Online Routing on the Mesh and Offline Routing on the Benes Network

M. Protsenko

Ferienakademie im Sarntal 2008
FAU Erlangen-Nürnberg, TU München, Uni Stuttgart

September 2008

Overview

Overview

1. Introduction
   - Parallel computation models
   - Notation and definitions

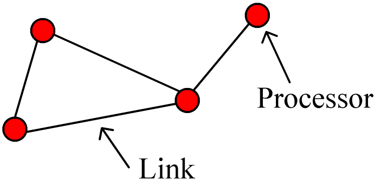
2. Permutation routing on the mesh networks
   - Online routing on linear array
   - Online routing on 2D array

3. Permutation networks
   - Congestion in butterfly network
   - Offline routing in beneš network

Parallel machine

A **parallel machine**:

- a set of processors $P = \{P_0, ..., P_{n-1}\}$
- a **communication graph** $G = (P, E)$



Overview

1. Introduction
   - Parallel computation models
   - Notation and definitions
2. Permutation routing on the mesh networks
   - Online routing on linear array
   - Online routing on 2D array
3. Permutation networks
   - Congestion in butterfly network
   - Offline routing in beneš network

Notation

- $[n] := \{0, ..., n-1\}$
- $bin_d(n)$: binary representation of n using d bits
  E. g., $bin_4(3) = 0011$
- $(\overline{a})_n = k$ for $\overline{a} \in [n]^d$: base-n representation of k
  E. g., $(201)_3 = 19$

Hamming distance

Let $\overline{a} = (a_{d-1}, ..., a_0)$, $\overline{b} = (b_{d-1}, ..., b_0) \in [n]^d$

- The **Hamming distance** between $\overline{a}$ and $\overline{b}$:

$$Hamming(\overline{a}, \overline{b}) := \sum_{i=0}^{d-1} |a_i - b_i|$$

- **Example:**

  $\overline{a} = 01101$
  $\overline{b} = 10111$
  $hamming(\overline{a}, \overline{b}) = 3$

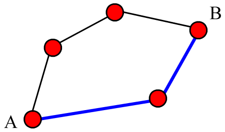
Shortest path, diameter

$G = (V, E)$ - graph and $x, y \in V$ then

- the **shortest path** $dist_G(x, y)$:
  the **minimum** number of edges in path between x and y;
- the **diameter** of G:

$$diam(G) = max\{dist_G(x, y) | x, y \in V\}$$

- **Example:** $dist_G(A, B) = 2$, $diam(G) = 2$



Routing

We have:

- a **network**

$$M = (P, E), P = [N]$$

- a **function**

$$f : [N] \times [p] \to [N]$$

- **messages**

$$x_{0,0}, \dots, x_{0,p-1}, x_{1,0}, \dots, x_{1,p-1}, \dots, x_{N-1,0}, \dots, x_{N-1,p-1}$$

Function routing

- M routs $x_{0,0}, ..., x_{N-1,p-1}$ according to f if:
  - in the beginning processor i stores the package $(i, f(i,k), x_{i,k})$
  - in the end processor f(i,k) stores a copy of a message $x_{i,k}$
- If p=1:
  f is a permutation on $[N] \Rightarrow$ permutation routing

Routing protocol

A **synchronous routing protocol** for M consists of protocols for all processors *i*. Each processor *i* has a **buffer** to store packages.
A **routing step** for every processor i:

- *i* chooses one package from his buffer
- *i* selects one of his neighbors *j*
- *i* sends the package to *j*
- All processors start the t-th routing step **simultaneously**

Routing time & buffer size

- The **routing time** is the number of routing steps
- The **buffer size** is the maximum amount of packages that can be stored in a buffer at the same time

Definitions

- **Routing with preprocessing** (also **off-line routing**): routing protocol depends on f
  At the beginning of routing some computation is performed depending on f to generate protocols for processors i
- **routing without preprocessing** (**on-line routing**): the routing protocol is independent from f

Mesh networks

The n-dimensional **mesh** with edge length n, **M(n,d)**:

- Set of processors $P = \{\overline{a} | \overline{a} \in [n]^d\}$
- Communication graph $G = (P, E)$,

$$E = \{\{\overline{a}, \overline{b}\} | \overline{a}, \overline{b} \in [n]^d, hamming(\overline{a}, \overline{b}) = 1\}$$

Edge in **Dimension i**:
the edge connecting two vertices $\overline{a}$ and $\overline{b}$ with $|a_i - b_i| = 1$

Mesh networks: examples

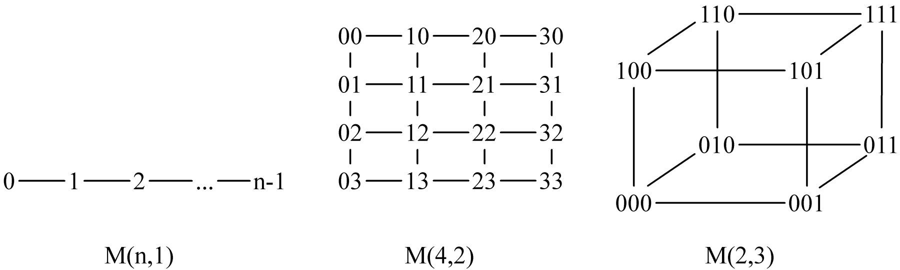
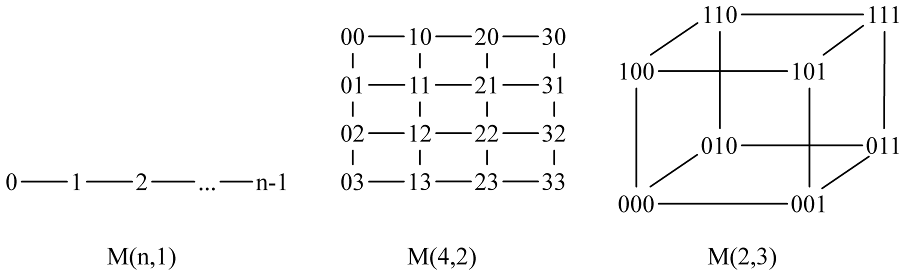


Figure: Some examples of M(n,d)

Mesh networks: properties



Properties of $M(n,d)$

1. $M(n,d)$ has n^d nodes and $dn^d - dn^{d-1}$ edges.
2. $dist(\overline{a}, \overline{b}) = hamming(\overline{a}, \overline{b})$;
3. $diam(M(n,d)) = (n-1) \cdot d$
4. $M(n,d)\mid_{\{\overline{a} \mid a_i = l\}} \cong M(n, d-1)$ for $d > 0$ and fixed i, l
5. $M(n,d)\mid_{\{i\overline{b} \mid i \in [n]\}} \cong M(n,1)$ for fixed $\overline{b} \in [n]^{d-1}$

Overview

Online routing on linear array
Permutation routing without preprocessing in $M(n,1)$ can be performed with routing time $2 \cdot (n-1)$ and buffer size 3

Linear array

The algorithm works in two phases:

- **1st phase:** send packages which destination is to the left
- **2nd phase:** send all another packages

Linear array

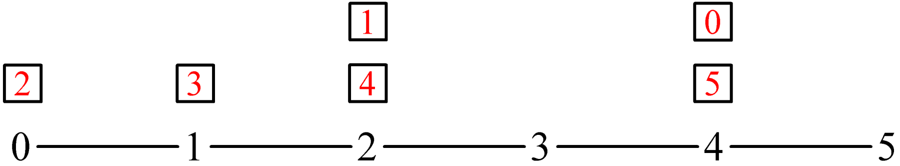
- The buffer size is 3 because any processor *i* stores at most 3 packages:
  - it's own package
  - package addressed to it
  - some other package that must be transferred
- The routing time is obvious

Example

1st phase:

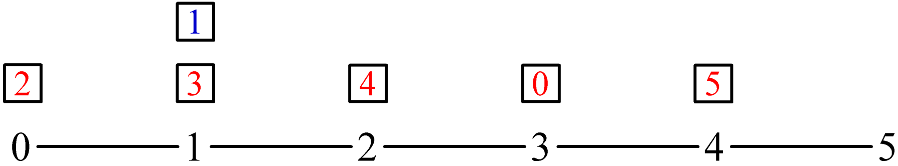
Example

1st phase:



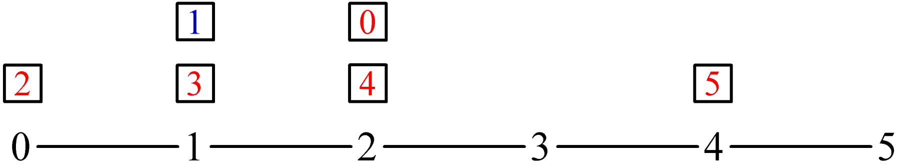
Example

1st phase:



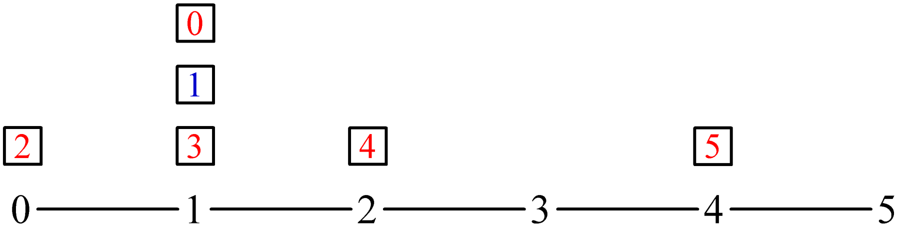
Example

1st phase:



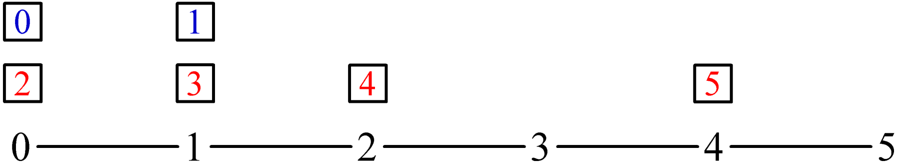
Example

1st phase:



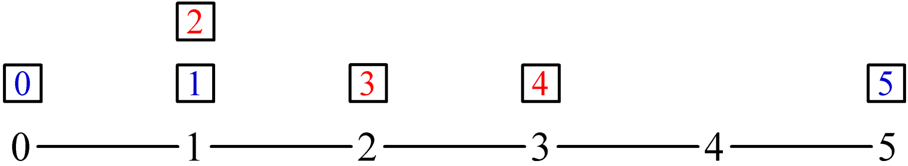
Example

1st phase:



Example

2nd phase:



Example

2nd phase:

Overview

Online routing on 2D array

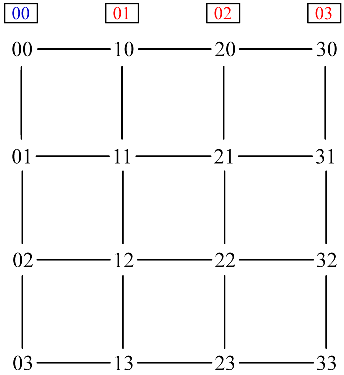
Permutation routing without preprocessing in $M(n, 2)$ can be performed with routing time at most $4 \cdot (n-1)$ and buffer size at most n

2D array

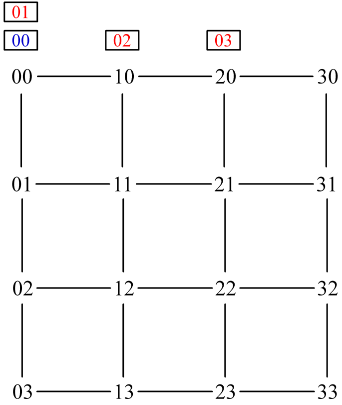
- Routing in 2 phases:
  - routing in **rows**
  - routing in **columns**
- Packages with farthermost destination have higher priority
- Routing time for each dimension $2 \cdot (n - 1)$
- A node can become within first phase at most n packages

2D array: example

Rows:

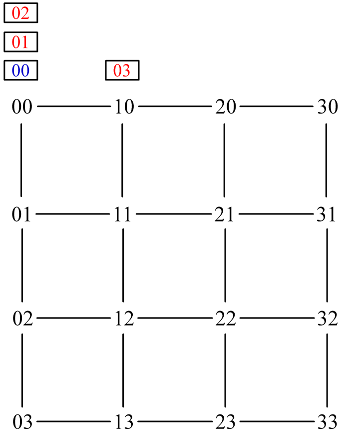


Rows:

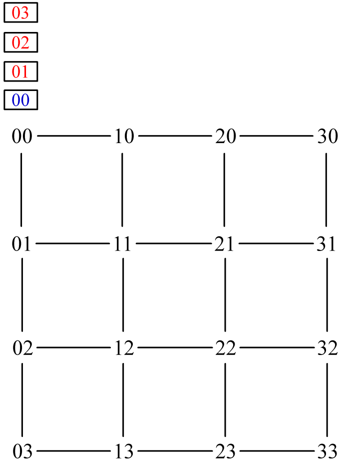


2D array: example

Rows:

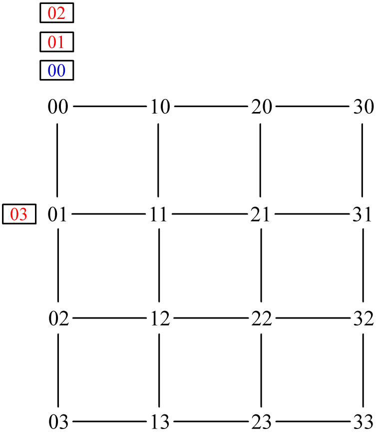


Rows:



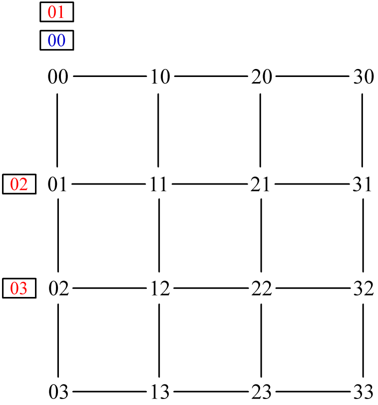
2D array: example

Columns:

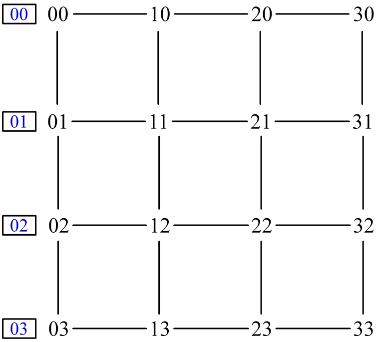


2D array: example

Columns:



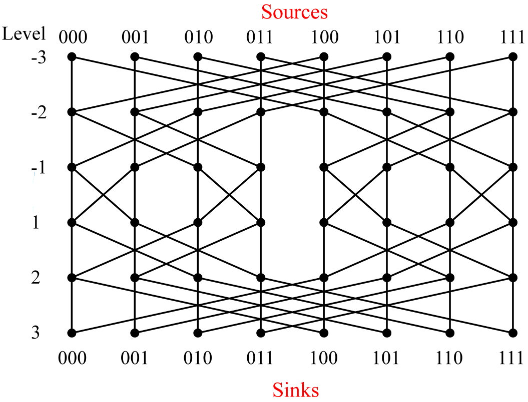
Columns:



The **Permutation network (n,d)-PN**:

- Processors set $P = \{(l, a) | l \in \{-d, ..., -1, 1, ..., d\}, a \in [n]^d\}$
- Communication graph (P, E)

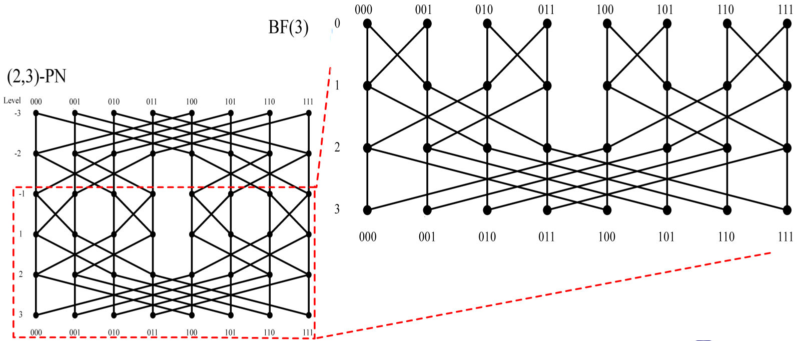
The (2,d)-PN: **Beneš** or **Waksman** network



Butterfly network

The **d-dimensional butterfly network BF(d)**

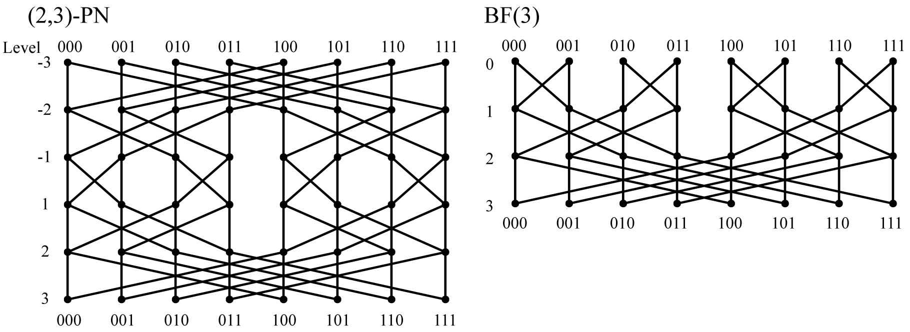
- **(2,d)-PN** with reduced set of processors
- The levels: **0..d**



Properties of (n,d)-PN and BF(d)

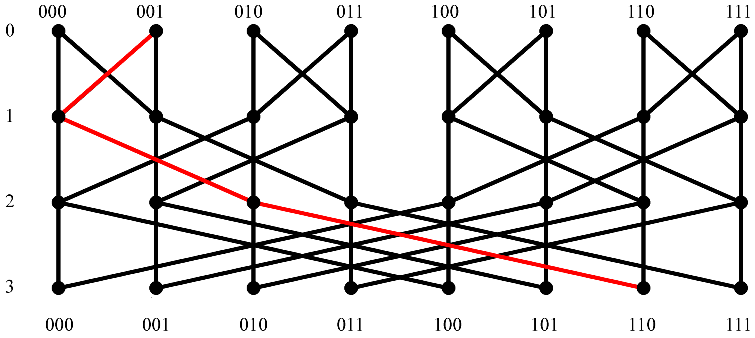
(a) (n,d)-PN has $2d \cdot n^d$ processors;

(b) BF(d) has $(d+1) \cdot 2^d$ processors;



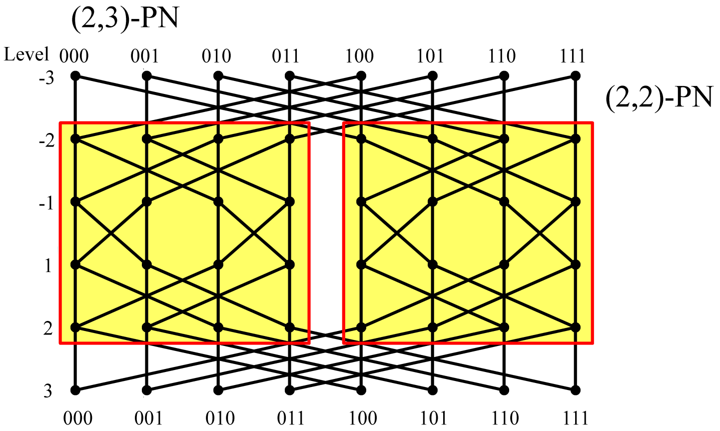
Shortest path on butterfly network

For every processor $(0, \overline{a})$ and $(d, \overline{b})$ there is **exact one shortest path** from $(0, \overline{a})$ to $(d, \overline{b})$.



Recursive decomposition of (n,d)-PN

Permutation networks have **recursive structure.**



Overview

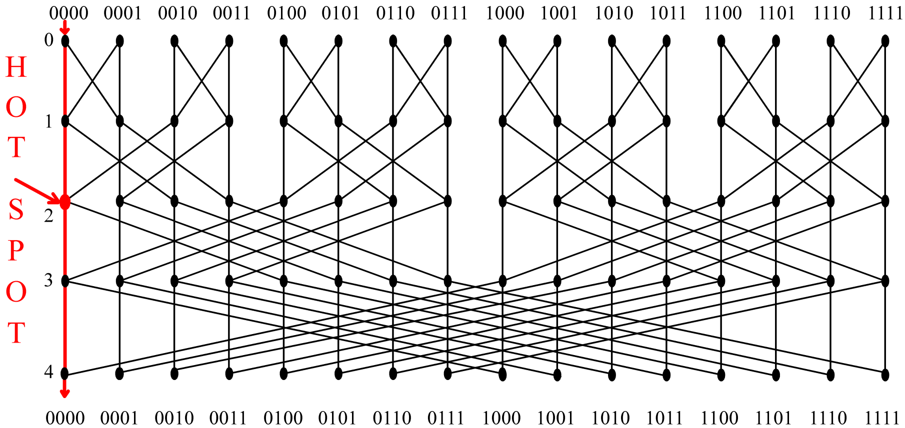
Bit reversal permutation

Example: bit reversal permutation
This permutation reverses the bits of a number. E.g.,

$$1011 \Rightarrow 1101$$

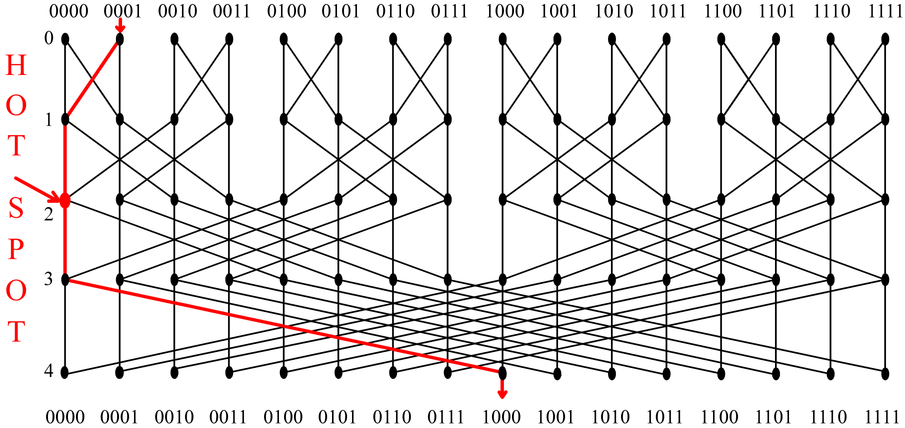
Bit reversal permutation on BF(4)

$$0000 \Rightarrow 0000$$



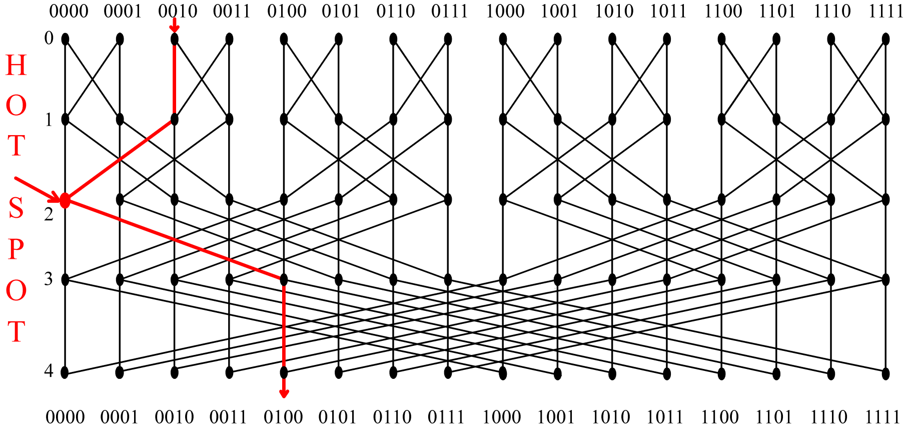
Bit reversal permutation on BF(4)

$0001 \Rightarrow 1000$



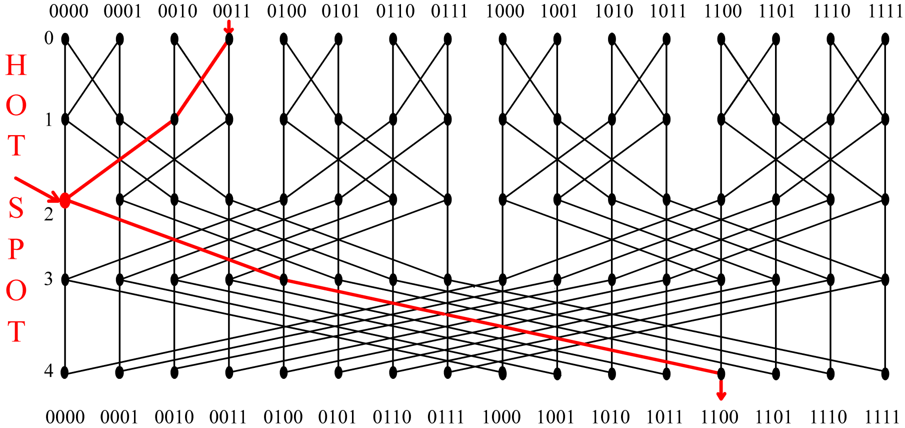
Bit reversal permutation on BF(4)

0010 $\Rightarrow$ 0100



Bit reversal permutation on BF(4)

0011 $\Rightarrow$ 1100



Overview

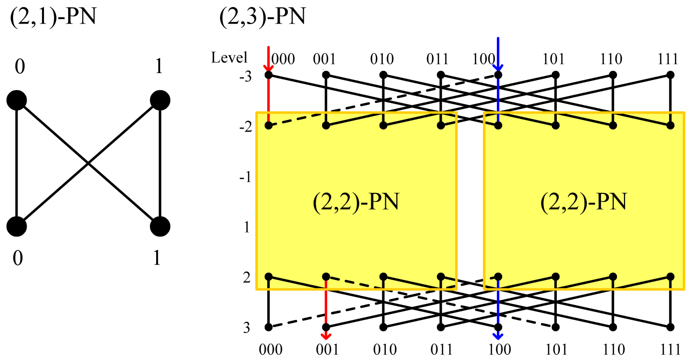
Disjoint paths in (2,d)-PN

For any permutation π there is in (2,d)-PN a set of **disjoint paths** from sources to sinks with length $2d - 1$

Disjoint paths in (2,d)-PN

Construct **disjoint paths** recursive

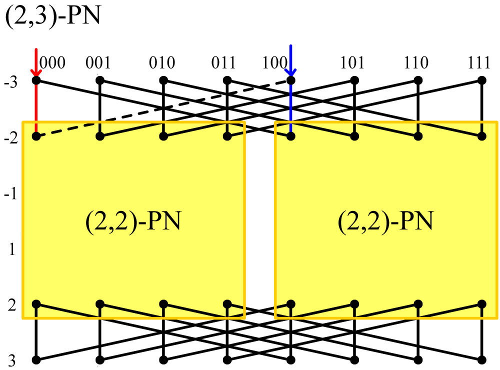
- (2,1)-PN is obviously
- (2,d)-PN consists of **two** (2,d-1)-PN-partitions
- Ensure that any two paths are connected to different sources and sinks of partitions ⇒ the job is done

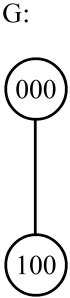


Routing graph

For permutation π construct a **routing graph G**:

- vertices are **paths** numbered after their start points
- edges connect paths that can **crossover** on sinks or sources of partitions

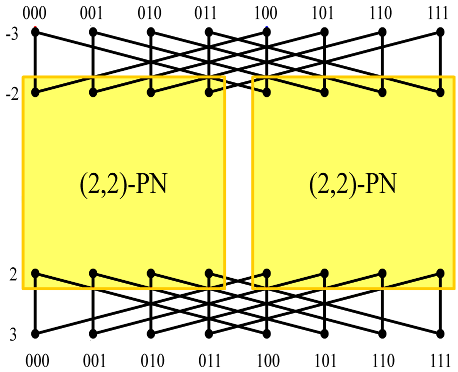




Type 1 edges

Pathes that potentially cross on **sources**:

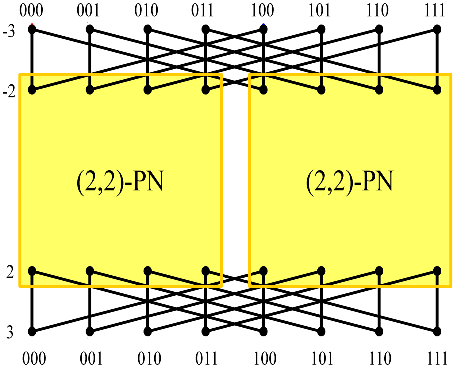
- we connect with type 1 edges
- **start nodes** differ in most significant bit **only**
- each path is incident to **exact one** type 1 edge



Type 2 edges

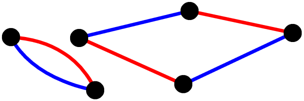
Pathes that potentially cross on **sinks**:

- we connect with type 2 edges
- **end nodes** differ in most significant bit **only**
- each path is incident to **exact one** type 2 edge



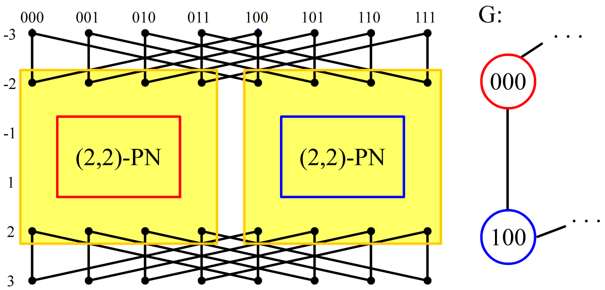
Routing graph

- Each vertex of routing graph is incident to
  - exact one type 1 edge
  - exact one type 2 edge
- $\Rightarrow$ Routing graph G is **2-regular**
- Note: two vertices can be connected with both type 1 and type 2 edges



How we avoid crossing of paths:

- color both partitions in different colors (0 and 1)
- **routing graph**:
  - assign to each vertex (path) one of two colors
  - no two vertices with same color may be adjacent
    $\Rightarrow$ (**vertex coloring**)
- lay each path through partitions of same color



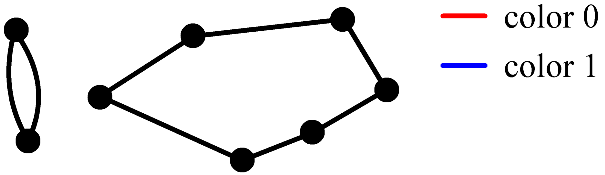
Vertex coloring

Routing graph G is 2-vertex-colorable:

- start with some vertex, assign color **0**
- a adjacent vertex become color **1**
- vertex adjacent to this become color **0**
- and so on **...**

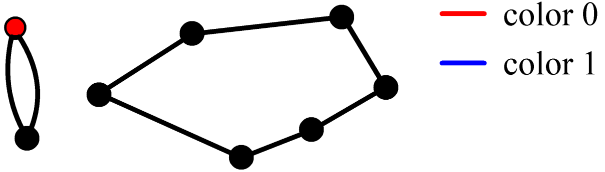
Vertex coloring

Routing graph G:



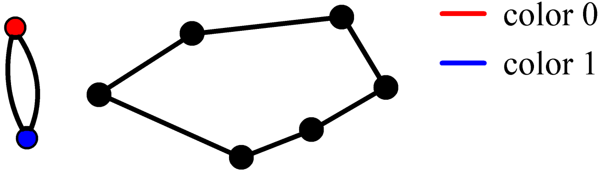
Vertex coloring

Routing graph G:



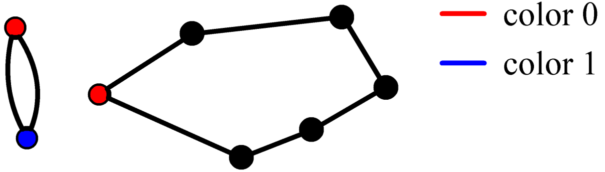
Vertex coloring

Routing graph G:



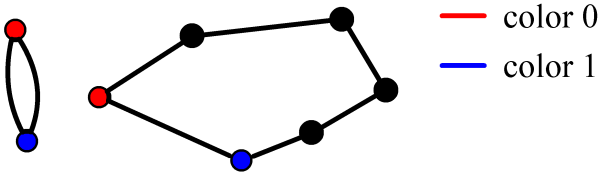
Vertex coloring

Routing graph G:



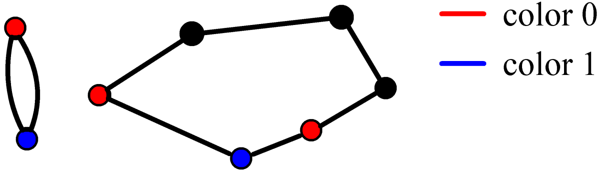
Vertex coloring

Routing graph G:



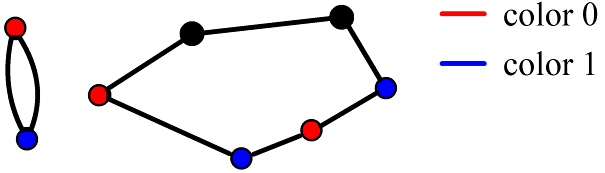
Vertex coloring

Routing graph G:



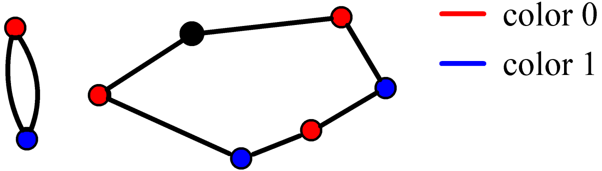
Vertex coloring

Routing graph G:



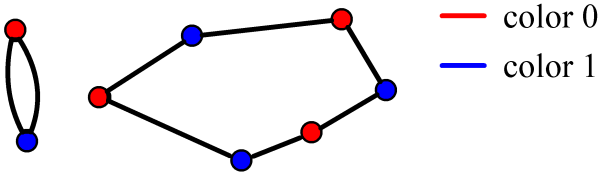
Vertex coloring

Routing graph G:



Vertex coloring

Routing graph G:



Result

Result:

- routing graph is **2-vertex-colorable**
- $\Rightarrow$ we can lay paths **avoiding crossovers**
- $\Rightarrow$ we can route any permutation on (2,d)-PN **without congestion**
- $\Rightarrow$ **routing time:** 2d-1 and **buffer size:** 1

Example: permutation

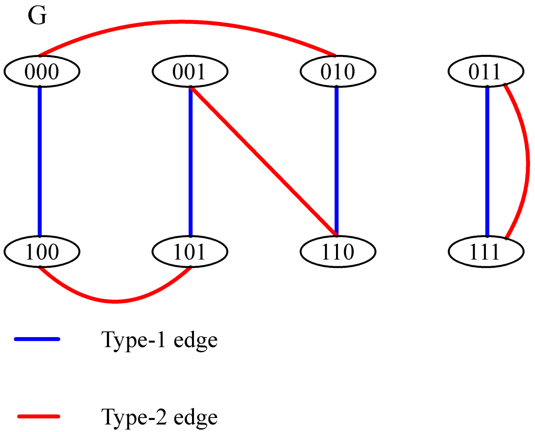
Permutation:

dec	*bin*
$0 \rightarrow 4$	$000 \rightarrow 100$
$1 \rightarrow 1$	$001 \rightarrow 001$
$2 \rightarrow 0$	$010 \rightarrow 000$
$3 \rightarrow 3$	$011 \rightarrow 011$
$4 \rightarrow 2$	$100 \rightarrow 010$
$5 \rightarrow 6$	$101 \rightarrow 110$
$6 \rightarrow 5$	$110 \rightarrow 101$
$7 \rightarrow 7$	$111 \rightarrow 111$

Example: routing graph

Routing graph

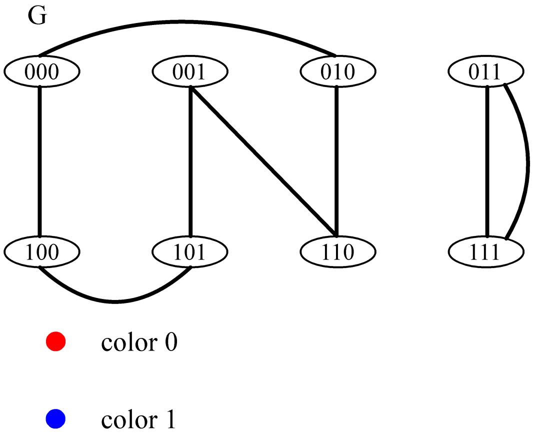
$000 \longrightarrow 100$
$001 \longrightarrow 001$
$010 \longrightarrow 000$
$011 \longrightarrow 011$
$100 \longrightarrow 010$
$101 \longrightarrow 110$
$110 \longrightarrow 101$
$111 \longrightarrow 111$



Example: vertex coloring

Vertex coloring

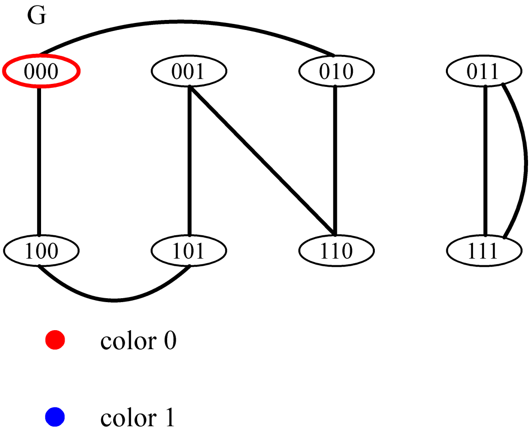
$000 \rightarrow 100$
$001 \rightarrow 001$
$010 \rightarrow 000$
$011 \rightarrow 011$
$100 \rightarrow 010$
$101 \rightarrow 110$
$110 \rightarrow 101$
$111 \rightarrow 111$



Example: vertex coloring

Vertex coloring

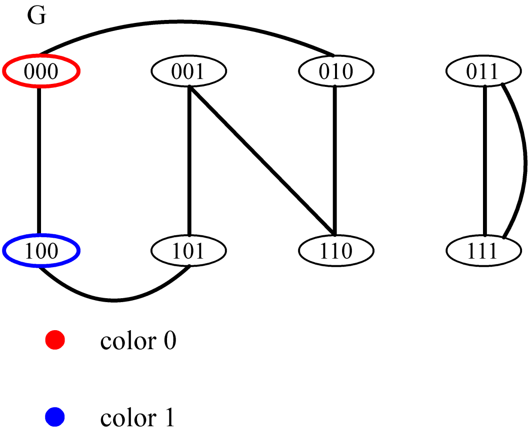
000 $\rightarrow$ 100
001 $\rightarrow$ 001
010 $\rightarrow$ 000
011 $\rightarrow$ 011
100 $\rightarrow$ 010
101 $\rightarrow$ 110
110 $\rightarrow$ 101
111 $\rightarrow$ 111



Example: vertex coloring

Vertex coloring

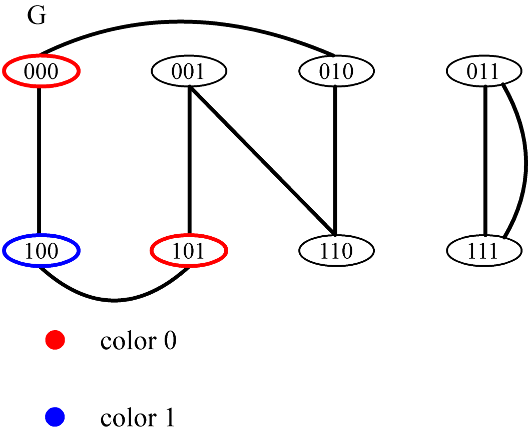
000 → 100
001 → 001
010 → 000
011 → 011
100 → 010
101 → 110
110 → 101
111 → 111



Example: vertex coloring

Vertex coloring

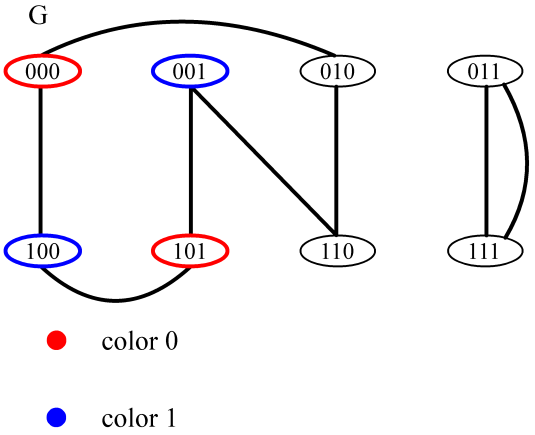
000 → 100
001 → 001
010 → 000
011 → 011
100 → 010
101 → 110
110 → 101
111 → 111



Example: vertex coloring

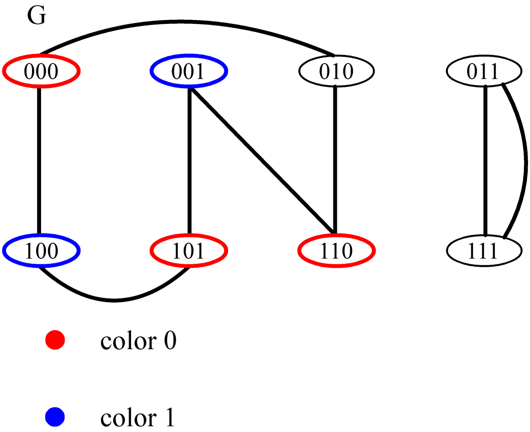
Vertex coloring

$000 \rightarrow 100$
$001 \rightarrow 001$
$010 \rightarrow 000$
$011 \rightarrow 011$
$100 \rightarrow 010$
$101 \rightarrow 110$
$110 \rightarrow 101$
$111 \rightarrow 111$



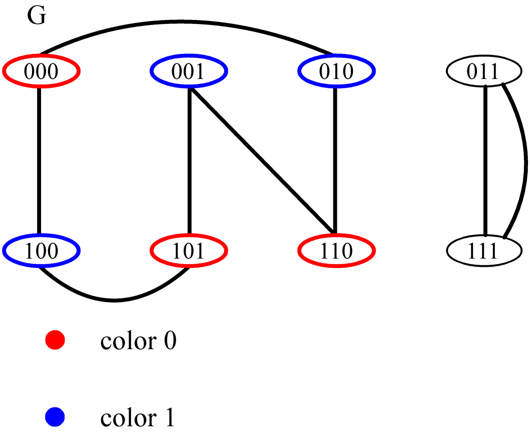
Vertex coloring

$000 \to 100$
$001 \to 001$
$010 \to 000$
$011 \to 011$
$100 \to 010$
$101 \to 110$
$110 \to 101$
$111 \to 111$



Vertex coloring

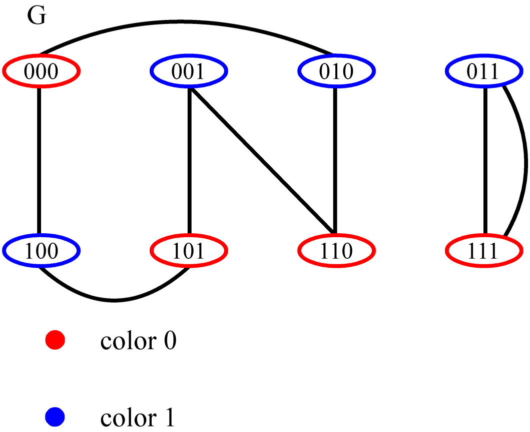
$000 \rightarrow 100$
$001 \rightarrow 001$
$010 \rightarrow 000$
$011 \rightarrow 011$
$100 \rightarrow 010$
$101 \rightarrow 110$
$110 \rightarrow 101$
$111 \rightarrow 111$



Example: vertex coloring

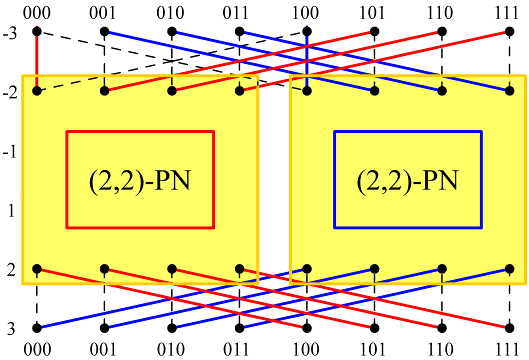
Vertex coloring

000 → 100
001 → 001
010 → 000
011 → 011
100 → 010
101 → 110
110 → 101
111 → 111



Partitions

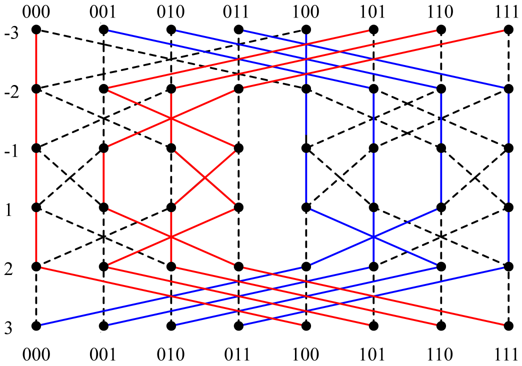
000 → 100
001 → 001
010 → 000
011 → 011
100 → 010
101 → 110
110 → 101
111 → 111



Example: pathes

Pathes

000 $\longrightarrow$ 100
001 $\longrightarrow$ 001
010 $\longrightarrow$ 000
011 $\longrightarrow$ 011
100 $\longrightarrow$ 010
101 $\longrightarrow$ 110
110 $\longrightarrow$ 101
111 $\longrightarrow$ 111



Conclusion

- The permutation networks is a very important and wide used family of networks
- There is an efficient offline routing algorithm for permutation networks