Routing with Bounded Buffers and Hot-Potato Routing in Vertex-Symmetric Networks*

Friedhelm Meyer auf der Heide and Christian Scheideler

Department of Mathematics and Computer Science
and Heinz Nixdorf Institute
University of Paderborn
33095 Paderborn, Germany

Abstract. In this paper we present and analyze on-line routing schemes with contant buffer size and hot-potato routing schemes for vertex-symmetric networks. In particular, we prove that for *any* vertex-symmetric network with n vertices, degree d, and diameter $D = \Omega(\log n)$, a randomly chosen function and any permutation can be routed in time

- $O(\log n \cdot D)$, with high probability (w.h.p.), if constant size buffers are available for each edge,
- $O(\log n \cdot D \log^{1+\epsilon} D)$ for any $\epsilon > 0$, w.h.p., if for each vertex buffers of size 3, independent of the degree of the network, are available.

The schedule for the second result can be converted into a hot-potato routing schedule, if a self-loop is added to each vertex.

E.g., for any bounded degree vertex-symmetric network with self-loops and diameter $O(\log n)$ (among them expanders) we obtain a hot-potato routing protocol that needs time $O(\log^2 n (\log\log n)^{1+\epsilon})$ for any $\epsilon > 0$ to route a randomly chosen function and any permutation, w.h.p..

Our protocols also allow bounds on the space requirements for vertices and packets in the network: we show that $O(D(\log logD + \log d))$ space suffices for storing routing information in the vertices and $O(\log D)$ space suffices for storing routing information in the packets.

This is the first result about space-efficient routing where both the buffer size and the space for storing routing information is strongly bounded. Previous results are only known about routing protocols that either can reduce the buffer size or the space for storing routing information. For space-efficient hot-potato routing no general results are known.

In order to prove the results above we introduce a new off-line routing protocol for arbitrary networks which is fast even for vertex buffers of size 1. This bound can not be reached by any other non-trivial off-line routing protocol yet.

* email: {fmadh,chrsch}@uni-paderborn.de, Fax: +49-5251-603514. Supported in part by DFG-Sonderforschungsbereich 1511 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen", by DFG Leipniz Grant Me872/6-1, and by the Esprit Basic Research Action Nr 7141 (ALCOM II)

1 Introduction

Packet routing schedules have extensively been studied for a wide range of networks and routing strategies. Whereas much is known about the runtime under the condition that enough space for storing routing tables and an unbounded buffer size is available (see, e.g., [MV95]), little is known about how space restrictions such as bounded buffers or space bounds for storing routing tables influences the runtime. For hot-potato routing, even without space restrictions little is known so far about the routing time in networks.

Hot-potato routing or deflection routing is a variant of packet routing where the packets are always moving, i.e., they are treated as hot potatos. In each step a processor must send out all packets it received at the beginning of the step. The advantage of hot-potato routing is that packets are not stored between time steps as in the store-and-forward routing model. Thus buffers are not required except for input and output buffers in each vertex. Input buffers contain the packets before they start to move, output buffers contain them after they have reached their destinations. Packets in transit are never stored in buffers. This, together with a space-efficient design of rules how to move the packets forward, keeps the hardware cheap and routing cycles very fast. Because of these reasons hot-potato routing is especially useful for optical networks [AS92, M89, S90], where buffering would involve the packets to be stored in electronic media.

In this paper we present on-line routing schemes with bounded buffer size and hot-potato routing schemes, and analyze the time and space they need on vertex-symmetric networks. These schemes use a new space-efficient routing strategy based on simulations of networks (see [MS95]), and a new off-line routing protocol. This off-line protocol seems to be of independent interest.

1.1 Routing with Bounded Buffer Size

The *routing network* is represented by a connected graph $H = (V, E)$, where $V = [n]$ is the set of all vertices (or processors) and $E \subset V \times V$ is the set of all edges (or links) in H. Each $\{v, w\} \in E$ consists of two links, one in each direction.

Each vertex has one *input buffer* and one *output buffer*. Initially, the packets are stored in the input buffers. If a packet reaches its destination, it is stored in the output buffer. Packets in transit are stored in so-called *transit* buffers. We call a transit buffer *edge buffer* if it is attached to an edge and *vertex buffer* if it is attached to a vertex.

We only consider *oblivious* routing strategies, i.e., a packet with origin u and destination v has to travel along a prescribed *routing path* $p(u, v)$ in H. The set of these paths for all $\binom{n}{2}$ pairs $\{u, v\}$ of vertices in H is called a *path system* and denoted by $\mathcal{P}$. A *shortest path system* contains only paths $p(u, v)$ that are shortest paths from u to v in H. Clearly, any shortest path has to be *vertex-simple*, that is, it uses no vertex more than once.

A *packet* consists of a *source* $u \in V$, a *destination* $v \in V$, *routing information*, and a *message*. The source and destination need $\log n$ bits. Throughout this paper we restrict the routing information to be very small, namely of length at most $O(\log n)$. We assume the messages to have uniform length.

Given a path system $\mathcal{P}$ in H, a *routing protocol* consists of a *contention resolution protocol* and a *routing structure* for all vertices in H.

The contention resolution protocol controls how long incoming packets have to wait in a vertex before they are sent to the next vertex on their paths. It has to ensure that

no two packets *collide* during the routing, that is, try to use the same edge at the same time.

The edge along which a packet has to be sent is determined with the help of a *routing structure* stored in v. This is a (static) data structure that, given the destination and the routing information of a packet, enables v to compute the next edge the packet has to use w.r.t. its path prescribed in $\mathcal{P}$, and (maybe) update the packet's routing information. We demand that this access needs constant time, i.e. a constant number of operations on $\log n$-bit words.

Routing is performed in synchronous rounds. In a *round*, each vertex uses its contention resolution protocol to choose one packet to forward. Then it uses its routing structure to compute the outgoing edge the packet has to go. If the buffer on the edge to be used is full, the packet is returned to the buffer it came from, otherwise its routing information is updated and the packet is forwarded.

Clearly, the following parameters greatly influence the time needed to route a function $f : V \to V$ in H:

- the *dilation* D of $\mathcal{P}$, that is, the length of the longest path in $\mathcal{P}$,
- the *congestion* C, i.e. the maximum number of routing paths $p(u, f(u))$ in $\mathcal{P}$ that pass through the same vertex in H, and
- the *buffer size* B available at edges or vertices to store packets.

In routing schemes with bounded buffers two events may occur that prevent the scheme from terminating: *deadlocks* and *livelocks*. A deadlock appears if a set of packets is not able to move forward any more. Livelocks occur if a set of packets is routed in such a way that they mutually deflect each other in an infinite loop from which they never recover. As we will see, our routing protocols are constructed in a way that deadlocks and livelocks can not appear.

1.2 Hot-Potato Routing

The hot-potato routing model differs from the routing model for bounded buffers only in three aspects.

- Hot-potato routing strategies do not need any transit buffers.
- Each time step, a vertex may receive and send out several packets, at most one packet per edge.
- Since packets can not be buffered, the contention resolution protocol now decides which packets to forward on their routing path, and which to route to other directions. Our protocol will make sure that a packet is never more than one edge away from its routing path.

1.3 Vertex-Symmetric Networks

In this paper we only deal with routing schemes for vertex-symmetric networks. This class is defined as follows.

Definition 1. A graph $H = (V, E)$ is called *vertex-symmetric* if for any pair u, v of vertices in H there exists an automorphism $\varphi : V \to V$ mapping u to v such that for the graph $H_\varphi = (V, E_\varphi)$ with $E_\varphi = \{\{\varphi(x), \varphi(y)\} \mid \{x, y\} \in E\}$ it holds $H_\varphi = H$.

Vertex-symmetric networks form a very general class and include most of the standard networks such as the d-dimensional torus, the butterfly, the hypercube, etc.. Furthermore, the best expanders that have an explicit construction are all Cayley graphs and therefore vertex-symmetric (see, e.g., [M94]).

The goal of this paper is to find a path system and a routing protocol for any vertex-symmetric network such that a small buffer size and little space suffices for storing routing structures in the vertices and routing information in the packets while still maintaining a fast routing time.

1.4 Previous Results

In the following the term "*with high probability*" (w.h.p.) means "with probability of at least $1 - \frac{1}{n^{-\alpha}}$", where n is the number of vertices and α is any constant.

We start with results on off-line routing. It is not difficult to see that $(C \cdot D + 1)D$ rounds suffice to route f using a hot-potato protocol. This upper bound follows from the fact that a packet may collide along its prescribed path with at most $C \cdot D$ other packets. A simple graph coloring argument yields that $C \cdot D + 1$ routing phases, each taking time D, suffice to guarantee that no two packets collide. In [LMR88] an off-line routing protocol is presented that works for networks with constant size edge buffers. (The required buffer sizes are not explicitly computed, but it seems they have to be fairly large.) They show that $O(C + D)$ time suffices to route all packets. On the other hand, if there is at least one edge that transmits C packets and one packet that traverses $O(D)$ vertices then packet routing takes $\Omega(C + D)$ time, even if arbitrarily large buffers are allowed.

Let us now turn to on-line routing. If no restrictions are imposed on space requirements then, according to [MV95], it holds for arbitrary networks with diameter D that any function f with congestion C can be routed on-line in time $O(D + C + \log n)$, w.h.p.. Their results can be used to prove that, for all vertex-symmetric networks with diameter $D = \Omega(\log n)$ and degree d, a randomly chosen function can be routed in time $O(D)$, w.h.p., if $O(n \cdot D \cdot \log d)$ space is available in each vertex and routing information of length $O(\log n)$ is available in each packet. This result was generalized by [MS95] where it is shown, e.g., that for every $s \in [2, n]$, a random function can be routed in time $O(\log_s n \cdot D)$ if $O(s \cdot D \cdot \log d)$ space is available in each vertex and $O(\log(s \cdot D))$ space is available for storing routing information in each packet. Unfortunately, all on-line protocols mentioned above need edge buffers of size $O(\log n)$, w.h.p..

In [LMRR94] it is shown that for any bounded-degree leveled network with depth L, any set of n packets whose paths have congestion C can be routed in time $O(C + L + \log n)$, using 1 buffer per edge. Leighton and Rao furthermore prove in [LR88] that for any network with flux α, any CRCW PRAM algorithm can be simulated with delay $O(\frac{\log^2 n}{\alpha})$, w.h.p., using a sufficiently large constant number of buffers per edge. This very general result has the drawback that the diameter of a network can be by a factor of $\log n$ lower than $\frac{\log n}{\alpha}$. In this paper, instead, we directly consider the diameter of a network.

Experimental results on simulations of hot-potato routing on various networks are documented in [AS92, GH92, M89]. In several of these papers a probabilistic analysis of simple protocols is presented, but various independence assumptions are made to make the analysis tractable.

Feige and Raghavan [FR92] present a simple deterministic hot-potato routing protocol that routes a random function on the $(n, 2)$-torus in $2n + O(\log n)$ steps, w.h.p..

Furthermore they present a simple deterministic routing protocol that routes a random function on the d-dimensional hypercube in $O(d)$ steps, w.h.p..

In [MW95] a probabilistic hot-potato routing protocol is presented that routes a random function on the (n,d)-torus in $dn + O(d^3 \log n)$ steps, w.h.p., if $d = O(n^\epsilon)$ with $0 < \epsilon < \frac{1}{2}$.

Apart from the Butterfly network, the torus, and the hypercube no other non-trivial results for hot-potato routing on vertex-symmetric networks are known so far.

1.5 New Results

Our main result is an upper bound for the routing time and space requirement for various space-efficient routing schemes on vertex-symmetric networks. In particular, we prove:

Main Theorem: *Let $H = (V, E)$ be any connected vertex-symmetric network with n vertices, degree d, and diameter $D = \Omega(\log n)$. Then a randomly chosen function and any permutation can be routed in time*

- *$O(\log n \cdot D)$, w.h.p., if sufficiently large, constant size vertex buffers and $O(D \log d)$ space per vertex are available,*
- *$O(\log n \cdot D \log^{1+\epsilon} D)$ for any $\epsilon > 0$, w.h.p., if vertex buffers of size 3 and $O(D(\log\log D + \log d))$ space per vertex are available.*

The schedule for the second result can be converted into a hot-potato routing schedule if a self-loop is added to each vertex. For all these schedules, $O(\log D)$ space is sufficient for storing routing information in each packet, w.h.p..

If H has degree two then an efficient hot-potato routing scheme is straightforward, since H must be a ring. So in the following we will only deal with vertex-symmetric networks with degree at least three.

We now give a short summary of the techniques used to derive the Main Theorem. Our approach is 'Routing via Simulation' introduced in [MS95]. For this purpose we will embed a network $G = (V, R)$ in H that is based on a Butterfly network. We will describe it precisely in Section 3. For this network we can prove the following result with a variant of Ranade's protocol (see [R91] or [L92]).

Vertex buffers of size 2 suffice to route a randomly chosen function in G using time $O(\log n)$, w.h.p..

Consider a network $H = (V, E)$. Fix a shortest path system $\mathcal{P}_R^H$ in H which contains shortest paths $p_H(u, v)$ in H only for pairs $(u, v) \in R$. Our routing strategy will simulate routing in G by routing in H:

Suppose, a packet with origin u and destination v travels along the path $p_G(u, v)$ in G. In order to simulate the traversal of an edge $(x, y) \in R$, it chooses the path $p_H(x, y)$.

Our way to implement routing with bounded buffer size in H will be to use off-line routing schemes for simulating one routing step in G. If we are satisfied with larger (still constant) bounds on vertex buffers, we may use the off-line protocol from [LMR88] mentioned in Section 2.

For vertex buffers of very small size or hot-potato protocols, we need a new off-line protocol. We prove the following result which may be of independent interest.

For any set of packets with vertex-simple paths having congestion C and dilation D, there is an off-line protocol for vertex buffers of size one that needs time

$$O((D + C\log(C + D))(\log\log\log(C + D))^{1+\epsilon})$$

for any $\epsilon > 0$ to route all packets.

Let S_n be the set of all permutations on V. For a permutation $\pi \in S_n$ let $\pi \circ R = \{\{\pi(u), \pi(v)\} \mid \{u, v\} \in R\}$. We change the definition of the congestion C in a way that C denotes the maximum number of paths in a shortest path system $\mathcal{P}^H_{\pi \circ R}$ that cross each other at a vertex. Then, according to [MS95], it holds:

Let $H = (V, E)$ be defined as in the Main Theorem and $G = (V, R)$ be defined as in Section 3. Then there is an embedding $\pi \in S_n$ of G into H and a shortest path system $\mathcal{P}^H_{\pi \circ R}$ such that the congestion C is $O(D)$.

Therefore routing a random function f in H needs, w.h.p., at most time

- $O(\log n \cdot D)$ with vertex buffers of sufficiently large, constant size,
- $O(\log n \cdot D \log^{1+\epsilon} D)$ for any $\epsilon > 0$ with vertex buffers of size 3,
- $O(\log n \cdot D \log^{1+\epsilon} D)$ for any $\epsilon > 0$ in hot-potato mode if each vertex has a self-loop.

Using the Valiant-Brebner paradigm, it is not difficult to show that this result leads to protocols for routing arbitrary permutations in H within the same time bounds by sending the packets first to randomly chosen destinations before they are sent to those destinations prescribed by the permutation. Finally, using techniques similar to those in [MS95], we obtain the space bounds mentioned in the Main Theorem.

1.6 Organization of the Paper

The following section contains our new off-line routing protocol. Section 3 uses this protocol to establish a routing scheme with buffer size 3 per vertex and a hot-potato routing scheme for arbitrary networks. Finally, Section 4 contains space bounds for the contention resolution protocol, routing structures, and routing information that hold for all vertex-symmetric networks.

2 Off-line Routing Schemes

In this section we will present two off-line routing schemes. The following theorem follows directly from a result in [LMR88] (see also [LMR94]) by changing their model in a way that we consider vertices instead of edges.

Theorem 2. *For any set of packets with vertex-simple paths having congestion C and dilation D, there is an off-line schedule for vertex buffers of constant size that needs $O(C + D)$ time to route all packets.*

In [LMR88] and [LMR94] the buffer size is not explicitly computed, but it seems that it has to be fairly large, caused by the last of a sequence of refinements of schedules to develop this off-line schedule. In the following we present an off-line routing scheme which only needs one buffer for each vertex.

Theorem 3. *For any set of packets with vertex-simple paths having congestion C and dilation D, there is an off-line schedule for vertex buffers of size 1 that needs time*

$$O((D + C\log(C+D))\log\log(C+D))(\log\log\log(C+D))^{1+\epsilon})$$

for any $\epsilon > 0$ to route all packets.

Proof: Before proceeding, we need to introduce some notation. Let V be the set of all vertices in the network and $\mathcal{P}$ be a path system consisting of all paths the packets use to reach their destinations. Consider a schedule S for routing the packets. Let a path $P \in \mathcal{P}$ of length ℓ be represented as a sequence $((v_1,t_1),\ldots,(v_\ell,t_\ell)) \in (V \times \mathbb{N})^\ell$ of vertices and time steps the vertices of P are reached by the packet using P in S. We define $\mathcal{A}_k^S$ to be the collection of all paths $((v'_1,t'_1),\ldots,(v'_k,t'_k))$ for which there is a path $((v_1,t_1),\ldots,(v_\ell,t_\ell)) \in \mathcal{P}$ and an i such that $((v'_1,t'_1),\ldots,(v'_k,t'_k)) = ((v_i,t_i),\ldots,(v_{k+i-1},t_{k+i-1}))$ and $t_i,\ldots,t_{k+i-1}$ are consecutive time steps. Let $\mathcal{F}_k^S$ be the collection of paths $((v'_1,t'_1),\ldots,(v'_k,t'_k))$ in $\mathcal{A}_k^S$ that furthermore fulfill $t'_1 = j \cdot k$ for an integer j. A *T-frame* consists of T consecutive time steps. For a fixed $A \in \mathcal{A}_k^S$ and T-frame F, the *frame congestion* $C_{A,F}^S$ is defined as

$$C_{A,F}^S = \text{number of packets that traverse a vertex in } A \text{ within frame } F \text{ in } S \ .$$

Our strategy for constructing an efficient schedule is to make a succession of refinements to an initial schedule S_0, in which each packet moves at every step until it reaches its destination. The proof uses the Lovász Local Lemma [AES92, p.55] at each refinement step.

Lemma 4 (Lovász Local Lemma). *Let $A_1,\ldots,A_n$ be a set of "bad" events in an arbitrary probability space. Suppose that each event A_i is mutually independent of a set of all the other events A_j but at most b, and that $Pr(A_i) \le p$. If $ep(b+1) < 1$ then, with probability greater than zero, no bad event occurs.*

During these refinements, we choose suitable vertices for the packets to wait at. These vertices are called *secure* vertices. The secure vertices are selected in such a way that no two secure vertices are direct neighbors on a path of a packet. So if a packet decides to move forward along its path it can do so without violating the restriction of one buffer per vertex or moving waiting packets too far away from their secure vertices, even if all secure vertices are occupied. This can be done by simply exchanging packets if a packet wants to enter a secure vertex with a packet waiting at it and, as soon as the packet moves on, moving the waiting packet back to its secure vertex.

The first step of our refinement is to assign an initial delay to each packet. The delays are chosen from the range $[1,\alpha_1 C]$, where α_1 will be determined later. In the resulting schedule, S_1, a packet that is assigned a delay t waits in its starting vertex for t steps, then moves along its prescribed path without waiting until it enters its destination vertex. The time S_1 needs is at most $D + \alpha_1 C$. Let $\gamma_i = 4^{\log^*(C+D)-(i-1)}$ for any $i \ge 1$, $T_1 = 32\gamma_1\log(C+D)$, and $T_i = 32\gamma_i \log T_{i-1}$ for any $i > 1$. We use the Lovász Local Lemma to show that if the delays are chosen randomly, independently, and uniformly, then with nonzero probability the frame congestion for any $A \in \mathcal{A}_{T_1}^{S_0}$ for some fixed frame of size $3T_1T_2$ is less than $\frac{T_1}{8\gamma_1}$. Thus, such a set of delays must exist.

To apply the Lovász Local Lemma, we associate a bad event with each path $A \in \mathcal{A}_{T_1}^{S_0}$. The bad event for A is that at least $\frac{T_1}{8\gamma_1}$ packets traverse the vertices in A in some fixed $3T_1T_2$-frame defined later. To show that there is a way of choosing the delays so

that no bad event occurs, we need to bound the dependence b among the bad events and the probability p that a bad event occurs.

Calculating the dependence is straightforward. Whether or not a bad event occurs solely depends on the delays assigned to the packets that pass through a path A. Thus, the bad events for paths A and A' are independent unless some packet passes through a vertex in A and a vertex in A'. Clearly, at most $C \cdot T_1$ packets pass through A, and each of these packets passes through at most D other vertices. Since at most C packets pass through each of these vertices, there are at most $C \cdot T_1 \cdot D \cdot C \cdot T_1$ sets $A' \in \mathcal{A}_{T_1}^{S_0}$ that depend on A. Thus the dependence b of the bad events is at most $D(C \cdot T_1)^2$.

It remains to compute the probability that a bad event occurs. Let p be the probability of the bad event corresponding to path A for a fixed $3T_1T_2$-frame defined later. Then

$$p \leq \binom{T_1 \cdot C}{\frac{T_1}{8\gamma_1}} \left(\frac{3T_1T_2}{\alpha_1 C}\right)^{\frac{T_1}{8\gamma}} .$$

This expression is derived as follows for the case that there is no packet that moves through more than one vertex in A.

- There are $\binom{T_1 \cdot C}{\frac{T_1}{8\gamma_1}}$ ways to select $\frac{T_1}{8\gamma_1}$ packets out of at most $T_1 \cdot C$ that move through vertices in A.
- The probability that a packet crosses a vertex within the $3T_1T_2$-frame is at most $\frac{3T_1T_2}{\alpha C}$.

It is not difficult to show that this upper bound for p is also an upper bound for the situation that there are packets that move through several vertices in A. Clearly, by choosing $\alpha_1 \geq 24e\gamma_1T_1T_2$ it holds that $ep(b+1) < 1$. So, according to the Lovász Local Lemma, the packets can be given delays in such a way that no bad event occurs. Using these delays we obtain a schedule S_1.

We now want to assign a secure vertex to each path $A \in \mathcal{F}_{T_1T_2}^{S_1}$ in such a way that there is no $A \in \mathcal{F}_{T_1T_2}^{S_1}$ for which the distance between secure vertices is smaller than γ_1. With the help of the Lovász Local Lemma we will show that such an assignment of secure vertices indeed exists.

For each $A \in \mathcal{F}_{T_1T_2}^{S_1}$, decompose A into T_2 disjoint paths $A_1, \ldots, A_{T_2} \in \mathcal{F}_{T_1}^{S_1}$ of length T_1. Each of these paths A_i chooses randomly and independently a candidate for the secure vertex of A. To apply the Lovász Local Lemma, we associate a bad event with each path $A \in \mathcal{F}_{T_1T_2}^{S_1}$. The bad event for A is that none of the candidates chosen by $A_1, \ldots, A_{T_2}$ fulfills the requirement that all other candidates chosen by paths $A' \in \mathcal{F}_{T_1}^{S_1}$ that intersect A within the $3T_1T_2$-frame chosen for the A_i in A have a distance of at least γ_1 from this candidate.

Whether or not a bad event occurs solely depends on the choices of those paths $A' \in \mathcal{F}_{T_1}^{S_1}$ that intersect A. Thus the dependence b of the bad events is at most $\frac{T_1}{8\gamma_1} \cdot T_1T_2$.

It remains to calculate the probability that a bad event occurs. Let p be the probability of the bad event corresponding to path A. Then

$$p \leq \left((2\gamma_1 + 1) \cdot \frac{T_1}{8\gamma_1} \cdot \frac{1}{T_1}\right)^{T_2} .$$

This expression is derived as follows. On a fixed path there are at most $2\gamma_1 + 1$ vertices at distance γ_1 from the chosen candidate. For each of these vertices within distance γ_1, at most $\frac{T_1}{8\gamma_1}$ other paths run through them for the fixed $3T_1T_2$-frame. The probability that one of these paths decides to choose this vertex as its candidate is $\frac{1}{T_1}$. Clearly,

it holds that $ep(b+1) < 1$. So, according to the Lovász Local Lemma, there exists a secure vertex for every $A \in \mathcal{F}^{S_1}_{T_1T_2}$. This completes the design of schedule S_1.

The idea behind refining schedule S_1 is to cut the paths the packets use in S_1 in pieces at the secure vertices such that each piece lies within two consecutive T_1T_2-frames starting with time step iT_1T_2 for an integer $i \geq 0$. Let schedule S_1^1 be a part of schedule S_1 consisting of the first two consecutive T_1T_2-frames and all pieces of paths lying within them, as shown in the picture.

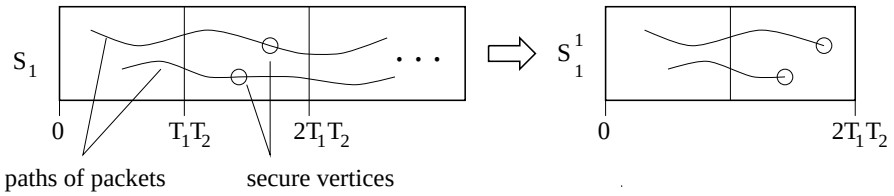

Figure 1: Construction of schedule S_1^1

We refine schedule S_1^1 in the following way. Those packets that have a piece of their path in S_1^1 choose additional delays from the range $[1, \alpha_2 T_1 T_2]$, where α_2 will be determined later. In the resulting schedule S_2^1 a packet that is assigned a delay t_1 in S_1^1 and t_2 in S_2^1 waits in its starting vertex for $t_1 + t_2$ steps, then moves along its prescribed path without waiting until it enters its secure vertex. Thus the time S_2^1 needs is at most $(2 + \alpha_2)T_1T_2$. We again use the Lovász Local Lemma to show that if the new delays are chosen randomly, independently, and uniformly then, with nonzero probability, the frame congestion for any path $A \in \mathcal{A}^{S_1^1}_{T_2}$ for some fixed frame of size $3T_2T_3$ is less than $\frac{T_2}{8\gamma_2}$. Thus, such a set of delays must exist.

To apply the Lovász Local Lemma, we associate a bad event with each path $A \in \mathcal{A}^{S_1^1}_{T_2}$. The bad event for A is that at least $\frac{T_2}{8\gamma_2}$ packets traverse the vertices in A in some $3T_2T_3$-frame. To show that there is a way of choosing the delays so that no bad event occurs, we again need to bound the dependence b among the bad events and the probability p that a bad event occurs.

Whether or not a bad event occurs depends solely on the delays assigned to the packets that pass through a vertex set A. Thus, the bad events for paths A and A' are independent unless some packet passes through a vertex in A and a vertex in A'. Clearly, at most $\frac{T_1}{8\gamma_1}$ packets pass through A, and each of these packets passes through at most $2T_1T_2$ other vertices within the first two T_1T_2-frames. Since at most $\frac{T_1}{8\gamma_1}$ packets pass through each of these vertices, there are at most $\frac{T_1}{8\gamma_1} \cdot 2T_1T_2 \cdot \frac{T_1}{8\gamma_1} \cdot T_2$ sets $A' \in \mathcal{A}^{S_1^1}_{T_2}$ that depend on A. Thus the dependence b of the bad events is at most $T_1^3 \cdot T_2^2$.

It remains to compute the probability that a bad event occurs. Let p be the probability of the bad event corresponding to path A for a fixed $3T_2T_3$-frame. Then

$$p \leq \binom{\frac{T_1}{8\gamma_1}}{\frac{T_2}{8\gamma_2}} \left(\frac{T_2 \cdot 3T_2T_3}{\alpha_2 T_1 T_2}\right)^{\frac{T_2}{8\gamma_2}} .$$

This expression is derived in a similar way as above. Note that, since each of the $\frac{T_1}{4\gamma_1}$ packets may move through all vertices in A, we get a probability of at most $\frac{T_2 \cdot 3T_2T_3}{\alpha_2 T_1 T_2}$.

Clearly, by choosing $\alpha_2 \geq \frac{3e}{2}T_3$ it holds that $ep(b+1) < 1$. Thus according to the Lovász Local Lemma, the packets can be given delays in such a way that no bad event occurs. Using these delays we obtain a schedule S_2^1.

We now want to assign a secure vertex to each path $A \in \mathcal{F}_{T_2T_3}^{S_2^1}$ in such a way that there is no $A \in \mathcal{F}_{T_2T_3}^{S_2^1}$ for which the distance between secure vertices is smaller than γ_2. With the help of the Lovász Local Lemma we will show that such an assignment of secure vertices indeed exists.

For each $A \in \mathcal{F}_{T_2T_3}^{S_2^1}$, decompose A into T_3 disjoint paths $A_1, \ldots, A_{T_3} \in \mathcal{F}_{T_2}^{S_2^1}$. Each of these paths A_i chooses randomly and independently a candidate for the secure vertex of A. To apply the Lovász Local Lemma, we associate a bad event with each path $A \in \mathcal{F}_{T_2T_3}^{S_2^1}$. The bad event for A is that none of the candidates chosen by $A_1, \ldots, A_{T_3}$ fulfills the requirement that all other candidates chosen by paths $A' \in \mathcal{F}_{T_2}^{S_2^1}$ that intersect A within the $3T_2T_3$ frame chosen for the A_i in A have a distance of at least γ_2 from this candidate.

Whether or not a bad event occurs solely depends on the choices of those paths $A' \in \mathcal{F}_{T_2}^{S_2^1}$ that intersect A. Thus the dependence b of the bad events is at most $\frac{T_2}{8\gamma_2} \cdot T_2T_3$.

It remains to calculate the probability that a bad event occurs. Let p be the probability of the bad event corresponding to path A. Then

$$p \leq \left((2\gamma_2 + 1) \cdot \frac{T_2}{8\gamma_2} \cdot \frac{1}{T_2} + \frac{2\gamma_2 + 1}{\gamma_1} \right)^{T_2} .$$

This expression is derived as follows. On a fixed path there are at most $2\gamma_2 + 1$ vertices at distance γ_2 from the chosen candidate. For each of these vertices within distance γ_2, at most $\frac{T_2}{8\gamma_2}$ other paths run through them for the fixed $3T_2T_3$-frame. The probability that one of these paths decides to choose this vertex as its candidate is $\frac{1}{T_2}$. Furthermore, the probability that a new secure vertex come to close to an old secure vertex is at most $\frac{2\gamma_2+1}{\gamma_1}$. Clearly, it holds that $ep(b+1) < 1$. So, according to the Lovász Local Lemma, there exists a secure vertex for every $A \in \mathcal{F}_{T_2T_3}^{S_2^1}$. This completes the design of schedule S_2 for the first two T_1T_2-frames in S_1.

Applying this calculation to all other consecutive pairs of T_1T_2-frames in schedule S_1 we get a schedule S_2^i for every $2T_1T_2$-frame starting at time iT_1T_2 for an integer i. In order to build a schedule S_2 for our routing problem out of these schedules S_2^i, we paste S_2^i and S_2^{i+1} together by letting all packets that are active in both schedules wait at their secure vertices w.r.t. schedule S_1. Note that, if the $3T_1T_2$-frames used for developing schedule S_1 are chosen appropriately, during schedule S_2^i and S_2^{i+1} no vertex is used twice as secure vertex since the two consecutive pairs of T_1T_2-frames, S_2^i and S_2^{i+1} are built of, together cover $3T_1T_2$ time steps in schedule S_1.

The refinements are continued recursively using a stretch factor of $\alpha_i \geq \frac{3e}{2}T_i$, until $T_i \geq T_{i-1}$, that is, T_i is a constant. At that stage a simple graph coloring argument yields a schedule S_i that needs time

$$\begin{aligned} &O((D + \alpha_1 C) \cdot (2 + \alpha_2) \cdot \ldots \cdot (2 + \alpha_{\log^*(C+D)})) \\ &= O((D + C\log(C+D)\log\log(C+D))(\log\log\log(C+D))^{1+\epsilon}) \end{aligned}$$

for any $\epsilon > 0$. □

3 Routing with Bounded Buffers and Hot-Potato Routing in Arbitrary Networks

In this section we will show how efficient routing with bounded buffers and hot-potato routing can be done in arbitrary networks H with n vertices. Our strategy will be to find a suitable 1-1 embedding of the following graph G into H.

Definition 5. For $n \geq 4$ let $d = \max\{i \in \mathbb{N} \mid i \cdot 2^i \leq n\}$. Let $G = (V, R)$ be a network with vertex set

$$V = \{(\ell, x) \mid \ell \in \{0, \ldots, \lfloor n/2^d \rfloor\} \wedge x = (x_{d-1}, \ldots, x_0) \in \{0,1\}^d \wedge 2^d \cdot \ell + x < n\},$$

and edge set R defined in the following way.

- For each $\ell \in [d]$, every vertex $(\ell, x) \in V$ is connected with (ℓ', x) and $(\ell', (x_{d-1}, \ldots, x_{\ell+1}, 1, x_{\ell-1}, \ldots, x_0))$ where $\ell' = \ell + 1$ if $2^d(\ell+1) + x < n$ and 0 otherwise.
- For each ℓ, $d \leq \ell \leq \lfloor n/2^d \rfloor$, each vertex $(\ell, x) \in V$ is connected with $(\ell + 1, x)$ if $2^d(\ell+1) + x < n$ and $(0, x)$ otherwise.

G is chosen in such a way that it consists of a Butterfly network connected back-to-back by linear arrays of length at most $\log n + 2$. A picture will clarify this for $n = 22$.

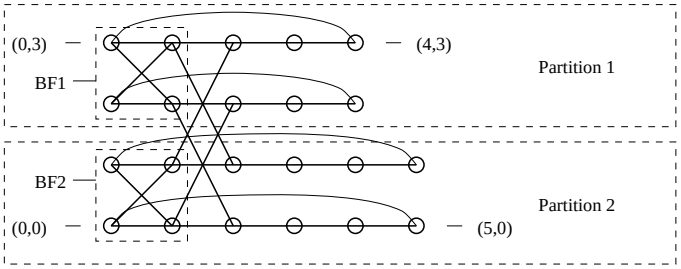

Figure 2: Network G with 22 vertices

Suppose each vertex in G wants to send a packet stored in its input buffer to a random destination in G. We will route these packets in two phases.

- **Phase 1:** Route all packets in Partition 1 to their destinations in two subphases, each using the Ranade protocol. During the first subphase only those packets become active that have their random destinations in Partition 1. The picture below clarifies how the packets have to be routed in this phase.

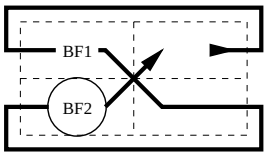

Figure 3: The first subphase

The first time the packets move through BF1 only those edges connecting vertices of different rows are used. In BF2 the packets are routed to the rows of their random destinations. When packets move back from BF2 to Partition 1, only edges connecting vertices of the same row are used afterwards.
During Subphase 2 only those packets become active that have their random destinations in Partition 2. In this case the packets in Partition 1 are first moved to Partition 2 and then routed to their destinations analogous to Subphase 1.
The analysis of Ranade's protocol in [R91] can easily be modified such that each edge in G only needs one buffer to ensure that, w.h.p., after $O(\log n)$ rounds Phase 1 is completed. (Note that the input and output buffers do not count as buffers here.)

– **Phase 2:** Route all packets stored in Partition 2 in a similar way as done for Partition 1 in Phase 1.

So altogether two buffers at each vertex in G suffice to route the packets in G according to a randomly chosen function f with the above scheme in $O(\log n)$ rounds, w.h.p..

Let $G = (V, R)$ be embedded 1-1 into H via a permutation $\pi \in S_n$. Fix a shortest path system $\mathcal{P}^H_{\pi \circ R}$ in H. Suppose now we want to simulate the Ranade protocol on G by H. Then we can simulate each routing step of G on-line by using the off-line protocol for routing packets along all paths in $\mathcal{P}^H_{\pi \circ R}$. So H needs two buffers per vertex for simulating the protocol described above, and one buffer per vertex to be able to use our off-line protocol presented in Theorem 3 for simulating one routing step in G. From this we can conclude the following theorem.

Theorem 6. *Fix a 1-1 embedding of G into H. Let C denote its congestion and D its dilation. Then a random function can be routed in time*

– *$O(\log n \cdot (C + D))$, w.h.p., if vertex buffers of sufficiently large constant size are available;*
– *$O(\log n \cdot (D + C) \log^{1+\epsilon}(C + D))$ for any $\epsilon > 0$, w.h.p., if vertex buffers of size 3 are available.*

If we add a self-loop to each vertex we can easily coordinate the second routing protocol in Theorem 6 in such a way that the $3n$ buffers can be simulated by the edges of H. (Note that each edge consists of two links capable of transporting one packet per time step.) Thus the following result holds.

Corollary 7. *The schedule for the second result in Theorem 6 can be converted into a hot-potato routing schedule, if every vertex in H has degree at least 3 and a self-loop.*

In order to route an arbitrary permutation we subdivide the n packets into α sets of size $\frac{n}{\alpha}$ for sufficiently large α. For each of these sets, the packets are first sent to randomly chosen rows before they are sent to their destinations. Note that the packets can be evenly distributed among the vertices of their randomly chosen rows using the fact that for sufficiently large α it holds w.h.p. that for any row the number of packets that randomly chose it is at most the number of vertices in the row. Thus, using the Valiant-Brebner paradigm, the following corollary holds.

Corollary 8. *The schedules presented in Theorem 6 can be used to obtain a routing protocol that routes any permutation in H in time*

– *$O(\log n \cdot (C + D))$, w.h.p., if vertex buffers of sufficiently large constant size are available;*
– *$O(\log n \cdot (D + C) \log^{1+\epsilon}(C + D))$ for any $\epsilon > 0$, w.h.p., if vertex buffers of size 3 are available.*

4 Space-Efficient Routing Structures and Contention Resolution Protocols

In this section we will present a space-efficient routing structure and contention resolution protocol for our routing schemes. In particular, we prove the following theorem.

Theorem 9. *Let H be defined as in the Main Theorem. Then the routing protocols for Theorem 6 can be implemented in such a way that*

(1) space $O(D \log d)$ for storing routing structures in a vertex,
(2) space $O(\log D)$ for storing routing information in a packet,
(3) space $O(D)$ for the contention resolution protocol if the off-line protocol presented in [LMR88] is used,
(4) space $O(D \log\log D)$ for the contention resolution protocol if the off-line protocol in Theorem 3 is used,

suffice to route a random function f in H, w.h.p..

Proof: With a proof similar to the proof of Theorem 4.1 in [MS95] it can be shown that, for a suitable embedding π of G into H and a suitable path system $\mathcal{P}^H_{\pi \circ R}$ simulating the edges of G, a path system can be constructed for H that needs space $O(D \log d)$ in each vertex, and space $O(\log D)$ for storing routing information in a packet.

In remains to prove the space bounds for the contention resolution protocol. Let T_v denote a table that associates to each packet traversing v a number of steps it has to wait there before moving on.

The off-line routing protocol presented in [LMR88] is constructed in such a way that all packets that traverse v wait there for at most a constant number of time steps. The packet that starts at v may wait for at most $O(D)$ time steps. Thus T_v needs space $O(D)$.

According to the proof of the off-line schedule presented in Theorem 3, a packet may wait at most $O(D \log^{1+\epsilon} D)$ time steps for any $\epsilon > 0$ before it is started. Once a packet is started it may wait at a vertex for at most $O(\log^{1+\epsilon} D)$ time steps for any $\epsilon > 0$. Thus T_v needs space $O(D \log\log D)$. This proves the theorem. □

5 Conclusions

Very recently, we used the techniques described above to obtain a *deterministic* scheme that routes arbitrary permutations within the same time and space bounds as stated in the Main Theorem. This is achieved by an embedding of a variant of the Multibutterfly network (see [U92]) instead of the variant of the Butterfly network described in Definition 5.

6 Acknowledgements

We would like to thank Berthold Vöcking for several helpful comments on an early draft of the paper.

References

[AES92] N. Alon, P. Erdős, J. Spencer. *The Probabilistic Method.* Wiley Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, 1992.

[AS92] A. Acampora, S. Shah. Multihop Lightwave Networks: a Comparison of Store-and-Forward and Hot-Potato Routing. *IEEE Transaction on Communications*, pp. 1082-1090, 1992.

[FR92] U. Feige, P. Raghavan. Exact Analysis of Hot-Potato Routing. In *Proc. of the 33rd Symp. on Foundations of Computer Science*, pp. 553-562, 1992.

[GH92] A. Greenberg, B. Hajek. Deflection Routing in Hypercube Networks. *IEEE Transactions on Communications*, pp. 1070-1081, 1992.

[L92] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann, San Mateo, CA, 1992.

[LMR88] F.T. Leighton, B.M. Maggs, S.B. Rao. Universal Packet Routing Algorithms. In *Proc. of the 29th Ann. Symp. on Foudations of Computer Science*, pp. 256-271, 1988.

[LMRR94] F.T. Leighton, B.M. Maggs, A.G. Ranade, S.B. Rao. Randomized Routing and Sorting on Fixed-Connection Networks. *Journal of Algorithms* 17, pp. 157-205, 1994.

[LMR94] F.T. Leighton, B.M. Maggs, S.B. Rao. Packet Routing and Job-Shop Scheduling in O(Congestion + Dilation) Steps. *Combinatorica* 14, pp. 167-186, 1994.

[LR88] F.T. Leighton, S.B. Rao. An Approximate Max-Flow Min-Cut Theorem for Uniform Multicommodity Flow Problems with Applications to Approximation Algorithms. In *Proc. of the 29th Ann. Symp. on Foudations of Computer Science*, pp. 422-431, 1988.

[M89] N. Maxemchuk. Comparison of Deflection and Store-and-Forward Techniques in the Manhattan Street and Shuffle-Exchange Networks. In *Proc. of the IEEE INFOCOM*, pp. 800-809, 1989.

[M94] M. Morgenstern. Existence and Explicit Constructions of $q+1$ Regular Ramanujan Graphs for Every Prime Power q. *Journal of Comb. Theory, Series B* 62, pp. 44-62, 1994.

[MS95] F. Meyer auf der Heide, C. Scheideler. Space-Efficient Routing in Vertex-Symmetric Networks. To appear at *SPAA 95*.

[MV95] F. Meyer auf der Heide, B. Vöcking. A Packet Routing Protocol for Arbitrary Networks. In *Proc. of the 12th Symp. on Theoretical Aspects of Computer Science*, pp. 291-302, 1995.

[MW95] F. Meyer auf der Heide, M. Westermann. Hot-Potato Routing on Multi-Dimensional Tori. To appear at WG95 (21st Workshop on Graph-Theoretical Concepts in Computer Science), June 1995.

[R91] A.G. Ranade. How to Emulate Shared Memory. *Journal of Computer and System Sciences* 42, pp. 307-326, 1991.

[S90] T. Szymanski. An Analysis of Hot-Potato Routing in a Fiber Optic Packet Switched Hypercube. In *Proc. of the IEEE INFOCOM*, pp. 918-925, 1990.

[U92] E. Upfal. An O(log N) Deterministic Packet Routing Scheme. *Journal of the ACM* 39, 1992.

This article was processed using the LaTeX macro package with LLNCS style