Advanced Course on

Communication in Networks

Friedhelm Meyer auf der Heide

Heinz Nixdorf Institute &
Faculty for Computer Science,
Electrical Engineering,
and Mathematics

Contents

Overview

This course describes some advanced topics from the theory of communication in networks. We will deal with

- Routing in networks
- Scheduling jobs in distributed systems
- Data management in networks.

My lecture notes on "Kommunikation in parallelen Rechenmodellen" form a basis for this lecture; part of the material is adapted from the lecture notes "Communication in parallel systems" by Christian Scheideler.

1 Routing

A *network* is described by a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of *processors* and an undirected graph $G = (\mathcal{P}, E)$. E contains the *edges* or *links*. Each processor has a *buffer* of some size B for each incident edge. Such a buffer can store B packets. Two processors P_i, P_j can directly communicate, if they are connected by a link. In this case P_i can send a packet to P_j in one step.

If P_i and P_j are not connected by a link, they have to choose a *routing path*, i.e., a path from P_i to P_j in G, and to send the packet from vertex to vertex along this path. We will now define the routing problem in some more detail.

A *packet* contains data, its source $s \in \mathcal{P}$, its destination $d \in \mathcal{P}$, and possibly some more information for the routing.

A *routing problem* is described by a set $S = \{(s_i, d_i), i = 1, \ldots, m\}$ of source-destination pairs.

Routing according to S means to send a packet from s_i to d_i, for each $i = 1, \ldots, m$.

We will always distinguish between two parts of the routing process:

- *path selection*: How are the paths in G selected that connect the sources with their destinations?
- *packet switching*: How does a node decide which of the packets it currently stores to move next? This implies the decision, when a node injects a packet into the system.

If the paths w_i are fixed for each source destination pair $(s_i, d_i) \in S$, the following parameters are well defined:

- the *dilation* D (of S in G), i.e., the maximum length of the routing paths used.

- the *congestion* C (of S in G), i.e., the maximum number of routing paths that share the same edge.

It is easily seen that

- $\max\{D, C\}$ is a lower bound for the routing time,
- $D \cdot C$ is an upper bound for the routing time,
- C is an upper bound on the buffer size.

A routing strategy proceeds in rounds.
In a round, each node can choose, for each outgoing edge, a packet from the buffers of the incoming edges, and move it along the edge to the respective buffer of the neighboring node. In addition it can inject a new packet into the system.
The choice of the packets to be moved next is done by the routing (or switching) protocol, the decision, which packet to move along which edge is done by the path selection. The routing time needed for a routing problem S is the number of rounds needed until all packets have reached their destinations.
We will distinguish between different types of routing strategies:

- oblivious versus adaptive. Routing is called oblivious, if each routing path is chosen solely dependent on its source and its destination. In case of a dependence of the choices (which is reasonable in order to reduce congestion) we deal with adaptive routing.
- online versus offline. Routing is called online, if initially each node only knows the packets it injects into the system, but no further packets. (It may know the network topology and the selected paths in case of oblivious routing). Thus the switching protocol is based only on local information.

If global information about the routing problem can be used for the switching protocol and/or the path selection, we talk about adaptive routing.

1.1 A simple example

Consider the ring of length n consisting of n nodes $[n]$ $(=: \{0, \dots, n-1\})$, and edges $\{\{i, i+1\}, i \in [n-1]\} \cup \{\{n-1, 0\}\}$.
Permutation routing can easily be done in time $\lfloor \frac{n}{2} \rfloor$, because no collisions occur, and the ring has diameter $\lfloor \frac{n}{2} \rfloor$.
Now consider an arbitrary routing problem S of size m. How long does it take? If all m packets have to be sent from 0 to $\lfloor \frac{n}{2} \rfloor$, obviously time $m + \lfloor \frac{n}{2} \rfloor - 1$ is necessary and sufficient. Can we route every problem of size m within this time bound? Yes!

Consider the following routing strategy: Our path system consists of the shortest paths. (Note: These paths have length at most $\lfloor \frac{n}{2} \rfloor$.) Our switching role is furthest-to-go (FTG): If packets contend to use the same link in the same direction at the same time, the one that still has the longest path to its destination is preferred. In case of a tie, the one with maximum index wins.

Theorem 1.1 *Every routing problem of size m on the ring of length n can be solved in time at most $m + \lfloor \frac{n}{2} \rfloor - 1$.*

Proof. Let $P = (p_1, \ldots, p_m)$ denote the set of all packets, and let the *rank* $\text{rank}_v(p_i)$ of packet p_i at node v be defined as the number of edges p_i still has to traverse from v plus $i/(m+1)$. According to our preference rules it must hold for any two packets p, q at some node v where p is preferred against q that

$$\text{rank}_v(p) > \text{rank}_v(q) .$$

We will need this observation later.

Now, let us use a proof method called *backwards analysis.* We start with the last packet that reached its destination. Let this packet be called $q_1 \in \{p_1, \ldots, p_m\}$, and let its destination be called v_0. We follow q_1 backwards in time till it was delayed by some other packet, say q_2, at some node v_1. Let ℓ_1 be the number of edges traversed from v_0 to v_1. We continue to follow q_2 backwards in time until it was delayed by some other packet, say q_3, at some node v_2. Let ℓ_2 be the number of edges traversed from v_1 to v_2. In general, we will follow packet q_i backwards in time until it was delayed by some other packet, say q_{i+1}, at some node v_i. Let ℓ_i be the number of edges traversed from v_{i-1} to v_i. If we reach a packet q_i that has not been delayed by any other packet, we simply follow it backwards in time until it reached its source node, called v_i.

Assume now that we touched s packets $q_1, \ldots, q_s$ in the course of this argument. Using our observation above, we can state the following three facts:

1. $\text{rank}_{v_0}(q_1) \geq 0$
2. $\text{rank}_{v_i}(q_{i+1}) > \text{rank}_{v_i}(q_i)$
3. $\text{rank}_{v_i}(q_i) = \text{rank}_{v_{i-1}}(q_i) + \ell_i$

We will use these facts to prove the following lemmas.

Lemma 1.2 $\sum_{i=1}^{s} \ell_i \leq \lfloor n/2 \rfloor$.

Proof. A straightforward induction argument shows that the above facts imply $\text{rank}_{v_k}(q_k) \geq \sum_{i=1}^{k} \ell_i$ for every $k \in \{1, \ldots, s\}$. However, since $\text{rank}_{v_s}(q_s)$ must be smaller than $\lfloor n/2 \rfloor + 1$ due to a maximum path length of $\lfloor n/2 \rfloor$ we get

$$\lfloor n/2 \rfloor + 1 > \text{rank}_{v_s}(q_s) \geq \sum_{i=1}^{k} \ell_i$$

which yields the lemma since the ℓ_i are integers. $\square$

Lemma 1.3 $s \leq m$

Proof. Due to fact (2) and the fact that if $\text{rank}_v(p) > \text{rank}_v(q)$ for two packets p and q then this relationship also holds for all other nodes visited by p and q, it holds that no packet can appear more than once in the backwards argument. Hence, $s \leq m$. $\square$

Obviously, the total runtime of our strategy is equal to the time covered by the backwards argument. Since the number of time steps covered by this argument is exactly $\sum_{i=1}^{s} \ell_i + s - 1$, the two lemmas above imply a runtime of at most

$$\lfloor n/2 \rfloor + m - 1 \ .$$

$\square$

Theorem 1.1 is worst case optimal, since a routing problem can easily be set up that requires $m + \lfloor n/2 \rfloor - 1$ steps.

2 Some networks and path systems

We present some basic networks and some of their properties.

- **(n, d)-meshes.**

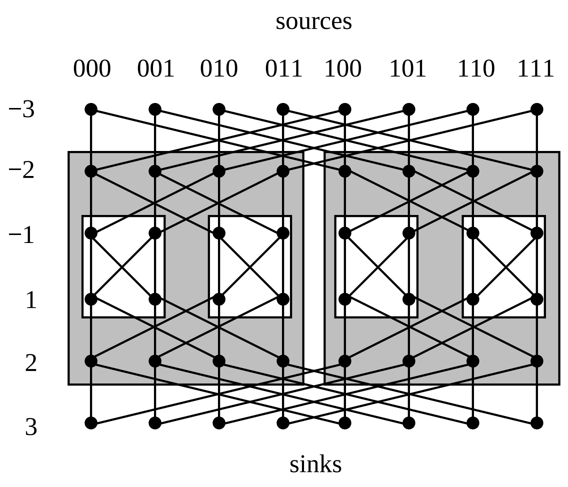
  Let n, d be positive integers. The (n, d)-mesh $M(n, d)$ has nodes $[n]^d$ (note: $[n] := \{0, \ldots, n-1\}$). $\{\bar{a}, \bar{b}\} \in E$, iff $|\bar{a} - \bar{b}| = 1$, i.e., $\bar{a}$ and $\bar{b}$ differ in exactly one coordinate i, and $|a_i - b_i| = 1$.

  Some properties of $M(n, d)$:

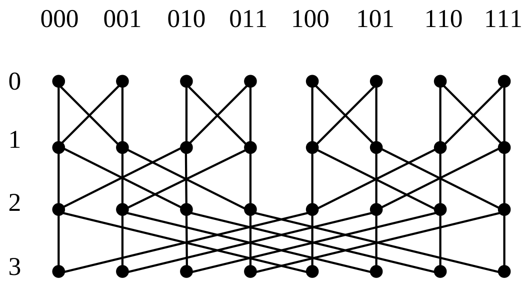
  - *size:* $N = n^d$.
  - *diameter:* $d \cdot (n-1) = d \cdot (N^{\frac{1}{d}} - 1)$.
  - *degree:* $2d$.
  - *a simple path system:*
    dimension order paths, i.e. path from $(a_1, \ldots, a_d)$ to $(b_1, \ldots, b_d)$ goes:
    $(a_1, \ldots, a_d) \rightarrow (b_1, a_2, \ldots, a_d) \rightarrow (b_1, b_2, a_3, \ldots, a_d) \rightarrow \ldots (b_1, \ldots, b_d)$.

    These paths are shortest paths. But even for permutation routing, we get high congestion.

    Example: Reversal permutation: $(a_1, \ldots, a_d) \rightarrow (a_d, a_{d-1}, \ldots, a_1)$.
    All packets with sources $(0, \ldots, 0, a_{d/2}, \ldots, a_d)$ go through $(0, \ldots, 0)$, thus congestion $\geq \sqrt{N}$. (We will later see that no path system has better worst case congestion for permutations in the mesh).



(a) $(2,3)$-PN



(b) BF(3)

Figure 1: Examples for the Butterfly and the Benes-Network.

- *special cases:*
  - $M(2,d)$ is better known as the *d-dimensional Hypercube.*
  - $M(n,2)$ is a *linear array* of n nodes.

- **(n,d)-Permutation networks.**

  Let n,d be positive integers. The (n,d)-Permutation network, (n,d)-PN has nodes $\{(l,\bar{a}), l \in \{-d,\ldots,-1,1,\ldots d\}, \bar{a} \in [n]^d\}$ and edges

$$\begin{aligned} E &= \{\{(l,\bar{a}),(l+1,\bar{a}')\} \mid l < -1, a_i = a'_i \text{ für alle } i \neq |l|-1\} \\ &\cup \{\{(l,\bar{a}),(l-1,\bar{a}')\} \mid l \geq 1, a_i = a'_i \text{ für alle } i \neq l-1\} \\ &\cup \{\{(-1,\bar{a}),(1,\bar{a}')\} \mid a_i = a'_i \text{ für alle } i \neq 0\} \end{aligned}$$

  An example is depicted in Figure 1(a) and a recursive definition in Figure 2. We do not formalize it in this text.

  We only want to route from sources (nodes $(0,\bar{a})$) to sinks (nodes $(2d,\bar{a})$).

  Some properties of (n,d)-PN:

  - *Source-destination distance:* $2d-1$.
  - *size:* $2d \cdot n^d =: 2d \cdot N = 2\log_2(N) \cdot N$.
  - *degree:* $2n$.

– *path selection:* see Section 2.1

– *special cases:*

  * *Benes- Network:* $(2, d)$-PN (Figure 1)
  * *Butterfly:* upper half of $(2, d)$-PN (Figure 1)

2.1 Adaptive Path Selection in (n, d)-PN and $M(n, d)$

Our first goal is to find disjoint paths in (n, d)-PN connecting source $\bar{a}$ to destination $\Pi(\bar{a}), \bar{a} \in [n]^d$, for arbitrary permutations Π. To do so, we need some prerequisities from graph theory.

Let $G = (V, E)$ be an undirected graph (multiple edges allowed). $M \subseteq E$ is a *matching* in G, if no two edges in M are incident, i.e., share a vertex. M is a *perfect matching*, if it covers all vertices of G. $f : E \to \{1, \ldots, k\}$ is a k-coloring of G, if incident edges e, e' fulfil $f(e) \neq f(e')$ (i.e., have different colors).

Note: Each color-class $E_\ell = \{e \in E, f(e) = \ell\}$ forms a matching.

For a set $S \subseteq V, \Gamma(S)$ denotes $\{u \in V \setminus S, \exists \{u, v\} \in E \text{ with } u \in S\}$. ($\Gamma(S)$ is the *neighbourhood* of S in G.)

Theorem 2.1 *(Hall's Theorem)*
Let $G = (V_1 \dot{\cup} V_2, E)$ be bipartite, $|V_1| = |V_2|$. Then: G contains a perfect matching, if and only if, for each $S \subseteq V_1, |\Gamma(S)| \geq |S|$. □

(For a proof, see Script "Kommunikation in parallelen Rechenmodellen".)

Corollary 2.2 *(Coloring Theorem)*
Each bipartite graph of degree c is c-colorable. □

We now are ready to state our first result.

Theorem 2.3 *For every permutation Π on $[n]^d$ there are disjoint paths $w_{\bar{a}}, \bar{a} \in [n]^d$ of lengths $2d - 1$ such that $w_{\bar{a}}$ connects source $\bar{a}$ with destination $\Pi(\bar{a})$ of (n, d)-PN.*

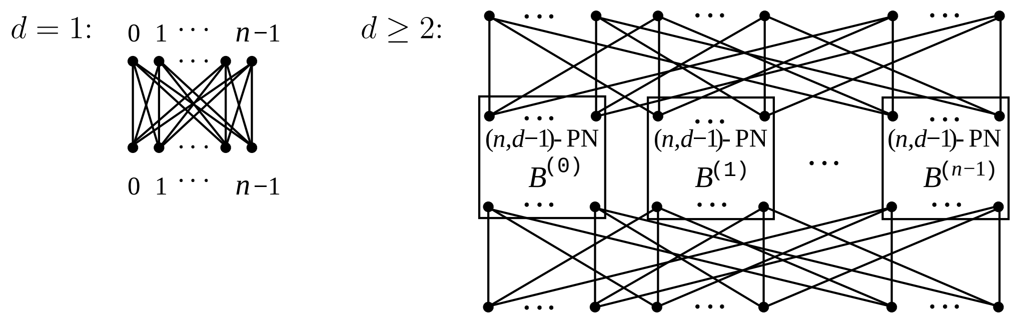
Proof: We will construct these paths inductively on d. (see Figure 2 for the recursive decomposition of (n, d)-PN.)

$d = 1$:

In this case the paths are just edges in the complete bipartite graph $(n, 1)$-PN.

$d > 1$:

Note: We may direct $w_{\bar{a}}$ through any of the n subnetworks $B^{(0)}, \ldots, B^{(n-1)}$ of (n, d)-PN. If we choose some $B^{(i)}$, then the source and the destination of $B^{(i)}$ used by $w_{\bar{a}}$ are unique. Our goal now is to choose a $B^{(i)}$ for each $w_{\bar{a}}$ such that



$d = 1$:
0 1 ⋯ n−1
0 1 ⋯ n−1
$d \geq 2$:
(n,d−1)- PN $B^{(0)}$
(n,d−1)- PN $B^{(1)}$
(n,d−1)- PN $B^{(n-1)}$

Figure 2: Recursive Construction of (n, d)-PN.

$(*)$ every source and every destination of every $B^{(i)}$ is used by exactly one $w_{\bar{a}}$.

If we find such an assignment of paths to the $B^{(i)}$'s, then we are left with a permutation routing problem in each $B^{(i)}$. For these problems, disjoint paths exist by induction hypothesis.

Thus all we have to do is to assign paths to $B^{(i)}$'s, such that $(*)$ holds. For this we consider G_Π, the *collision graph* of Π in (n,d)-PN: G_Π is bipartite, its nodes are $[n]^{d-1} \dot\cup [n]^{d-1}$, representing the sources and destinations of $B^{(0)}$, resp.

$\{\bar{a}, \bar{b}\}$ is an edge of G_Π, if there are $i, j \in [n]$ such that $\Pi(i\bar{a}) = j\bar{b}$, i.e., the path $w_{i\bar{a}}$, when directed through $B^{(0)}$, enters $B^{(0)}$ at source $\bar{a}$ and leaves it at destination $\bar{b}$.

By construction G_Π has degree n. Thus, by Theorem 2.2, G_Π is n colorable.

Let W_i denote the set of paths $w_{\bar{a}}$ defining an edge in G_Π with color i. Direct these paths through $B^{(i)}$. As the edges of G_Π defined by the paths in W_i form a matching, the above assignment fulfils property $(*)$

Thus, by the discussion above, we can proceed inductively to construct the desired disjoint paths.

□

An immediate consequence of Theorem 2.3 is the following.

Theorem 2.4 *Offline adaptive permutation routing in (n, d)-PN can be done in time $2d-1$.* □

We now will see that the some techniques also yield efficient permutation routing in $M(n, d)$.

Theorem 2.5 *Offline, adaptive permutation routing in $M(n, d)$ can be done in time $2(n-1)d$.*

(Note that this is only by a factor 2 away from the diameter bound.)

Proof. We again argue inductively w. r. t. d.

$d = 1$:
On a linear array we can route permutations in time $2(n-1)$.

$d > 1$:
We partition $M(n,d)$ in n copies $M^{(0)}, \ldots, M^{(n-1)}$ of $M(n,d-1)$. Now consider the same graph G_Π with vertex set $[n]^{d-1} \dot{\cup} [n]^{d-1}$ as in the previous proof.
By this, each packet $p_{\bar{a}}, a \in [n]^d$ is assigned a color $f(\bar{a}) \in [n]$.
The 1D-source array of packet $p_{i\bar{a}}$, $i \in [n], \bar{a} \in [n]^{d-1}$, is $A_{\bar{a}} = \{0\bar{a}, \ldots, (n-1)\bar{a}\}$. (Note: this is a copy of $M(n,1)$). The 1D-destination array of packet $p_{\bar{b}}, \bar{b} \in [n]^{d-1}$, is $A_{\bar{a}}$, $\bar{a} \in [n]^{d-1}$ with $\Pi(\bar{b}) = i\bar{a}$ for some $i \in [n]$.
Now we are ready to state the routing protocol:

(i) for each $\bar{a} \in [n]^{d-1}$ in parallel: permute $p_{i\bar{a}}$, $i \in [n]$, $i \in [n]$ on $A_{\bar{a}}$ such that $p_{i\bar{a}}$ reaches $M^{f(i\bar{a})}$,

(ii) recursively permute the packets in each $M^{(i)}$, such that each packet reaches its 1D-destination array,

(iii) permute each array $A_{\bar{a}}$ such that each packet reaches its destination. Note that, analogously to our discussion w. r. t. routing in (n,d)-PN, the routing problems in (i), (ii), (iii) are in fact permutation routing problems.

□

3 Oblivious path selection in the hypercube

The following theorem demonstrates that no matter what kind of network and what kind of path system is used for oblivious routing, there is always a permutation that causes high congestion.

Theorem 3.1 *(Borodin/Hopcraft). Consider an arbitrary network G with n vertices and degree d, and an arbitrary path system P in G. Then there is a permutation Π such that routing Π along paths from P causes congestion at least $\sqrt{\frac{n}{d}}$.* □

For the hypercube H_d this means that routing along the path system described in Section 2, the so-called dimension-order routing, or along any other fixed path system, there is always a permutation that causes congestion $\sqrt{\frac{n}{\log(n)}}$, where $n = 2^d$ denotes the size of H_d. (Note: H_d has degree d.)
We now show that this very large congestion can be avoided by introducing randomization to the path system. We now apply the so-called *Valiant's trick*:
Consider a fixed path system P in network G. The new path system is constructed as follows: For each source-destination pair (s,d) choose randomly, uniformly, independently

an intermediate node v, and choose as path the concatenation of the paths from s to v and from v to d from P.

We apply this to the system P of dimension-order paths for the hypercube H_d. The following probabilities are w. r. t. these random choices of the intermediate nodes.

Theorem 3.2 *Let $n = 2^d, \ell \geq 1$ arbitrary. Consider H_d and an arbitrary permutation Π on its nodes. Then, the expected congestion caused by routing Π along the random path system constructed via the dimension-order paths yields congestion at most $(4\ell + 8)\log(n)/\log\log(n)$, with probability $(1-n^{-\ell})$. The paths have length at most $2d = 2\log(n)$.*

Proof. Fix an edge $e = ((c_0, \ldots, , c_{i-1}, c_i, c_{i+1}, \ldots, c_{d-1}), (c_0, \ldots, c_{i-1}, \bar{c}_i, c_{i+1}, \ldots, c_{d-1}))$ of H_d.

Now consider a source-destination pair $((a_0, \ldots, a_{d-1}), (v_0, \ldots, v_{d-1}))$. The corresponding dimension order path p goes through e, if and only if the following holds:

- $(v_0, \ldots, v_{i-1}) = (c_0, \ldots, c_{i-1})$
- $a_i = c_i$ and $v_i = \bar{c}_i$
- $(a_{i+1}, \ldots, a_{d-1}) = (c_{i+1}, \ldots, c_{d-1})$

Now assume that $(a_0, \ldots, a_{d-1})$ is fixed, and $(v_0, \ldots, v_{d-1})$ is chosen randomly. Let $X_{p,e}$ denote the event that the path p goes through edge e. Then,

$$\mathbf{Pr}\left[X_{p,e}\right] = \begin{cases} \mathbf{Pr}\left[(v_0, \ldots, v_{i-1}) = (c_0, \ldots, c_{i-1}) \text{ and } v_i = \bar{c}_i\right] & \text{if } (a_i, \ldots, a_{d-1}) = (c_i, \ldots, c_{d-1}) \\ 0 & \text{else} \end{cases}$$

Thus,

$$\mathbf{Pr}\left[X_{p,e}\right] = \begin{cases} (\frac{1}{2})^{i+1} & \text{if} (a_i, \ldots, a_{d-1}) = (c_i, \ldots, c_{d-1}) \\ 0 & \text{else} \end{cases}$$

Therefore, each of the 2^i sources a with $(a_i, \ldots, a_{d-1}) = (c_i, \ldots, c_{d-1})$ have probability $(\frac{1}{2})^{i+1}$ for the event that their path to a random destination crosses e. These events are independent. Therefore:

$$\begin{aligned} &\mathbf{Pr}\left[\text{at least } r \text{ paths from a source to a random node } v \text{ go through } e\right] \\ &\quad\leq \binom{2^i}{r} \cdot \left(\frac{1}{2}\right)^{(i+1)\cdot r} \qquad \left(\binom{n}{k} \leq \left(\frac{n \cdot e}{k}\right)^k\right) \\ &\quad\leq \left(\frac{2^i \cdot e}{r}\right)^r \cdot \left(\frac{1}{2}\right)^{(i+1)\cdot r} \\ &\quad= \left(\frac{e}{2r}\right)^r \quad . \end{aligned}$$

Since the hypercube has $\frac{d}{2} \cdot 2^d$ edges, we can conclude:

$$
\begin{aligned}
&\mathbf{Pr}\,[\mathsf{Congestion} \geq \gamma d \,/\, \log(d)] \\
&\qquad \leq \tfrac{d}{2} \cdot 2^d \cdot \left(\tfrac{e}{2\gamma d / \log(d)}\right)^{\gamma d / \log(d)} && \left(\tfrac{d}{2} \leq 2^d \quad \gamma \geq 2\right) \\
&\qquad \leq 2^{2d} \cdot \left(\tfrac{d}{\log(d)}\right)^{-\gamma d / \log(d)} && \left(\tfrac{d}{\log(d)} \geq \sqrt{d}\right) \\
&\qquad \leq 2^{2d} \cdot d^{-\frac{1}{2}\gamma d / \log(d)} = 2^{2d} \cdot 2^{-\frac{1}{2}\gamma d} \\
&\qquad \leq 2^{(2-\frac{1}{2}\gamma)d}.
\end{aligned}
$$

Thus, $\mathbf{Pr}[\mathsf{Congestion} \geq \gamma d \,/\, \log(d)] \leq n^{-\ell}$ for $\gamma = 2\ell + 4$.
Thus, we have shown that routing a packet from each source node a to a random destination v yields congestion at most $(2\ell + 4) \cdot d \,/\, \log(d)$, with probability at least $1 - n^{-\ell}$. Now, given a permutation Π, the same argument holds for routing from random intermediate nodes to the destination nodes. Thus the Theorem follows. □

Thus we can achieve very low congestion for arbitrary permutations, with high probability. But we have to pay with non-optimal path length: Whereas shortest paths in H_d have length at most $d = \log(n)$, we need paths up to length $2\log(n)$. Can we do better? Yes! Very recently, Berthold Vöcking has presented a simple trick to do so:
For each source-destination pair a, b choose a random intermediate point $v = (v_0, \ldots, v_{d-1})$ as above. Our Theorem yields congestion $O(d/\log(d))$, with high probability (w. h. p).
Now consider a second path system, where each intermediate note v is replaced by $\bar{v} = (\bar{v}_0, \ldots, \bar{v}_{d-1})$.
For each source destination pair now choose the shorter of the two paths in order to form the *randomized complemental path system.*

Theorem 3.3 *Each path in the randomized complemental paths system has length at most* $d = \log(n)$*. Routing an arbitrary permutation along this path system causes congestion* $O(d \,/\, \log(d))$*, w. h. p.*

Proof. The congestion bound is clear, because, by the previous theorem, it even holds if we route Π along both path systems.
We have to check the path lengths. For a destination pair a, b with intermediate node v, the path length is $h(a, v) + h(b, v)$, where $h(.,.)$ denotes the Hamming distance.
Note: $h(a, v) = d - h(a, \bar{v})$ for all $a, v \in \{0, 1\}^d$.
Let p denote the path $a \to v \to b$, p' the path $a \to \bar{v} \to b$, $|p|$ the length of path p.
Then: $|p| = h(a, v) + h(b, v) = d - h(a, \bar{v}) + d - h(b, \bar{v}) = 2d - |p'|$.
Thus $|p|$ or $|p'|$ is at most d. □

4 Offline Switching

Suppose that a collection of simple (i.e. loop free) paths is given to us. Along each path one packet has to be sent. All we know about the path collection is that is has a congestion of C and a dilation of D. In this case it is clear that at least $\max[C, D]$ steps are needed to send the packets to their destinations. On the other hand, for any greedy method (every time when at least one packet is waiting to traverse a link, this link forwards a packet) at most $C \cdot D$ steps are needed. However, would it be also possible to send the packets to their destinations in $O(C + D)$ steps? We will investigate this problem in this section.

4.1 Probabilistic Methods

First, we introduce two methods that will be important for proving the existence of good switching strategies.

Lemma 4.1 (Chernoff Bounds) *Let $X_1, \ldots, X_n$ are independent binary random variables, and let $X = \sum_{i=1}^{n} X_i$. Then it holds for all $\delta \geq 0$ and $\mu \geq \mathbf{E}[X]$ that*

$$\begin{aligned} \mathbf{Pr}\left[X \geq (1+\delta)\mu\right] &\leq \left(\frac{\mathrm{e}^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu} \\ &\leq \mathrm{e}^{-\frac{\delta^2 \mu}{2(1+\delta/3)}} \\ &\leq \mathrm{e}^{-\min[\delta^2,\, \delta]\cdot\mu/3} , \end{aligned} \tag{1}$$

where e *denotes the Euler number* 2.718.... *Furthermore, it holds for all $0 \leq \delta \leq 1$ and $\mu \leq \mathbf{E}[X]$ that*

$$\begin{aligned} \mathbf{Pr}[X \leq (1-\delta)\mu] &\leq \left(\frac{\mathrm{e}^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu} \\ &\leq \mathrm{e}^{-\delta^2\mu/2} . \end{aligned} \tag{2}$$

The Chernoff bounds are very useful for proving that certain outcomes occur with high probability. Sometimes, however, it is not known whether some outcome can occur at all. In this case the Lovász Local Lemma may be used.

Lemma 4.2 (Lovász Local Lemma, Symmetric Case) *Let $A_1, \ldots, A_n$ be "bad" events in an arbitrary probability space. Suppose that every event A_i is independent of all the other events A_j but at most d, and that $\mathbf{Pr}[A_i] \leq p$ for all $1 \leq i \leq n$. If*

$$\mathrm{e}p(d+1) \leq 1 , \tag{3}$$

then $\mathbf{Pr}[\bigcap_{i=1}^{n} \bar{A}_i] > 0$, i.e. it is possible for no bad event to hold.

4.2 Offline Switching

The central result of this section is the following theorem.

Theorem 4.3 (Leighton, Maggs, Rao) *For any collection of paths with congestion C and dilation D, there is an offline schedule that needs at most $(C+D) \cdot 2^{O(\log^*(C+D))}$ time steps to send a packet along each of these paths.*

Proof. W.l.o.g. we assume in the following that $\max[C, D]$ is beyond some constant value. Otherwise, we simply use a brute force schedule to arrive at a schedule of time $O(C+D)$.
We will present a proof given by Leighton, Maggs and Rao. Our strategy for constructing an efficient schedule is to make a succession of refinements, starting with an initial schedule S_0 in which each packet is forwarded at every time step until it reaches its destination (see Figure 3).

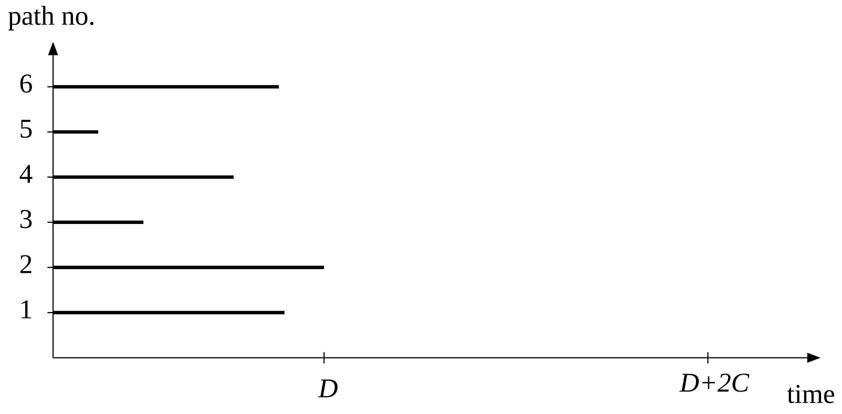


Figure 3: Schedule S_0.

This initial schedule is as short as possible. Its length is exactly D. Unfortunately, up to C packets may have to use an edge at a single time step in S_0, whereas in the final schedule at most one packet is allowed to use an edge at each step. Each refinement will bring us closer to meeting this requirement by bounding the congestion within smaller and smaller time intervals. The proof uses the Chernoff bounds and the Lovász Local Lemma at each refinement step. In the following, a *t-interval* denotes a time interval consisting of t consecutive time steps. A *t-frame* is defined as a t-interval that starts at an integer multiple of t.
Let $I_0 = \max[C, D]$ and $I_j = \lceil 18(\ln I_{j-1} + 1) \rceil$ for all $j \geq 1$. (Since $I_{j-1} \geq 1$, this means that $I_j \geq 18$ for all j.) The first step is to assign an initial delay to each packet, chosen independently and uniformly at random from the range $\Delta_1 = [2C]$. In the resulting schedule, S_1, a packet that is assigned a delay of δ waits in its source node for δ steps,

then moves on without waiting again until it reaches its destination (see Figure 4). The length of S_1 is at most $D+2C$. We use the Lovász Local Lemma to show that if the delays are chosen independently and uniformly at random and I_1 is sufficiently large, then with nonzero probability the congestion at any edge in any I_1-interval is at most I_1. Thus, such a set of delays must exist.

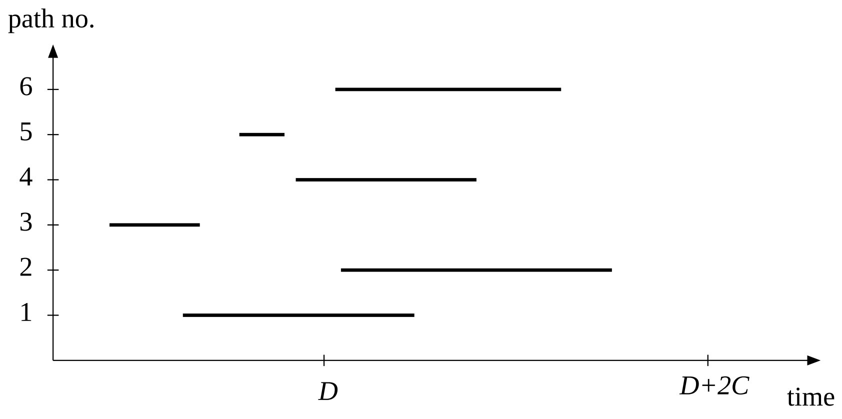


Figure 4: Schedule S_1.

To apply the Lovász Local Lemma, we associate a bad event with each edge. The bad event for edge e is that more than I_1 packets use e in some I_1-interval. For this we have to bound the dependence b among the bad events and the probability p that a bad event occurs.

We first bound the dependence. Whether or not a bad event occurs depends solely on the delays assigned to the packets that pass through the corresponding edge. Thus, two bad events are independent unless some packet passes through both of the corresponding edges. Since at most C packets pass through an edge and each of these packets passes through at most D other edges, the dependence b of the bad events is at most $C \cdot D$.

Next we bound the probability that a bad event occurs. Let us consider some fixed edge e and I_1-interval J. Let the packets traversing e be numbered from 1 to $m \leq C$. For every $i \in \{1, \ldots, m\}$, let the binary random variable X_i be one if and only if packet i traverses e during J. Let $X = \sum_{i=1}^{m} X_i$. Since the packets choose their delays uniformly at random from a range of size $2C$, we get $\mathbf{Pr}[X_i = 1] \leq I_1/(2C)$ for all $i \in \{1, \ldots, m\}$. Hence,

$$\mathbf{E}\left[X\right] = \sum_{i=1}^{m} \mathbf{E}\left[X_i\right] \leq I_1/2 \; .$$

Together with the Chernoff bounds (see Lemma 4.1, (1)) we therefore get with $\epsilon = 1$ that

$$\mathbf{Pr}\left[X \geq I_1\right] = \mathbf{Pr}\left[X \geq (1+\epsilon)I_1/2\right] \leq \mathrm{e}^{-I_1/6} \leq \mathrm{e}^{-3(\ln(\max[C,D])+1)} = (\mathrm{e} \cdot \max[C,D])^{-3} \; .$$

Since for each edge there are at most $D+2C$ many I_1-intervals to consider, the probability that a bad event occurs for edge e is bounded by

$$p \leq (D+2C) \cdot (\mathrm{e} \cdot \max[C,D])^{-3} < \frac{1}{\mathrm{e}(C \cdot D + 1)}$$

for $\max[C,D] \geq 18$. Thus, $\mathrm{e}p(b+1) \leq 1$ and therefore, by the Lovász Local Lemma (see Lemma 4.2, (3)), there is some assignment of delays such that the congestion in each I_1-interval is bounded by I_1.

We now break schedule S_1 into I_1-frames and continue to schedule each frame independently. So each frame can be viewed as a separate scheduling problem where the origin of a packet is its location at the beginning of the frame, and its destination is its location at the end of the frame. Our next refinement step will be to choose, for each frame, a random initial delay for each packet that visits at least one edge within this frame. In the resulting schedule S_2, the frames (enlarged by their delay ranges) are executed one after the other in a way that a packet that is assigned a delay δ in some frame F waits at its first edge in F for (additional) δ steps, and then moves on without waiting until it traverses its last edge in F (see Figure 5).

We concentrate in the following on some fixed frame F. Let each packet choose an additional delay out of the range $\Delta_2 = [2I_1]$. Hence, the length of the resulting schedule for this frame is at most $3I_1$. We use the Lovász Local Lemma to show that if the delays are chosen independently and uniformly at random, then with nonzero probability the congestion in any I_2-interval is at most I_2. We again associate a bad event with each edge. The bad event for edge e is that more than I_2 packets use e in some I_2-interval.

We first bound the dependence of these events. Whether or not a bad event occurs solely depends on the delays assigned to the packets that pass through the corresponding edge. Since at most I_1 packets pass through an edge in any frame and each of these packets passes through at most I_1 other edges, the dependence of the bad events is at most I_1^2.

Next we bound the probability that a bad event occurs. Consider some fixed edge e and I_2-interval J. Let the packets traversing e in F be numbered from 1 to $m \leq I_1$. For every $i \in \{1, \ldots, m\}$, let the binary random variable X_i be one if and only if packet i traverses e during J. Let $X = \sum_{i=1}^{m} X_i$. Since the packets choose their delays uniformly at random from a range of size $2I_1$, we get $\mathbf{Pr}[X_i = 1] \leq I_2/(2I_1)$ for all $i \in \{1, \ldots, m\}$. Hence,

$$\mathbf{E}[X] = \sum_{i=1}^{m} \mathbf{E}[X_i] \leq I_1 \cdot \frac{I_2}{2I_1} = I_2/2 \,.$$

Together with the Chernoff bounds we therefore get with $\epsilon = 1$ that

$$\mathbf{Pr}[X \geq I_2] = \mathbf{Pr}[X \geq (1+\epsilon)I_2/2] \leq \mathrm{e}^{-I_2/6} \leq \mathrm{e}^{-3(\ln I_1 + 1)} = (\mathrm{e} \cdot I_1)^{-3} \,.$$

Since for each edge there are at most $3I_1$ many I_2-intervals to consider, the probability that a bad event occurs for edge e is bounded by

$$p \leq 3I_1 \cdot (\mathrm{e} \cdot I_1)^{-3} < \frac{1}{\mathrm{e}(I_1^2 + 1)}$$

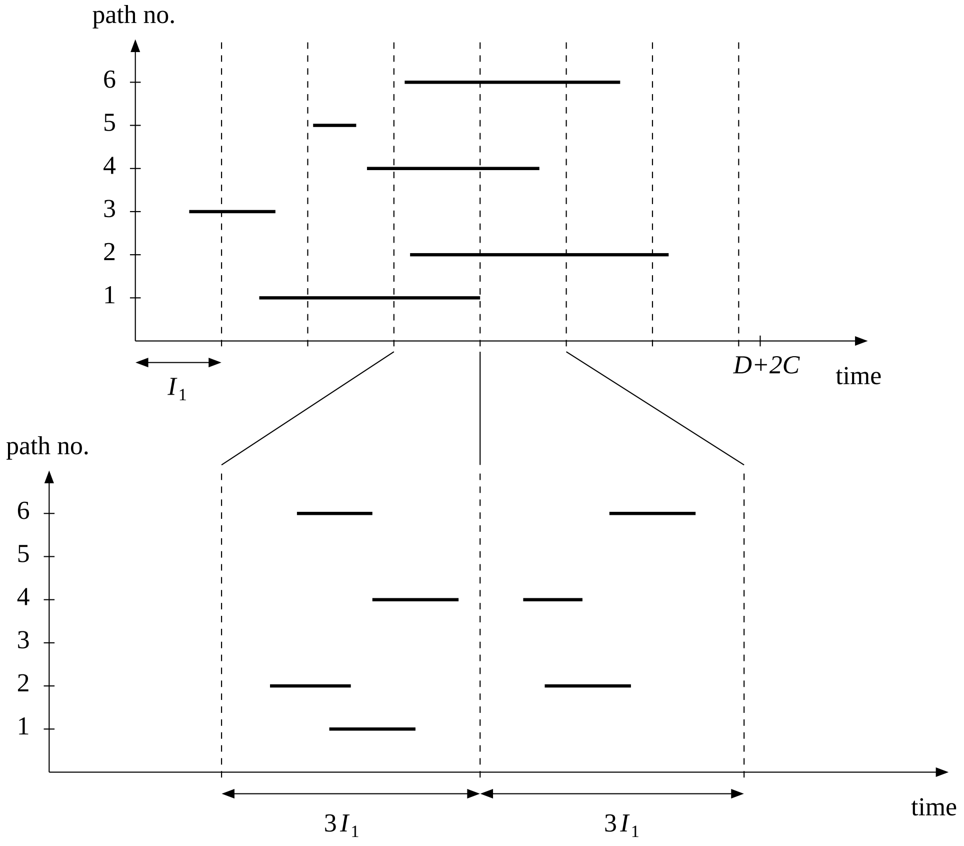


Figure 5: Refinement from schedule S_1 to schedule S_2.

for $I_1 \geq 18$. Thus, $ep(b+1) \leq 1$ and therefore, by the Lovász Local Lemma, there is some assignment of delays such that the congestion in each I_2-interval is bounded by I_2.

We continue to refine each I_2-frame in F recursively until we reach a round k in which $I_{k+1} \geq I_k/18$, i.e. I_k is a constant. We end up with a schedule S_k with total length at most

$$(D+2C)\prod_{j=1}^{k-1} 3 = (D+C) \cdot 2^{O(\log^*(C+D))}$$

and with a congestion C_k in each I_k-interval of at most I_k. In order to obtain a valid schedule, we simply schedule each I_k-frame in schedule S_k in a greedy way. Since each I_k-frame has a constant congestion and dilation, we altogether end up with a routing schedule of runtime $(C+D) \cdot 2^{O(\log^*(C+D))}$. □

5 Online Switching

We saw in the previous section that for every collection of simple paths there is a schedule to send packets along these paths, one packet per path, so that in at most $O(C + D)$ time steps all packets reach their destinations. The question we want to investigate in this section is whether a similar runtime can also be achieved for the online case. By "online" we mean that the system has no global knowledge about the routing problem. At the beginning, every processor only knows the packets that start in it (and, maybe, a few global parameters such as the congestion of the path collection). A processor can only use the information it receives during the routing. Thus, the processors may have do base their switching decisions on imcomplete information. We will show that nevertheless efficient switching can be done.

5.1 A Switching Protocol for Directed Acyclic Graphs

A *directed acyclic graph (or short DAG)* is a directed graph that has no directed cycle. We demand that any path chosen in a DAG for some routing problem must traverse its edges in the direction of their orientation, that is, it must be directed. We say that a DAG has a *depth* of D if the maximum number of edges a directed path can have in it is equal to D.

Suppose that we are given a collection of n paths in a DAG. Along each of these paths a packet has to be sent. Suppose w.l.o.g. that these packets are numbered from 1 to n. (Any numbering that ensures that no two packets have the same number would suffice.) The *random rank protocol* sends the packets along their paths in the following way:

At the beginning, every packet i chooses uniformly and independently at random an integer r_i out of some range $[K]$ (K will be determined later). Let the *rank* of packet i be defined as $\text{rank}(i) = r_i + \frac{i}{n+1}$. These ranks are used in the following way to resolve conflicts among the packets.

> For every time step and every edge with nonempty buffer, select the packet with minimum rank waiting at it and send it along that edge.

5.2 Analysis of the Switching Protocol

The following time bound has been shown by Leighton, Maggs, Ranade, and Rao.

Theorem 5.1 *Suppose we are given an arbitrary path collection $\mathcal{P}$ of size n and congestion C in a DAG of depth D. Let $K \geq 8C$. Then the random rank protocol needs at most $O(C + D + \log n)$ time steps, with high probability, to send all the packets to their destinations.*

Proof. Suppose that the runtime of the random rank protocol is equal to $T \geq D + S$. We want to show that it is very improbable that S is large. For this we need to find a structure that witnesses a large S. This structure should become more and more unlikely to exist the larger S becomes. The structure we are looking for will be the result of the following backwards argument that is similar to the argument used in Section 1.

Let p_1 be some packet that arrived at its destination v_0 in step T. We follow p_1 backwards in time until we reach an edge e_1 where it was delayed the last time before it reached its destination. Let us denote the length (i.e., the number of edges) of the path from the destination of p_1 to e_1 (inclusive) by ℓ_1 and the packet that delayed p_1 by p_2. Starting from e_1, we now follow p_2 backwards in time until we reach an edge e_2 where it was delayed by some other packet. We call this packet p_3 and denote the length of the path from e_1 (exclusive) to e_2 (inclusive) by ℓ_2. Afterwards, we follow p_3 backwards in time, and so on, until we arrive at a packet p_{s+1} that prevented the packet p_s at edge e_s from moving forward. (We will specify later how to choose s in relation to S.) The path from e_s to v_0 recorded by this process in reverse order is called *delay path* (see Figure 6). It consists of s contiguous parts of routing paths of length $\ell_1, \ldots, \ell_s \geq 0$ with $\sum_{i=1}^{s} \ell_i \leq D$, since no directed path can be longer than D. Because of the contention resolution rule it holds that $\text{rank}(p_i) > \text{rank}(p_{i+1})$ for all $i \in \{1, \ldots, s\}$. A structure that contains all of these features is defined as follows.

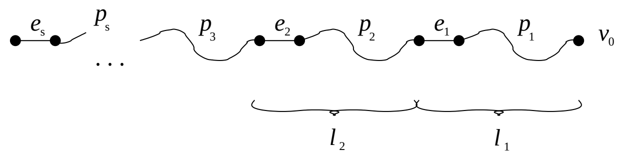


Figure 6: The structure of a delay path.

Definition 5.2 (delay sequence) *A* delay sequence of length s *consists of*

- s *not necessarily different* delay edges $e_1, \ldots, e_s$*;*
- $s+1$ delay packets $p_1, \ldots, p_{s+1}$ *such that the path of* p_i *traverses* e_i *and* e_{i-1} *in that order for all* $i \in \{2, \ldots, s\}$*, the path of* p_{s+1} *contains* e_s*, and the path of* p_1 *contains* e_1*;*
- s *integers* $\ell_1, \ldots, \ell_s \geq 0$ *such that* ℓ_1 *is the number of edges in the path of* p_1 *from* e_1 *(inclusive) to its destination, for all* $i \in \{2, \ldots, s\}$ ℓ_i *is the number of edges in the path of* p_i *from* e_i *(inclusive) to* e_{i-1} *(exclusive), and* $\sum_{i=1}^{s} \ell_i \leq D$*; and*
- $s+1$ *integers* $r_1, \ldots, r_{s+1}$ *with* $1 \leq r_{s+1} \leq \ldots \leq r_1 \leq K$*.*

A delay sequence is called active *if*

1. *for all* $i \in \{1, \ldots, s\}$ *we have* $\text{rank}(p_{i+1}) < \text{rank}(p_i)$ *and*
2. *for all* $i \in \{1, \ldots, s+1\}$ *we have* $\lfloor \text{rank}(p_i) \rfloor = r_i$*.*

Now, how does s relate to S? The following lemma gives an answer to this.

Lemma 5.3 *If the runtime of the random rank protocol is at least $D+s$, then there must exist an active delay sequence of length s.*

Proof. Suppose the random rank protocol needs $T \geq D+s$ steps. Since for any delay sequence of length s we have $\sum_{i=1}^{s} \ell_i \leq D$, it holds that $T \geq \sum_{i=1}^{s} \ell_i + s$ and therefore $T - \sum_{i=1}^{s} \ell_i - s \geq 0$. In this case, the backwards argument we used above is guaranteed to end with a packet p_{s+1} that delayed p_s at time step $T - \sum_{i=1}^{s}(\ell_i+1)+1 \geq 1$, which implies that there must be such a packet p_{s+1}. Due to the contention resolution rule, the packets must fulfill requirement (1) of a delay sequence to be active. Requirement (2) can be fulfilled by simply setting $r_i = \lfloor \text{rank}(p_i) \rfloor$. $\square$

Thus, if no active delay sequence of length s can be constructed from the given path collection and the ranks chosen by the packets, then the runtime of the random rank protocol can be at most $D+s$. Hence, all we need to do is to bound the probability that it is possible to construct a delay sequence of length s. For this we need the following two lemmata.

Lemma 5.4 *The number of different delay sequences of length s is at most*

$$n \cdot C^s \cdot \binom{D+s}{s} \cdot \binom{s+K}{s+1} .$$

Proof. There are at most $\binom{D+s}{s}$ possibilities to choose the ℓ_i such that $\sum_{i=1}^{s} \ell_i \leq D$. Furthermore, there are n packets from which p_1 can be chosen. Since p_1 and ℓ_1 determine the edge e_1 and the congestion at e_1 is at most C, there are at most C possibilities to choose packet p_2. The same holds for the packets $p_3, \ldots, p_{s+1}$ at the edges $e_2, \ldots, e_s$. Hence, we altogether have at most $\binom{D+s}{s} \cdot n \cdot C^s$ possibilities to choose the delay packets. Finally, there are at most $\binom{s+K}{s+1}$ ways to select the r_i such that $1 \leq r_{s+1} \leq \ldots \leq r_1 \leq K$. $\square$

Lemma 5.5 *No packet can appear twice in an active delay sequence.*

Proof. Since the ranks of the packets do not change during the routing, the lemma follows directly from requirement (1) for a delay sequence to be active. $\square$

Because the packets choose their ranks independently at random, Lemma 5.5 implies that the probability that some fixed delay sequence of length s is active (or more precisely, that requirement (2) for a delay sequence to be active is fulfilled) is at most $1/K^{s+1}$. Thus,

$$\begin{aligned}
&\mathbf{Pr}[\text{The random rank protocol needs at least } D+s \text{ steps}] \\
&\overset{\text{Lemma 5.3}}{\leq} \quad \mathbf{Pr}[\text{there exists an active delay sequence of length } s] \\
&\overset{\text{Lemmata 5.4, 5.5}}{\leq} \quad n \cdot C^s \cdot \binom{D+s}{s} \cdot \binom{s+K}{s+1} \cdot \frac{1}{K^{s+1}} \\
&\leq \quad n \cdot C^s \cdot 2^{D+s} \cdot 2^{s+K} \cdot \frac{1}{K^{s+1}} \\
&\leq \quad n \cdot 2^{2s+D+K} \cdot \left(\frac{C}{K}\right)^s
\end{aligned}$$

If we set $K \geq 8C$ and $s = K + D + (\alpha+1)\log n$, where $\alpha > 0$ is an arbitrary constant, then

$$\begin{aligned}
&\mathbf{Pr}[\text{The random rank protocol needs at least } D+s \text{ steps}] \\
&\leq \; n \cdot 2^{2s+D+K} \cdot 2^{-3s} = n \cdot 2^{-s+D+K} = \frac{1}{n^\alpha}
\end{aligned}$$

which concludes the proof of Theorem 5.1. □

References

[BA91] Marc Baumslag and Fred Annexstein. A unified framework for off-line permutation routing in parallel networks. *Mathematical Systems Theory*, 24:233–251, 1991.

[Ben64] V. Beneš. Permutation groups, complexes, and rearrangeable multistage connecting networks. *Bell System Technical Journal*, 43:1619–1640, 1964.

[Ben65] V. Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffic.* Academic Press, New York, NY, 1965.

[Bol98] Béla Bollobás. *Modern Graph Theory.* Springer-Verlag, Heidelberg, 1998.

[Fig89] Michael Figge. *Permutationsrouting auf hochdimensionalen Gittern.* Diplomarbeit, Universit"at Dortmund, 1989.

[HHL88] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18:319–349, 1988.

[LMR88] Tom Leighton, Bruce M. Maggs, and Satish B. Rao. Universal packet routing algorithms. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 256–271, 1988.

[LMRR94] Tom Leighton, Bruce M. Maggs, Abhiram G. Ranade, and Satish B. Rao. Randomized routing and sorting on fixed connection networks. *Journal of Algorithms*, 17:157–205, 1994.

[LPV81] Gavriela F. Lev, Nicholas Pippenger, and Leslie G. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Transactions on Computers*, 30:93–100, 1981.

[NS82] David Nassimi and Sartaj Sahni. Parallel algorithms to set up the Benes permutation network. *IEEE Transactions on Computers*, 31:148–154, 1982.

[Par80] D. Parker. Notes on shuffle/exchange-type switching networks. *IEEE Transactions on Computers*, C-29:213–222, 1980.

[Vöc01] B. Vöcking. Almost optimal permutation routing on hypercubes. In *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC)*, pages 530–539, 2001.

[Wak68] A. Waksman. A permuting network. *Journal of the ACM*, 15:159–163, 1968.