the routing takes $d + s$ steps than the delay sequence has length s

Properties

- $\text{rank}(P_0) \geq \text{rank}(P_1) \geq \cdots \geq \text{rank}(P_s)$
- $\sum_{i=0}^{s} \ell_i = d$
- if the routing takes $d + s$ steps than the delay sequence has length s

Definition 29 (Formal Delay Sequence)

- a path $\mathcal{W}$ of length d from a source to a target
- s integers $\ell_0 \geq 1, \ell_1, \ldots, \ell_s \geq 0$ and $\sum_{i=0}^{s} \ell_i = d$
- nodes $v_0, \ldots v_s, v_{s+1}$ on $\mathcal{W}$ with v_i being on level $d - \ell_0 - \cdots - \ell_{i-1}$
- $s + 1$ packets $P_0, \ldots, P_s$, where P_i is a packet with path through v_i and v_{i-1}
- numbers $R_s \leq R_{s-1} \leq \cdots \leq R_0$

Definition 29 (Formal Delay Sequence)

- a path $\mathcal{W}$ of length d from a source to a target
- s integers $\ell_0 \geq 1, \ell_1, \ldots, \ell_s \geq 0$ and $\sum_{i=0}^{s} \ell_i = d$
- nodes $v_0, \ldots v_s, v_{s+1}$ on $\mathcal{W}$ with v_i being on level $d - \ell_0 - \cdots - \ell_{i-1}$
- $s + 1$ packets $P_0, \ldots, P_s$, where P_i is a packet with path through v_i and v_{i-1}
- numbers $R_s \leq R_{s-1} \leq \cdots \leq R_0$

Definition 29 (Formal Delay Sequence)

- a path $\mathcal{W}$ of length d from a source to a target
- s integers $\ell_0 \geq 1, \ell_1, \ldots, \ell_s \geq 0$ and $\sum_{i=0}^{s} \ell_i = d$
- nodes $v_0, \ldots v_s, v_{s+1}$ on $\mathcal{W}$ with v_i being on level $d - \ell_0 - \cdots - \ell_{i-1}$
- $s+1$ packets $P_0, \ldots, P_s$, where P_i is a packet with path through v_i and v_{i-1}
- numbers $R_s \leq R_{s-1} \leq \cdots \leq R_0$

Definition 29 (Formal Delay Sequence)

- a path $\mathcal{W}$ of length d from a source to a target
- s integers $\ell_0 \geq 1, \ell_1, \ldots, \ell_s \geq 0$ and $\sum_{i=0}^{s} \ell_i = d$
- nodes $v_0, \ldots v_s, v_{s+1}$ on $\mathcal{W}$ with v_i being on level $d - \ell_0 - \cdots - \ell_{i-1}$
- $s + 1$ packets $P_0, \ldots, P_s$, where P_i is a packet with path through v_i and v_{i-1}
- numbers $R_s \leq R_{s-1} \leq \cdots \leq R_0$

Definition 29 (Formal Delay Sequence)

- a path $\mathcal{W}$ of length d from a source to a target
- s integers $\ell_0 \geq 1, \ell_1, \ldots, \ell_s \geq 0$ and $\sum_{i=0}^{s} \ell_i = d$
- nodes $v_0, \ldots v_s, v_{s+1}$ on $\mathcal{W}$ with v_i being on level $d - \ell_0 - \cdots - \ell_{i-1}$
- $s+1$ packets $P_0, \ldots, P_s$, where P_i is a packet with path through v_i and v_{i-1}
- numbers $R_s \leq R_{s-1} \leq \cdots \leq R_0$

We say a formal delay sequence is active if $\mathrm{rank}(P_i) = k_i$ holds for all i.

Let N_s be the number of formal delay sequences of length at most s. Then

$$\Pr[\text{routing needs at least } d + s \text{ steps}] \le \frac{N_s}{k^{s+1}}$$

We say a formal delay sequence is active if $\text{rank}(P_i) = k_i$ holds for all i.

Let N_s be the number of formal delay sequences of length at most s. Then

$$\Pr[\text{routing needs at least } d + s \text{ steps}] \leq \frac{N_s}{k^{s+1}}$$

Lemma 30

$$N_s \leq \left(\frac{2eC(s+k)}{s+1}\right)^{s+1}$$

- there are N^2 ways to choose $\mathcal{W}$
- there are $\binom{s+d-1}{s}$ ways to choose ℓ_i's with $\sum_{i=0}^{s} \ell_i = d$
- the collision nodes are fixed
- there are at most C^{s+1} ways to choose the collision packets where C is the node congestion
- there are at most $\binom{s+k}{s+1}$ ways to choose $0 \leq k_s \leq \cdots \leq k_0 < k$

Lemma 30

$$N_s \leq \left(\frac{2eC(s+k)}{s+1}\right)^{s+1}$$

- there are N^2 ways to choose $\mathcal{W}$
- there are $\binom{s+d-1}{s}$ ways to choose ℓ_i's with $\sum_{i=0}^{s} \ell_i = d$
- the collision nodes are fixed
- there are at most C^{s+1} ways to choose the collision packets where C is the node congestion
- there are at most $\binom{s+k}{s+1}$ ways to choose $0 \leq k_s \leq \cdots \leq k_0 < k$

Lemma 30

$$N_s \leq \left(\frac{2eC(s+k)}{s+1}\right)^{s+1}$$

- there are N^2 ways to choose $\mathcal{W}$
- there are $\binom{s+d-1}{s}$ ways to choose ℓ_i's with $\sum_{i=0}^{s} \ell_i = d$
- the collision nodes are fixed
- there are at most C^{s+1} ways to choose the collision packets where C is the node congestion
- there are at most $\binom{s+k}{s+1}$ ways to choose $0 \leq k_s \leq \cdots \leq k_0 < k$

Lemma 30

$$N_s \leq \left(\frac{2eC(s+k)}{s+1}\right)^{s+1}$$

- there are N^2 ways to choose $\mathcal{W}$
- there are $\binom{s+d-1}{s}$ ways to choose ℓ_i's with $\sum_{i=0}^{s} \ell_i = d$
- the collision nodes are fixed
- there are at most C^{s+1} ways to choose the collision packets where C is the node congestion
- there are at most $\binom{s+k}{s+1}$ ways to choose $0 \leq k_s \leq \cdots \leq k_0 < k$

Lemma 30

$$N_s \leq \left(\frac{2eC(s+k)}{s+1}\right)^{s+1}$$

- there are N^2 ways to choose $\mathcal{W}$
- there are $\binom{s+d-1}{s}$ ways to choose ℓ_i's with $\sum_{i=0}^{s} \ell_i = d$
- the collision nodes are fixed
- there are at most C^{s+1} ways to choose the collision packets where C is the node congestion
- there are at most $\binom{s+k}{s+1}$ ways to choose $0 \leq k_s \leq \cdots \leq k_0 < k$

Lemma 30

$$N_s \leq \left(\frac{2eC(s+k)}{s+1}\right)^{s+1}$$

- there are N^2 ways to choose $\mathcal{W}$
- there are $\binom{s+d-1}{s}$ ways to choose ℓ_i's with $\sum_{i=0}^{s} \ell_i = d$
- the collision nodes are fixed
- there are at most C^{s+1} ways to choose the collision packets where C is the node congestion
- there are at most $\binom{s+k}{s+1}$ ways to choose $0 \leq k_s \leq \cdots \leq k_0 < k$

Hence the probability that the routing takes more than $d+s$ steps is at most

$$N^3 \cdot \left(\frac{2e \cdot C \cdot (s+k)}{(s+1)k}\right)^{s+1}$$

We choose $s = 8eC - 1 + (\ell + 3)d$ and $k = s + 1$. This gives that the probability is at most $\frac{1}{N^\ell}$.

Hence the probability that the routing takes more than $d + s$ steps is at most

$$N^3 \cdot \left(\frac{2e \cdot C \cdot (s+k)}{(s+1)k}\right)^{s+1}$$

We choose $s = 8eC - 1 + (\ell + 3)d$ and $k = s + 1$. This gives that the probability is at most $\frac{1}{N^\ell}$.

- With probability $1 - \frac{1}{N^{\ell_1}}$ the random routing problem has congestion at most $\mathcal{O}(p + \ell_1 d)$.
- With probability $1 - \frac{1}{N^{\ell_2}}$ the packet scheduling finishes in at most $\mathcal{O}(C + \ell_2 d)$ steps.

Hence, with high probability routing random problems with p packets per source in a butterfly requires only $\mathcal{O}(p + d)$ steps.

What do we do for arbitrary routing problems?

- With probability $1 - \frac{1}{N^{\ell_1}}$ the random routing problem has congestion at most $\mathcal{O}(p + \ell_1 d)$.
- With probability $1 - \frac{1}{N^{\ell_2}}$ the packet scheduling finishes in at most $\mathcal{O}(C + \ell_2 d)$ steps.

Hence, with high probability routing random problems with p packets per source in a butterfly requires only $\mathcal{O}(p + d)$ steps.

What do we do for arbitrary routing problems?

- With probability $1 - \frac{1}{N^{\ell_1}}$ the random routing problem has congestion at most $\mathcal{O}(p + \ell_1 d)$.
- With probability $1 - \frac{1}{N^{\ell_2}}$ the packet scheduling finishes in at most $\mathcal{O}(C + \ell_2 d)$ steps.

Hence, with high probability routing random problems with p packets per source in a butterfly requires only $\mathcal{O}(p + d)$ steps.

What do we do for arbitrary routing problems?

- With probability $1 - \frac{1}{N^{\ell_1}}$ the random routing problem has congestion at most $\mathcal{O}(p + \ell_1 d)$.
- With probability $1 - \frac{1}{N^{\ell_2}}$ the packet scheduling finishes in at most $\mathcal{O}(C + \ell_2 d)$ steps.

Hence, with high probability routing random problems with p packets per source in a butterfly requires only $\mathcal{O}(p + d)$ steps.

What do we do for arbitrary routing problems?

- With probability $1 - \frac{1}{N^{\ell_1}}$ the random routing problem has congestion at most $\mathcal{O}(p + \ell_1 d)$.
- With probability $1 - \frac{1}{N^{\ell_2}}$ the packet scheduling finishes in at most $\mathcal{O}(C + \ell_2 d)$ steps.

Hence, with high probability routing random problems with p packets per source in a butterfly requires only $\mathcal{O}(p + d)$ steps.

What do we do for arbitrary routing problems?

Valiants Trick

Where did the scheduling analysis use the butterfly?

We only used

- all routing paths are of the same length d
- there are a polynomial number of delay paths

Choose paths as follows:

- route from source to random destination on target level
- route to real target column (albeit on source level)
- route to target

All phases run in time $\mathcal{O}(p + d)$ with high probability.

Valiants Trick

Where did the scheduling analysis use the butterfly?

We only used

- all routing paths are of the same length d
- there are a polynomial number of delay paths

Choose paths as follows:

- route from source to random destination on target level
- route to real target column (albeit on source level)
- route to target

All phases run in time $\mathcal{O}(p + d)$ with high probability.

Valiants Trick

Where did the scheduling analysis use the butterfly?

We only used

- all routing paths are of the same length d
- there are a polynomial number of delay paths

Choose paths as follows:

- route from source to random destination on target level
- route to real target column (albeit on source level)
- route to target

All phases run in time $\mathcal{O}(p + d)$ with high probability.

Valiants Trick

Where did the scheduling analysis use the butterfly?

We only used

- all routing paths are of the same length d
- there are a polynomial number of delay paths

Choose paths as follows:

- route from source to random destination on target level
- route to real target column (albeit on source level)
- route to target

All phases run in time $\mathcal{O}(p + d)$ with high probability.

Valiants Trick

Multicommodity Flow Problem

- undirected (weighted) graph $G = (V, E, c)$
- commodities (s_i, t_i), $i \in \{1, \dots, k\}$
- a multicommodity flow is a flow $f : E \times \{1, \dots, k\} \to \mathbb{R}^+$
  - for all edges $e \in E$: $\sum_i f_i(e) \le c(e)$
  - for all nodes $v \in V \setminus \{s_i, t_i\}$: $\sum_{u:(u,v) \in E} f_i((u,v)) = \sum_{w:(v,w) \in E} f_i((v,w))$

Goal A (Maximum Multicommodity Flow)
maximize $\sum_i \sum_{e=(s_i,x) \in E} f_i(e)$

Goal B (Maximum Concurrent Multicommodity Flow)
maximize $\min_i \sum_{e=(s_i,x) \in E} f_i(e) / d_i$ (throughput fraction), where d_i is demand for commodity i

Valiants Trick

Multicommodity Flow Problem

- undirected (weighted) graph $G = (V, E, c)$
- commodities (s_i, t_i), $i \in \{1, \ldots, k\}$
- a multicommodity flow is a flow $f : E \times \{1, \ldots, k\} \to \mathbb{R}^+$
  - for all edges $e \in E$: $\sum_i f_i(e) \le c(e)$
  - for all nodes $v \in V \setminus \{s_i, t_i\}$: $\sum_{u:(u,v)\in E} f_i(u,v) = \sum_{w:(v,w)\in E} f_i(v,w)$

Goal A (Maximum Multicommodity Flow)
maximize $\sum_i \sum_{e=(s_i,x)\in E} f_i(e)$

Goal B (Maximum Concurrent Multicommodity Flow)
maximize $\min_i \sum_{e=(s_i,x)\in E} f_i(e)/d_i$ (throughput fraction), where d_i is demand for commodity i

Valiants Trick

Multicommodity Flow Problem

- undirected (weighted) graph $G = (V, E, c)$
- commodities (s_i, t_i), $i \in \{1, \ldots, k\}$
- a multicommodity flow is a flow $f : E \times \{1, \ldots, k\} \to \mathbb{R}^+$
  - for all edges $e \in E$: $\sum_i f_i(e) \leq c(e)$
  - for all nodes $v \in V \setminus \{s_i, t_i\}$: $\sum_{u:(u,v)\in E} f_i((u,v)) = \sum_{w:(v,w)\in E} f_i((v,w))$

Goal A (Maximum Multicommodity Flow)
maximize $\sum_i \sum_{e=(s_i,x)\in E} f_i(e)$

Goal B (Maximum Concurrent Multicommodity Flow)
maximize $\min_i \sum_{e=(s_i,x)\in E} f_i(e)/d_i$ (throughput fraction), where d_i is demand for commodity i

Valiants Trick

Multicommodity Flow Problem

- undirected (weighted) graph $G = (V, E, c)$
- commodities (s_i, t_i), $i \in \{1, \dots, k\}$
- a multicommodity flow is a flow $f : E \times \{1, \dots, k\} \to \mathbb{R}^+$
  - for all edges $e \in E$: $\sum_i f_i(e) \le c(e)$
  - for all nodes $v \in V \setminus \{s_i, t_i\}$: $\sum_{u:(u,v)\in E} f_i((u,v)) = \sum_{w:(v,w)\in E} f_i((v,w))$

Goal A (Maximum Multicommodity Flow)
maximize $\sum_i \sum_{e=(s_i,x)\in E} f_i(e)$

Goal B (Maximum Concurrent Multicommodity Flow)
maximize $\min_i \sum_{e=(s_i,x)\in E} f_i(e)/d_i$ (throughput fraction), where d_i is demand for commodity i

Valiants Trick

Multicommodity Flow Problem

- undirected (weighted) graph $G = (V, E, c)$
- commodities (s_i, t_i), $i \in \{1, \dots, k\}$
- a multicommodity flow is a flow $f : E \times \{1, \dots, k\} \to \mathbb{R}^+$
  - for all edges $e \in E$: $\sum_i f_i(e) \le c(e)$
  - for all nodes $v \in V \setminus \{s_i, t_i\}$: $\sum_{u:(u,v)\in E} f_i((u,v)) = \sum_{w:(v,w)\in E} f_i((v,w))$

Goal A (Maximum Multicommodity Flow)
maximize $\sum_i \sum_{e=(s_i,x)\in E} f_i(e)$

Goal B (Maximum Concurrent Multicommodity Flow)
maximize $\min_i \sum_{e=(s_i,x)\in E} f_i(e)/d_i$ (throughput fraction), where d_i is demand for commodity i

Valiants Trick

Multicommodity Flow Problem

- undirected (weighted) graph $G = (V, E, c)$
- commodities (s_i, t_i), $i \in \{1, \dots, k\}$
- a multicommodity flow is a flow $f : E \times \{1, \dots, k\} \to \mathbb{R}^+$
  - for all edges $e \in E$: $\sum_i f_i(e) \le c(e)$
  - for all nodes $v \in V \setminus \{s_i, t_i\}$: $\sum_{u:(u,v)\in E} f_i((u,v)) = \sum_{w:(v,w)\in E} f_i((v,w))$

Goal A (Maximum Multicommodity Flow)
maximize $\sum_i \sum_{e=(s_i,x)\in E} f_i(e)$

Goal B (Maximum Concurrent Multicommodity Flow)
maximize $\min_i \sum_{e=(s_i,x)\in E} f_i(e)/d_i$ (throughput fraction), where d_i is demand for commodity i

Valiants Trick

Multicommodity Flow Problem

- undirected (weighted) graph $G = (V, E, c)$
- commodities (s_i, t_i), $i \in \{1, \dots, k\}$
- a multicommodity flow is a flow $f : E \times \{1, \dots, k\} \to \mathbb{R}^+$
  - for all edges $e \in E$: $\sum_i f_i(e) \le c(e)$
  - for all nodes $v \in V \setminus \{s_i, t_i\}$:
    $\sum_{u:(u,v)\in E} f_i((u,v)) = \sum_{w:(v,w)\in E} f_i((v,w))$

Goal A (Maximum Multicommodity Flow)
maximize $\sum_i \sum_{e=(s_i,x)\in E} f_i(e)$

Goal B (Maximum Concurrent Multicommodity Flow)
maximize $\min_i \sum_{e=(s_i,x)\in E} f_i(e)/d_i$ (throughput fraction), where d_i is demand for commodity i

Valiants Trick

Multicommodity Flow Problem

- undirected (weighted) graph $G = (V, E, c)$
- commodities (s_i, t_i), $i \in \{1, \dots, k\}$
- a multicommodity flow is a flow $f : E \times \{1, \dots, k\} \to \mathbb{R}^+$
  - for all edges $e \in E$: $\sum_i f_i(e) \le c(e)$
  - for all nodes $v \in V \setminus \{s_i, t_i\}$: $\sum_{u:(u,v)\in E} f_i((u,v)) = \sum_{w:(v,w)\in E} f_i((v,w))$

Goal A (Maximum Multicommodity Flow)
maximize $\sum_i \sum_{e=(s_i,x)\in E} f_i(e)$

Goal B (Maximum Concurrent Multicommodity Flow)
maximize $\min_i \sum_{e=(s_i,x)\in E} f_i(e)/d_i$ (throughput fraction), where d_i is demand for commodity i

Valiants Trick

A Balanced Multicommodity Flow Problem is a concurrent multicommodity flow problem in which incoming and outgoing flow is equal to

$$c(v) = \sum_{e=(v,x)\in E} c(e)$$

Valiants Trick

For a multicommodity flow S we assume that we have a decomposition of the flow(s) into flow-paths.

We use $C(S)$ to denote the congestion of the flow problem (inverse of througput fraction), and $D(S)$ the length of the longest routing path.

Valiants Trick

For a multicommodity flow S we assume that we have a decomposition of the flow(s) into flow-paths.

We use $C(S)$ to denote the congestion of the flow problem (inverse of througput fraction), and $D(S)$ the length of the longest routing path.

For a network $G = (V, E, c)$ we define the characteristic flow problem via

- demands $d_{u,v} = \frac{c(u)c(v)}{c(V)}$

Suppose the characteristic flow problem has a solution S with $C(S) \leq F$ and $D(S) \leq F$.

For a network $G = (V, E, c)$ we define the characteristic flow problem via

- demands $d_{u,v} = \frac{c(u)c(v)}{c(V)}$

Suppose the characteristic flow problem has a solution S with $C(S) \leq F$ and $D(S) \leq F$.

Definition 31
A (randomized) oblivious routing scheme is given by a path system $\mathcal{P}$ and a weight function w such that

$$\sum_{p\in\mathcal{P}_{s,t}} w(p) = 1$$

Construct an oblivious routing scheme from S as follows:

- let $f_{x,y}$ be the flow between x and y in S
- $$f_{x,y} \geq d_{x,y}/C(S) \geq d_{x,y}/F = \frac{1}{F}\frac{c(x)c(y)}{c(V)}$$
- for $p \in \mathcal{P}_{x,y}$ set $w(p) = f_p/f_{x,y}$

gives an oblivious routing scheme.

Valiants Trick

We apply this routing scheme twice:

- first choose a path from $\mathcal{P}_{s,v}$, where v is chosen uniformly according to $c(v)/c(V)$
- then choose path according to $\mathcal{P}_{v,t}$

If the input flow problem/packet routing problem is balanced doing this randomization results in flow solution S (twice).

Hence, we have an oblivious scheme with congestion and dilation at most $2F$ for (balanced inputs).

Valiants Trick

We apply this routing scheme twice:

- first choose a path from $\mathcal{P}_{s,v}$, where v is chosen uniformly according to $c(v)/c(V)$
- then choose path according to $\mathcal{P}_{v,t}$

If the input flow problem/packet routing problem is balanced doing this randomization results in flow solution S (twice).

Hence, we have an oblivious scheme with congestion and dilation at most $2F$ for (balanced inputs).

Example: hypercube.

Oblivious Routing for the Mesh

We can route any permutation on an $n \times n$ mesh in $\mathcal{O}(n)$ steps, by x-y routing. Actually $\mathcal{O}(d)$ steps where d is the largest distance between a source-target pair.

What happens if we do not have a permutation?

$x - y$ routing may generate large congestion if some pairs have a lot of packets.

Valiants trick may create a large dilation.

Oblivious Routing for the Mesh

We can route any permutation on an $n \times n$ mesh in $\mathcal{O}(n)$ steps, by x-y routing. Actually $\mathcal{O}(d)$ steps where d is the largest distance between a source-target pair.

What happens if we do not have a permutation?

$x - y$ routing may generate large congestion if some pairs have a lot of packets.

Valiants trick may create a large dilation.

Oblivious Routing for the Mesh

We can route any permutation on an $n \times n$ mesh in $\mathcal{O}(n)$ steps, by x-y routing. Actually $\mathcal{O}(d)$ steps where d is the largest distance between a source-target pair.

What happens if we do not have a permutation?

$x - y$ routing may generate large congestion if some pairs have a lot of packets.

Valiants trick may create a large dilation.

Oblivious Routing for the Mesh

We can route any permutation on an $n \times n$ mesh in $\mathcal{O}(n)$ steps, by x-y routing. Actually $\mathcal{O}(d)$ steps where d is the largest distance between a source-target pair.

What happens if we do not have a permutation?

$x - y$ routing may generate large congestion if some pairs have a lot of packets.

Valiants trick may create a large dilation.

Let for a multicommodity flow problem P $C_{\text{opt}}(P)$ be the optimum congestion, and $D_{\text{opt}}(P)$ be the optimum dilation (by perhaps different flow solutions).

Lemma 32

There is an oblivious routing scheme for the mesh that obtains a flow solution S with $C(S) = \mathcal{O}(C_{\text{opt}}(P) \log n)$ and $D(S) = \mathcal{O}(D_{\text{opt}}(P))$.

Let for a multicommodity flow problem P $C_{\text{opt}}(P)$ be the optimum congestion, and $D_{\text{opt}}(P)$ be the optimum dilation (by perhaps different flow solutions).

Lemma 32

There is an oblivious routing scheme for the mesh that obtains a flow solution S with $C(S) = \mathcal{O}(C_{\text{opt}}(P) \log n)$ and $D(S) = \mathcal{O}(D_{\text{opt}}(P))$.

Lemma 33

For any oblivious routing scheme on the mesh there is a demand P such that routing P will give congestion $\Omega(\log n \cdot C_{\text{opt}})$.