Compact Oblivious Routing in Weighted Graphs

Philipp Czerner, Harald Räcke

{czerner, raecke}@in.tum.de
Department of Informatics, TU München, Germany

August 24, 2020

The space-requirement for routing-tables is an important characteristic of routing schemes. For the cost-measure of minimizing the total network load there exist a variety of results that show tradeoffs between stretch and required size for the routing tables. This paper designs compact routing schemes for the cost-measure congestion, where the goal is to minimize the maximum relative load of a link in the network (the relative load of a link is its traffic divided by its bandwidth). We show that for arbitrary undirected graphs we can obtain oblivious routing strategies with competitive ratio $\tilde{\mathcal{O}}(1)$ that have header length $\tilde{\mathcal{O}}(1)$, label size $\tilde{\mathcal{O}}(1)$, and require routing-tables of size $\tilde{\mathcal{O}}(\deg(v))$ at each vertex v in the graph.

This improves a result of Räcke and Schmid who proved a similar result in *unweighted* graphs.

1 Introduction

Oblivious routing strategies choose routing paths independent of the traffic in the network and are therefore usually much easier to implement than adaptive routing solutions that might require centralized control and/or lead to frequent reconfigurations of traffic routes. Because of this simplicity a lot of research in recent years has been performed on the question whether the quality of route allocations performed by oblivious algorithms is comparable to that of adaptive solutions (see e.g. [2, 5, 6, 17, 19–21, 26]). For some cost-metrics this is indeed the case. For example for minimizing the total traffic in the network (a.k.a. total load), shortest path routing is a simple optimal oblivious strategy. When one aims to minimize the congestion, i.e., the maximum (relative) load of a network link, one can still obtain strategies with a competitive ratio of $\mathcal{O}(\log n)$, i.e.,

the congestion generated by these strategies is at most an $\mathcal{O}(\log n)$-factor than the best possible congestion [21].

However, another important aspect for implementing oblivious routing strategies on large networks is the size of the required routing tables. This aspect has been investigated thoroughly for the cost-measure total load (see e.g. [7, 9, 11, 18, 24, 25, 29]), and various trade-offs between competitive ratio (also called stretch for the total load scenario) and the table-size have been discovered.

If for example every vertex stores the next hop on a shortest path to a target one can obtain a stretch of 1 at the cost of having routing tables of size $\mathcal{O}(n \log n)$ per node. If one allows non-optimal solutions Thorup and Zwick [25] have shown how to obtain a stretch of $4k - 5$ for any $k > 2$ with routing tables of size $\tilde{\mathcal{O}}(n^{1/k})$. This routing scheme works for the so-called labeled scenario in which the designer of the routing-scheme is allowed to relabel the vertices of the network in order to make routing decisions easier. Of course, there is still a restriction on the label-size as otherwise the power of being able to assign labels to vertices could be abused.

In the (more difficult) so-called *name-independent* model the designer is not allowed to relabel the vertices. Abraham et al. [1] have shown that for general undirected graphs one can asymptotically match the bounds for the labeled variant. They obtain a stretch of $\mathcal{O}(k)$ and routing tables of size $\tilde{\mathcal{O}}(n^{1/k})$. If a famous conjecture due to Erdős [8] about the existence of low-girth graphs holds then there is also a lower bound that says that obtaining a stretch better than $2k + 1$ requires routing tables of size $\Omega(n^{1/k})$. This means that for general undirected graphs the existing tradeoffs between stretch and space are fairly tight.

There exist many more results that analyze problem variants as e.g. obtained by restricting the graph representing the network (see e.g. [7, 10, 12, 13, 18, 24]); so the problem of designing compact routing schemes is very well studied for the cost-measure *total load*.

However, for the cost-measure congestion this is not the case. Räcke and Schmid [22] gave the first oblivious routing scheme that combines a guarantee w.r.t. the congestion with small routing-tables. They consider the labeled model and design an oblivious routing scheme that for a general undirected, unweighted graph G requires routing tables of size $\tilde{\mathcal{O}}(\deg(v))$ at each vertex v and obtains a competitive ratio of $\tilde{\mathcal{O}}(1)$ w.r.t. congestion.

There are important differences when comparing this result to its counter-parts for the total-load scenario. Firstly, the space used at a vertex v may depend on the degree of v. This is a reasonable assumption from a practical perspective as a node corresponds to a router in the network and it is reasonable to assume that the memory at a router (node) grows with the number of ports (number of incident edges). However, this assumption seems also crucial for getting any reasonable guarantees. In order to minimize congestion it is important to distribute the traffic among all network resources. It seems very difficult to do this if the routing table at a vertex is a lot smaller than the number of outgoing edges.

Another difference is that there is no tradeoff parameter k that gives a smooth transition from optimal routing with large tables to more compact routing. The reason is that for

congestion the competitive ratio may be $\Omega(\log n)$ even for unlimited routing tables [3].

One important shortcoming of the result by Schmid and Räcke [22] is that it only applies to unweighted graphs (there is a straightforward generalization that obtains routing tables of size $\mathcal{O}(W \operatorname{polylog} n)$ where W is the largest weight of an edge, but this is undesirable). This restriction is due to the fact that the result by Schmid and Räcke uses *paths* to route within well-connected clusters, which they obtain by randomized rounding.

There are two major obstacles in generalizing this result to the weighted case:

(1) In unweighted graphs, low congestion also ensures a low number of paths using an edge. However, an edge of weight W might be used by W small paths, which cannot be stored in a compact manner.

(2) Even within a well-connected cluster, it is not sufficient to route a commodity using a small number of paths, if the nodes are connected by many low-weight edges (illustrated in Figure 1). Hence a source node may have to route (and store) W small paths.

In this paper we give a construction of an oblivious routing scheme that avoids both problems, by storing aggregate routing information for many paths at once, as well as distributing storage across nodes for commodities that need to spread out over multiple paths. In this manner we obtain a polylogarithmic competitive ratio with polylogarithmic space requirement per edge in the network. Our main result is the following.

Theorem 12. *There is a compact oblivious routing scheme with competitive ratio at most $\mathcal{O}(\log^6 n \log^3 W)$, that uses routing tables of size $\mathcal{O}(\log^5 n \log W \log^3(nW) \cdot \deg(v))$ at a node v, packet headers of length $\mathcal{O}(\log^3(nW))$, and node labels of length $\mathcal{O}(\log^2 n)$.*

In particular this result shows that if we can route some demand in a network with a multicommodity flow f of congestion C, then it is possible to route the demands *space-efficiently*, i.e., one can set up small routing tables so that packets follow a (maybe) different flow f' that routes the same demands with a slightly worse congestion. This question of space-efficiently routing demands in a network is orthogonal to oblivious routing and it is not clear by how much the performance (i.e., the congestion) degrades because of the space-requirement. The above theorem gives a polylogarithmic upper bound but to the best of our knowledge this problem has not been studied before.

1.1 Further Work

Oblivious routing with the goal of either minimizing the total load (or stretch), minimizing the congestion or a combination of both is a well studied problem. The research started with deterministic algorithms and it was shown by Borodin and Hopcroft [5] that on *any* bounded degree graph G for *any* deterministic routing scheme there exists a permutation routing instance that incurs congestion $\Omega(\sqrt{n}/\Delta^{3/2})$. This result was improved by Kaklamanis et al. [15] to a lower bound of $\Omega(\sqrt{n}/\Delta)$. As there exist bounded degree

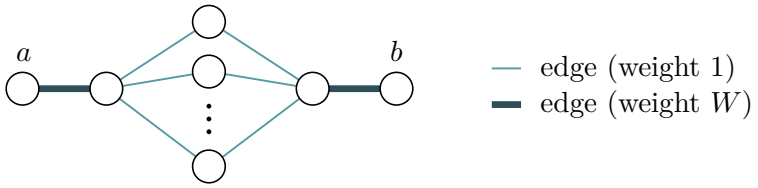

Figure 1: Routing a single demand over multiple edges. Sending data from a to b requires roughly W paths, but a has degree 1 and can store only $\tilde{\mathcal{O}}(1)$ bits.

graphs that can route any permutation with small congestion this gives a large lower bound on the competitive ratio of deterministic oblivious routing schemes.

For randomized algorithms Valiant and Brebner [28] showed how to obtain a polylogarithmic competitive ratio for the hypercube by routing to random intermediate destinations (known as Valiant's trick).

Räcke [20] presented the first oblivious routing scheme with a polylogarithmic competitive ratio of $\mathcal{O}(\log^3 n)$ in general undirected networks. This routing scheme is based on a hierarchical decomposition of a graph and forms the basis for the compact routing schemes that we construct in this paper. The construction in [20] was not polynomial time. This drawback was independently addressed by Bienkowski et al. [4] and Harrelson et al. [14]. Both papers give a polynomial-time algorithm for constructing the hierarchical decomposition (and, hence, the routing scheme)— the first with a competitive ratio of $\mathcal{O}(\log^4 n)$, and the second with a competitive ratio of $\mathcal{O}(\log^2 n \log\log n)$.

In 2014 Räcke et al. [23] presented another construction of the hierarchy that runs in time $\mathcal{O}(m \operatorname{polylog} n)$ and guarantees a competitive ratio of $\mathcal{O}(\log^5 n)$ (however, going from the hierarchy to the actual routing scheme may require superlinear time).

The above oblivious routing schemes that are based on hierarchical tree decompositions do not give the best possible competitive ratio. In [21] Räcke presents an oblivious routing scheme that is based on embedding a convex combination of trees into the graph G. This scheme obtains a competitive ratio of $\mathcal{O}(\log n)$, which is optimal due to a lower bound of Bartal and Leonardi for online routing in grids [3].

However, the number of trees that are used in the above result [21] is fairly large ($\Theta(m)$). Therefore, it seems difficult to design a compact routing scheme based on the tree embedding approach, and, therefore we use the earlier results that are based on hierarchical decompositions (a single tree!) but only guarantee slightly weaker competitive ratios.

1.2 Preliminaries

Throughout the paper we use $G = (V, E, w)$ to denote an undirected weighted graph with n node and m edges. We will refer to the weight of an edge also as the *capacity* of the edge. Wlog. we assume that the minimum edge weight is 1, that edge-weights are a power of 2, and that the largest edge weight is W. We call an edge of capacity/weight 2^i

a *class* i edge and use $N_{\text{class}} := 1 + \log_2 W$ to denote the total number of classes. Further, we use $\Gamma(v)$ to denote the neighborhood of a vertex v, i.e., $\Gamma(v) = \{u \in V \mid \{u, v\} \in E\}$.

The degree of a node v in the graph G will be referred to as $\deg_G(v)$, that is $\deg_G(v) := |\Gamma(v)|$. We apply that to directed graphs as well, where it refers to the number of outgoing edges.

While the edges E are undirected, it will be convenient to refer to a certain orientation of an edge, so we define $E_{\text{or}} := \{(u, v) \in V^2 : \{u, v\} \in E\}$. A mapping $f : E_{\text{or}} \to \mathbb{R}$ with $f((u, v)) = -f((v, u))$ for $(u, v) \in E_{\text{or}}$ is called a (single-commodity) flow. If $f(u, v) > 0$ for some edge $(u, v) \in E$, this indicates flow from u to v. The reverse flow of f is simply $-f$. For the sake of readability we omit double parentheses and write, e.g., $f(u, v)$ instead of $f((u, v))$.

A flow f may have multiple sources and sinks. The balance of a node $v \in V$ is denoted by $\text{bal}_f(v) := \sum_{u \in \Gamma(v)} f(u, v)$, so a positive balance indicates that the node is receiving more flow than sending out. A flow is *acyclic*, if there is no path $(p_0, ..., p_k)$ in G with $p_0 = p_k$ and $f(p_i, p_{i+1}) > 0$ for all i. Its congestion is the maximum ratio between the flow over an edge and its weight, denoted by $\text{cong}(f) := \max_{\{u,v\} \in E} |f(u, v)|/w(u, v)$. Given a multi-set of flows $F := \{f_1, f_2, ..., f_k\}$, its *total congestion* is $\text{cong}(F) := \max_{\{u,v\} \in E} \sum_k |f_k(u, v)|/w(u, v)$.

If a flow f has all flow originating at a single node s, i.e., $\text{bal}_f(s) \leq 0$ and $\text{bal}_f(u) \geq 0$ for $u \neq s$, we say that f is an s-flow. If additionally $\text{bal}_f(s) = -1$, we call f a *unit* s-flow. The set of all unit s-flows is denoted with $\text{flow}(s)$. If a flow f only sends from s to t, i.e., f is an s-flow and $-f$ is a t-flow we call f an s-t flow.

We use $\tilde{\mathcal{O}}$ to disregard logarithmic factors, so $g = \tilde{\mathcal{O}}(h)$ iff $g = \mathcal{O}(h \log^c(nW))$ for some constant c.

Oblivious Routing Scheme Now we define the concept of an oblivious routing scheme. The idea is to fix a single flow between each pair of nodes (u, v), and then multiply that flow with the actual demand from u to v to get the route. This flow can be interpreted probabilistically or fractionally, so if we have $f(e) = \frac{1}{2}$ for some edge e it means that the probability of the packet being routed across edge e is $\frac{1}{2}$; or that half a packet travels along that edge. We will use both interpretations interchangeably.

Definition 1. *An* oblivious routing scheme $S = (f_{u,v})_{u,v \in V}$ *consists of a unit u-v-flow for each pair of nodes $u, v \in V$. Given demands $d : V \times V \to \mathbb{R}_{\geq 0}$ the congestion of S w.r.t. d, denoted* $\text{cong}(S, d)$*, is the total congestion of the set of flows $\{d(u, v) f_{u,v} : u, v \in V\}$. The* competitive ratio *of S is* $\max_d \text{cong}(S, d)/\text{cong}_{\text{opt}}(d)$*, where* $\text{cong}_{\text{opt}}(d)$ *denotes the optimal congestion that can be obtained for demands d by any routing scheme.*

Defining a compact oblivious routing scheme formally is a bit more involved, as we have to clarify where information is stored and how it is used. Before we do so, we introduce the notion of a *routing algorithm*, which defines formally how packets are sent through the network. The intuition is that each packet carries a *packet header*, storing per-packet information. Each node stores a *routing table*, containing arbitrary information for the routing algorithm to use.

The routing algorithms forwards a packet in a local manner, meaning that it reads both the packet header and the routing table before choosing an outgoing edge on which to sent the packet. At the same time, it may modify the packet header. This procedure repeats, until routing algorithm indicates that the packet has reached its destination, by outputting no outgoing edge.

It remains to describe how the packet header is initialized. For oblivious routing schemes, we simply use the name of the target node as initial packet header. However, we will later define more general building blocks, namely transformation schemes. Hence a routing algorithm works with a set of abstract *input labels* as possible initial packet headers (and thus as input to the routing algorithm). For the purposes of a routing algorithm these are simply some set, but later definitions will describe their structure more concretely.

Definition 2. *A* routing algorithm $A = (\mathcal{A}, \mathcal{L}, \mathcal{T})$ *is a tuple,* $\mathcal{L} \subset \{0,1\}^*$ *denoting a finite set of* input labels *and* $\mathcal{T} : V \to \{0,1\}^*$ *a* routing table *for each node. Additionally,* $\mathcal{A} : \mathcal{T}(V) \times \{0,1\}^* \to (E \cup \{\emptyset\}) \times \{0,1\}^*$ *is a (possibly randomized) algorithm, taking both a routing table and a packet header as input, which calculates both the outgoing edge (if any) and the new packet header.*

We remark that the outgoing edge given by $\mathcal{A}$ has to be encoded in some manner, and it must be adjacent to the node the routing table belongs to. The routing table $\mathcal{T}(v)$ for a node v can contain information about v, such as a list of adjacent nodes, so any straightforward encoding, e.g., the index in this list, will work.

Given a routing algorithm $A = (\mathcal{A}, \mathcal{L}, \mathcal{T})$, a start vertex $v \in V$, and an input label $l \in \mathcal{L}$, the above mechanism defines a process for probabilistically distributing packets from v to targets in the network. We use $A(v, l)$ to denote a flow that describes the associated routing paths. This is defined as follows: We inject a packet at v, with l as packet header and execute $\mathcal{A}$ until no outgoing edge is returned. Then $A(v, l)(e)$ is the probability that the packet is routed over e (note that $\mathcal{A}$ may be randomized).

A routing algorithm $A = (\mathcal{A}, \mathcal{L}, \mathcal{T})$ is *compact*, if packet headers and input labels have size $\tilde{\mathcal{O}}(1)$, and the routing table of a node $v \in V$ has size $\tilde{\mathcal{O}}(\deg(v))$.

Recall that an oblivious routing scheme corresponds to a routing algorithm where the input labels are names of nodes in the graph. Consequently, we say that such a scheme is compact if its routing algorithm is compact.

Formally, we assign a name to each vertex in the graph, which we call *node label*, i.e., we have some function $\text{node} : V \to \{0,1\}^*$. For an oblivious routing scheme $S = (f_{u,v})_{u,v \in V}$ we use the set of all node labels $\text{node}(V)$ as input labels, so the initial packet header is the node label of the target node.

We say that S is *compact* if there exists a compact routing algorithm $A = (\mathcal{A}, \text{node}(V), \mathcal{T})$ with $f_{u,v} = A(u, \text{node}(v))$. This definition matches the one used by Räcke and Schmid [22], although it is more explicit.

The main result of this paper is the existence of a compact oblivious routing scheme, with competitive ratio $\tilde{\mathcal{O}}(1)$.

Transformation Schemes Our routing scheme will be composed out of several building blocks, which we call *transformation schemes*. Loosely speaking, they correspond to single-commodity flows which we are able to route.

We consider *distributions* or *weight functions* of the form $\mu : V \to \mathbb{R}_{\geq 0}$ that assign a non-negative weight to vertices in V. If we only specify a weight function on a subset $S \subseteq V$ we assume that it is 0 on $V \setminus S$. We use $\mu(S) := \sum_{v \in S} \mu(v)$ to denote the weight of a subset S, and $\mathbf{1}_v : V \to \mathbb{N}$ to denote the special weight function that has weight 1 on v and 0 elsewhere. For a distribution μ we use $\bar{\mu} := \frac{1}{\mu(V)}\mu$ to denote the corresponding *normalized distribution*.

Definition 3. *A* (compact[1]) transformation scheme (TS) *is a compact routing algorithm with a single input label.*

The above definition is not very useful by itself. The underlying idea is that we view a transformation scheme TS as on operation to transform one distribution of packets into another, by executing the routing algorithm. More precisely, given P packets each packet follows the flow $TS(v)$ at its source location $v \in V$. This will send it to some target node (probabilistically, according to $TS(v)$).

We say that a transformation scheme routes from some *input distribution* μ_{in} to an *output distribution* μ_{out}, if the above process transforms a set of P packets that are distributed according to $\bar{\mu}_{\text{in}}$ (i.e., a node v has $\mu_{\text{in}}(v)/\mu_{\text{in}}(V) \cdot P$ packets in expectation) into a set of packets that are distributed according to $\bar{\mu}_{\text{out}}$, i.e., afterwards a node has $\mu_{\text{out}}/\mu_{\text{out}}(V) \cdot P$ packets in expectation.

In addition we associate a *demand* $d(TS)$ and *congestion* $\text{cong}(TS)$ with a transformation scheme TS. We say a tranformation scheme routes demand $d(TS)$ from μ_{in} to μ_{out} with congestion $\text{cong}(TS)$ if the expected load on an edge e for the above process is at most $\text{cong}(TS) \cdot w(e)$ when $P = d(TS)$ (we allow P to be non-integral).

Note that, of course, the input for a transformation scheme could be any packet distribution. However, the congestion of the scheme is stated w.r.t. some fixed input distribution μ_{in} (its *natural input distribution*) and some total demand $d(TS)$.

From the congestion-value for μ_{in} and its demand d, one can then deduce the congestion-value for other inputs. If we, e.g., use the transformation scheme on a demand d' that is distributed according to ν we experience congestion at most $\max_{v \in V} d'\bar{\nu}(v)/d\bar{\mu}_{\text{in}}(v)$.

To make our notation more concise, we write a statement like "*TS routes μ_{in} to μ_{out} with demand d and congestion at most C*" as "*TS routes $\mu_{\text{in}} \xrightarrow{d} \mu_{\text{out}}$* with congestion (at most) C". We omit the demand d if it equals 1.

It may happen that for some transformation scheme TS we cannot exactly specify the output distribution that corresponds to its natural input distribution μ_{in}. We say TS routes $\mu_{\text{in}} \longrightarrow \mu_{\text{out}}$ with *approximation* σ if the real output distribution μ'_{out} fulfills $\bar{\mu}_{\text{out}}(v)/\sigma \leq \bar{\mu}'_{\text{out}}(v) \leq \bar{\mu}_{\text{out}} \cdot \sigma$.

Finally, in some proofs we will view packets as discrete entities and specify that the transformation scheme does not split them up.

[1] As all of our transformation schemes are compact (the later variants of deterministic and concurrent transformation schemes will be as well), we may drop the 'compact' when appropriate.

However, this collides with the fractional nature of the transformation scheme, which is caused by randomization. Therefore we introduce the following definition of a deterministic transformation scheme, that extracts this randomness and makes it explicit.

Definition 4. *A* (compact) deterministic transformation scheme $TS = (\mathcal{A}, \mathcal{P}(V), \mathcal{T})$ *is a compact routing algorithm where* $\mathcal{A}$ *is deterministic and* $\mathcal{P}(v) = \{1, ..., N_v\}$ *is a set of* path-ids *valid for a node* $v \in V$*.*

The idea of the above definition is that we can specify a "random seed" as input label, which will determine precisely how a packet is routed. The ordinary transformation scheme will correspond to choosing an input label u.a.r. from sets $\mathcal{P}(v)$. In this sense the above definition makes the random choices of a transformation scheme explicit.

Note that, technically, the definition of routing algorithms allows any path id in $\mathcal{P}(V)$ to be specified at a node v, not only the ones in $\mathcal{P}(v)$. We ensure that this does not occur.

As $\mathcal{A}$ is deterministic, the path id indeed determines the exact route a packet will take when starting at a certain node. More precisely, each flow $TS(v, l)$ for $v \in V, l \in \{1, ..., N_v\}$ is simply a path starting at v. Still, there is no guarantee that different path-ids send the packet to the same node.

We associate a transformation scheme with each deterministic TS, by choosing the path-id uniformly at random. In this fashion we extend the notions, such as congestion, input/output distributions, etc., that were defined above for ordinary transformation schemes to also cover deterministic transformation schemes.

Concurrent Transformation Schemes While a transformation scheme can mix packets arbitrarily, often we want to distribute several commodities at the same time, with separate input and output distributions for each commodity. This allows us to analyze the congestion more precisely and aggregate the routing information for different commodities. Hence we define the notion of a *concurrent transformation scheme*, which executes multiple transformation schemes in parallel.

The idea is that we take a transformation scheme and additionally specify a commodity as input.

Definition 5. *Let* I *denote a set of commodities. A* (compact) concurrent [deterministic] transformation scheme (CTS) *is a compact routing algorithm* $TS = (\mathcal{A}, I \times \mathcal{L}, \mathcal{T})$*, s.t.* $TS_i := (\mathcal{A}, \{i\} \times \mathcal{L}, \mathcal{T})$ *is a [deterministic] transformation scheme for each commodity* $i \in I$*.*

Note that transformation schemes have a single input label, in which case the $\mathcal{L}$ in the above definition is superfluous and the input to the concurrent transformation scheme is just the commodity. If it is deterministic, we need the path id as input, and $\mathcal{L}$ would be the set of possible path ids. Similar to before, any combination of commodity and path id may be specified at a node, according to the definition of a routing algorithm, but for our purpose only some of these make sense.

The congestion of such a concurrent transformation scheme is defined as follows. Let μ_i and d_i denote the input distribution and demand of TS_i, respectively. Let $X_i(e)$ denote

the expected load on an edge e if we execute TS_i on d_i packets distributed according to μ_i. The congestion of the CTS D is defined as $\text{cong}(D) := \max_e \frac{1}{w(e)} \sum_i X_i(e)$.

As an input to the routing algorithm, a commodity $i \in I$ has to be encoded in some fashion. Often, the commodity is determined by the source node (i.e., for each node v at most one input distributions μ_i is nonzero) and does not need to be specified. Otherwise, we will explicitly describe the necessary encoding as a property of the CTS.

Analogous to transformation schemes, we write "*TS routes* $\mu_i \xrightarrow{d_i} \nu_i$ with congestion (at most) C" for each commodity $i \in I$ for a CTS, and extend this notation to deterministic CTS by considering the associated transformation schemes.

2 Overview

In this section we give a high-level overview of the most important steps in our construction. The first part gives a rough outline of the general approach of routing along a decomposition tree that forms the basis for some oblivious routing schemes (e.g. [4,14,20]), and has also been used by Räcke and Schmid [22] to obtain compact routing schemes.

2.1 The Decomposition Tree

The result by Räcke and Schmid [22] as well as our extension of it use a decomposition tree, in particular the one described in [20]. We refer the reader to these for a more detailed description and just briefly mention the key ideas here. We start with a single cluster containing all nodes, and then further refine that until all clusters consist of just a single node. Hence we get a tree T where nodes are subsets of V, which we call *clusters*. The tree T has root V, i.e., the cluster containing all nodes, and leaves $\{v\}$ for each $v \in V$. For a cluster S with children $S_1, ..., S_r$ we have $S = S_1 \dot\cup ... \dot\cup S_r$, so the children are a partition of the parent. We use $\text{height}(T)$ to denote the maximum distance from any leaf to the root, and $\deg(T)$ to denote the largest number of children of any cluster.

Now we introduce a number of distributions, which will be important for routing within the decomposition tree. For any cluster S we define the *border-weight* $\text{out}_S : S \to \mathbb{N}$ by $\text{out}_S(v) := \sum_{u \notin S} w(v,u)$ for $v \in S$, counting the total weight of edges leaving the cluster adjacent to a node. Additionally, for any cluster S with child clusters $S_1, ..., S_r$ we define the *cluster-weight* $w_S : S \to \mathbb{N}$ as $w_S := \sum_i \text{out}_{S_i}$, which also counts edges between children of S. These distributions are shown schematically in Figure 2.

The decomposition from [20] has two essential properties:

- For each cluster S we can solve the multi-commodity flow problem with demands $d(u,v) := w_S(u)w_S(v)/w_S(S)$ for $u, v \in S$ with congestion $C \in \mathcal{O}(\log^2 n)$ *within* S, and
- the tree has logarithmic height, i.e., $\text{height}(T) \in \mathcal{O}(\log n)$.

The essential idea for oblivious routing is that in order to route between two nodes s and t in the graph we determine the path $\{s\} = S_1, S_2, \ldots, S_k = \{t\}$ in the tree and then we route a packet successively along this path by routing it from distribution $\bar{w}_{S_i}$

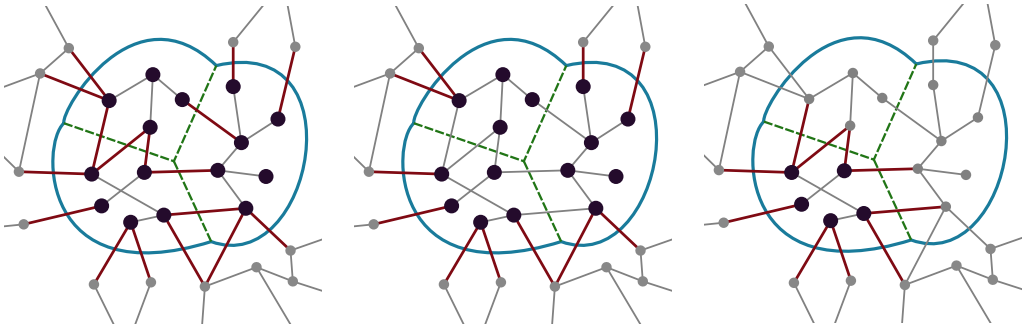

Figure 2: Distributions inside a cluster S with child S_i. The child clusters are separated by dashed lines. The respective distribution is nonzero only on the highlighted nodes and counts the total weight of highlighted edges adjacent to a node.

to distribution $\bar{w}_{S_{i+1}}$ for $i = 1, \ldots, k-1$. Note that distribution $\bar{w}_{S_1} = \bar{w}_{S_{\{s\}}} = \mathbf{1}_s$ and distribution $\bar{w}_{S_k} = \bar{w}_{S_{\{t\}}} = \mathbf{1}_t$, i.e., we indeed route from s to t.

Now suppose that the optimal congestion for the given demand d is $C_{\text{opt}}(d)$. How much demand does the above process induce for routing from w_{S_i} to w_S for a child-cluster S_i of some cluster S (for all packets)? Each packet that uses the tree edge (S_i, S) in its path has to leave the cluster S_i and thus create a load of 1 in out_{S_i}. Conversely, OPT has congestion $C_{\text{opt}}(d)$, i.e., a load of at most $\text{out}_{S_i}(S_i) \cdot C_{\text{opt}}(d)$ on out_{S_i}. Therefore the total demand that has to be routed for $w_{S_i} \longrightarrow w_S$ is at most $\text{out}_{S_i}(S_i) \cdot C_{\text{opt}}(d) = w_S(S_i) \cdot C_{\text{opt}}(d)$. An analogous argument holds for sending from w_S to w_{S_i}.

Now, we define a CTS for every cluster S that concurrently routes $w_{S_i} \xrightarrow{w_S(S_i)} w_S$ for all child-clusters with small congestion. If these schemes have congestion at most C then the overall competitive ratio of the compact oblivious routing scheme is $\text{height}(T)C$ as an edge is contained in at most $\text{height}(T)$ many clusters. Hence, we can restate our goal as follows. For every cluster S find

Mixing CTS
A CTS that routes $w_{S_i} \xrightarrow{w_S(S_i)} w_S$ for each child S_i, with congestion $\tilde{\mathcal{O}}(1)$.

Unmixing CTS
A CTS that routes $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ for each child S_i, with congestion $\tilde{\mathcal{O}}(1)$.

Here, we have to think about the encoding of the commodities, i.e., the indices of child clusters S_i. For our oblivious routing scheme we relabel the vertices so that the new name of a vertex v encodes the path from the root to the leaf $\{v\}$ in the decomposition tree. Then when we are given a packet with a source and a target node we can determine the path in the tree. For routing along an edge (S_i, S_{i+1}) of this path we extract the name of the child cluster and use this as commodity for the CTS. Furthermore, we will fix a specific name for each child cluster, incorporating a little bit of information for the

CTS. (As in the scheme by Räcke and Schmid [22], the name will be the index in the list of child clusters, sorted by size.)

2.2 Constructing Transformation Schemes

In this section we give an overview of the steps for constructing transformation schemes that for some cluster S route $w_{S_i} \xrightarrow{w_S(S_i)} w_S$ and $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ with small congestion. For this we use simplified versions of the main lemmata that are proven in the technical analysis in Section 3. We mark these simplified version with a "$'$", so Lemma $3'$ would be the simplified version of Lemma 3 in Section 3.

Single-commodity flows The first lemma that we show is how to construct a transformation scheme from a given flow, to route between *integral* distributions.

Lemma 1$'$ (Single-commodity flow routing)**.** *Let f denote a flow with congestion at most $\mathcal{O}(\text{poly}(nW))$, and μ, μ' integral distributions with $\mu' - \mu = \text{bal}_f$.*[2] *Then there exists a compact, deterministic transformation scheme that routes $\mu \xrightarrow{\mu(V)} \mu'$ with congestion $\mathcal{O}(\text{cong}(f))$ and has $N_v := \mu(v)$ valid path-ids at node v.*

This means that if we are given a flow then we can construct a transformation scheme that allows us to send packets from sources (outgoing net-flow) to targets (incoming net-flow). Note that there is no guarantee which target a packet will be sent to if the flow contains several targets.

Product multicommodity flow The second step of our approach is to obtain a concurrent transformation scheme that routes a *product multicommodity flow* (PMCF). Suppose that we are given a weight function $c : V \to \mathbb{N}$ on the vertices of the graph. We associate a multicommodity flow problem with this weight function by defining a demand $d(u, v) = c(u)c(v)/c(V)$ for any pair (u, v) of vertices. One can view this demand as each vertex u generating a flow of $c(u)$ and distributing it according to $\bar{c}$. Suppose that we can solve this multicommodity flow problem with some congestion $C = \mathcal{O}(\text{poly}(nW))$. We show that we can then obtain a CTS that routes a solution to the PMCF.

Lemma 4$'$ (PMCF-routing)**.** *Given a graph G together with a weight function $w : V \to \mathbb{N}$ on the vertices there exists a compact, deterministic CTS that routes $\mathbf{1}_u \xrightarrow{c(u)} c$ for each $u \in V$ with approximation $1 + \mathcal{O}(n^{-1})$ and congestion $\tilde{\mathcal{O}}(C)$.*

We obtain this result by making use of the KRV-framework [16]. One way to view this framework is that it tries to embed an expander into a graph by solving a small number of single-commodity maximum flow problems. Each maximum flow solution gives rise to a matching. One can then route to a random vertex by following the "*matching random walk*", i.e., in the i-th step the packet takes the (embedded) matching edge with probability 1/2.

[2] We remark that due to flow conservation, $\mu(V) = \mu'(V)$ holds.

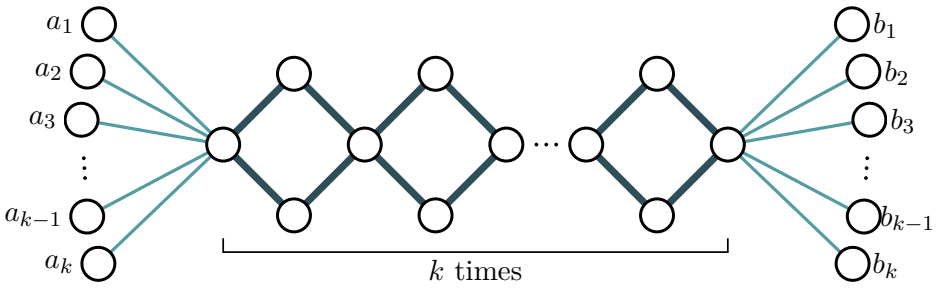

Figure 3: Routing multiple paths over a single edge. Each node a_i sends a packet to b_i. A single path needs k bits of information to encode, there are k paths and $n = \mathcal{O}(k)$ nodes, so on average each node needs to store $k^2/n = \Omega(n)$ bits if an arbitrary set of paths is chosen, which is too much. The same holds for paths chosen uniformly at random.

We proceed slightly differently. Instead of decomposing the flow into matchings and then route along the matchings (which seems difficult to do with small routing tables) we simply use Lemma 1′ to route along the flow. This means in the i-th step we stay with probability 1/2 or we route the packet along the flow to some target of the flow. More concretely, assume that the KRV-scheme uses a flow f between sources $S := \{v \in V \mid \text{bal}_f(v) < 0\}$ and targets $T := \{v \in V \mid \text{bal}_f(v) > 0\}$ in the i-th step. Then we construct two transformation schemes. Let

$$\mu(v) := \begin{cases} -\text{bal}_f(v) & v \in S \\ 0 & \text{otw.} \end{cases} \quad \text{and} \quad \mu'(v) := \begin{cases} \text{bal}_f(v) & v \in T \\ 0 & \text{otw.} \end{cases}$$

We use Lemma 1′ to construct a transformation scheme that routes $\mu \to \mu'$ and one that routes $\mu' \to \mu$. These then allow us to distribute packets in the described way. The guarantees of KRV still hold for this slightly modified scheme, which means that after performing a polylogarithmic number of such steps a packet is at a random location.

The above process and the transformation schemes for the individual iterations can be combined into a single concurrent transformation scheme. The id-sets M_v for this scheme contain bitstrings that encode for every iteration: (a) the id to be used in the transformation scheme for this iteration and (b) a bit that indicates whether to route along the flow or to stay. Note that the CTS is deterministic, i.e., after choosing the id the packet follows a fixed path in the network.

The result by Räcke and Schmid [22] also required a sub-routine for routing a product multicommodity flow. They used a randomized rounding approach on the multicommodity flow solution of the PMCF-instance, and crucially exploited the fact that the number of routing paths going through an edge in such a solution is $\tilde{\mathcal{O}}(C)$. As illustrated in Figure 3, we cannot do this in our scenario as the number of paths going through an edge might be as large as $\tilde{\mathcal{O}}(CW)$. Storing these paths would require large routing tables.

Routing arbitrary demands The PMCF-scheme described above routes $\mathbf{1}_u \xrightarrow{w(u)} c$. If we choose $c := w_S$ for some cluster S then this scheme gives us the first part of our goal: we can concurrently route $w_{S_i} \xrightarrow{w_S(S_i)} w_S$ with small congestion. To see this, observe that if we consider the demands in the PMCF-scheme for all commodities $u \in S_i$ combined, this is $\sum_u w_S(u)\mathbf{1}_u = w_S \restriction S_i = \text{out}_{S_i}$. We can go from w_{S_i} to out_{S_i} *within* S_i using Lemma 1′, and then the PMCF-scheme distributes it according to w_S.

Hence, by simply merging commodities $u \in S_i$ into one we obtain our desired CTS.

However, for our oblivious routing scheme we also need to be able to route commodities $w_S \xrightarrow{w_S(S_i)} w_{S_i}$. This turns out to be much more involved. Note that we cannot simply "route in reverse" because a transformation scheme is inherently directed.

We do not directly construct a transformation scheme that routes $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ but we first embed an auxiliary graph into the cluster S (via a transformation scheme). This auxiliary graph is directed and must have small degree for the embedding to be compact.

Lemma 9′ (general graph embedding)**.** *Let $G' = (S, A, d)$ denote a weighted, directed graph, where the total weight of incoming and outgoing edges of a node v is at most $w_S(v)$, and $\deg_{G'}(v) \in \tilde{\mathcal{O}}(\deg(v))$. Then there is a compact CTS that routes $\mathbf{1}_u \xrightarrow{d(u,v)} \mathbf{1}_v$ for each $(u, v) \in A$ with congestion $\tilde{\mathcal{O}}(C)$.*

This lemma is the main technical contribution of our work. A rough outline of the approach is as follows. The first observation is that one could combine the result for the PMCF-routing with Valiant's trick [27,28] of routing to random intermediate destinations. Suppose that we want to route from x to y. Then we first apply the PMCF-scheme of Lemma 4′; this brings us to a node z chosen according to w_S. At z we choose a path id id_y that brings us to y, i.e., if we apply the transformation scheme starting from node z with id id_y the packet is delivered to y. We choose id_y uniformly at random from all path ids that will deliver the packet to y. This applies Valiant's trick and the standard analysis shows that the congestion of this approach will be $\tilde{\mathcal{O}}(C)$.

However, implementing this approach with small routing tables is problematic. The node z could store a table of path ids which can be used for routing to y but this is clearly not compact.

If all edges have weight 1, we can apply a suitable randomized rounding to the above path generation method. Then the number of paths that go from x to y are just $\tilde{\mathcal{O}}(\deg(x))$. This allows the node x to store the necessary information for every path. In the weighted setting, however, the randomized rounding approach leads to $\tilde{\mathcal{O}}(W \cdot \deg(x))$ many paths. The resulting tables would not be compact.

Instead we proceed as follows. We say a path p is a *class l path* if the smallest capacity edge of p is from class l. (Recall that we assume all capacities to be powers of two, and that a class l edge is one with capacity 2^l.)

A pair (x, y) is from class l if l is the most frequent class that occurs when generating x-y-paths by the above process.

In a first step we change the path generation process to only use class l paths for a class l pair. This only increases the congestion by a factor of N_{class} (the number of classes). For a randomized rounding approach to guarantee a good congestion we need to spread the traffic between a class l pair (x, y) over roughly $k := d(x, y)/2^l$ many class l paths.

For this we split the packet into k different parts, each with a different path. However, we cannot store information for each such path in x directly, as k may be large. Instead, we identify k many other nodes, each of which we use to store the information for just a single path. Of course, we cannot simply pick any nodes in the cluster — they have to be reachable from x with low congestion.

As it turns out, the set of k class l paths from x to y already contains an appropriate choice. Each class l path contains a class l edge, and we pick one of its two adjacent nodes as helper node. Now observe that each path transports 2^l flow, so there can only be a small number of paths using that edge, because we have low congestion. That means that we can use $\tilde{\mathcal{O}}(1)$ space in the helper node, for each path that uses it.

Now that we have found k helper nodes that can store our routing information, the packets still have to reach these nodes, to pick up that information. So now we send a single-commodity flow from *all* source nodes x' of class l pairs to their helper nodes, and then back. We set up a TS for both directions of the flow, using Lemma 1.

Note that this does not guarantee that a packet from source node x reaches "its" helper node, but this is not required — it only needs to reach *a* node in which to store its routing information. Similarly, the packet may not get back to x, but end up at a different x'.

We have the same packet distribution as before, meaning every x' has the same number of packets, but possibly different ones. However, each packet has picked up $\tilde{\mathcal{O}}(1)$ of information while passing its helper node.

Suppose that all packets now in x have a target y' s.t. (x, y') is a class $l' \geq l$ pair. Then we can pick a single path for each of these packets, and store the information for that path in the helper node. (Recall that paths are generated by the PMCF-scheme of Lemma 4′, so we only need to store their path ids.)

If that is not the case we split the packet further, by applying the scheme recursively.

Hypercube embedding The previous lemma tells us that we can embed graphs with low degree. More precisely, we can embed any graph $G' = (S, A, d)$ which fulfills the following properties.

(1) The degree $\deg_{G'}(v)$ at a vertex is polylogarithmic.

(2) The capacity of incoming edges and the capacity of outgoing edges at a vertex is roughly equal to $w_S(v)$.

Now, in order to be able to construct an unmixing CTS, i.e., route $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ for a cluster S, we find some graph with the above properties where an unmixing CTS is easy to implement, and then embed that graph into G. In particular, we want a G' which has one additional property.

(3) There is a suitable numbering of the child clusters of S for which there exists a CTS for G' that routes $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ for each S_i with small congestion and commodity S_i encoded as integer i.

The result by Räcke and Schmidt [22] used a hypercube, where each node v received $w_S(v)$ hypercube ids. For weighted edges this would violate property (1).

Instead, we essentially embed several (disconnected) hypercubes, one for each class l. A node v then receives roughly $w_S^{(l)}(v)$ hypercube ids, at most one for each class l edge adjacent to it. The existence of a good CTS scheme for G' then follows from classical results about online routing on the hypercube [28].

3 Detailed Analysis

In this section we provide the details for constructing a mixing and an unmixing CTS for every cluster. This will then give the oblivious routing scheme.

Most of our lemmata related to the PMCF work for arbitrary weights c where the corresponding PMCF can be solved with congestion $C \in \mathcal{O}(\text{poly}(nW))$. We prove those without making further assumptions.

Note also that we can restrict the transformation schemes to route within a cluster: If a lemma is proven for arbitrary weights c, then it also works for weights w_S within the subgraph $G[S]$.

3.1 Constructing a mixing CTS

Single-commodity flows To get started, we show how to use a single-commodity flow to construct a transformation scheme. The key idea is that we decompose the flow into paths with unique identifiers and store intervals for each edge, to encode the outgoing paths.

Lemma 1 (Flow routing)**.** *Let f denote a flow with $\text{cong}(f) \in \mathcal{O}(\text{poly}(nW))$, and μ, μ' integral distributions with $\mu' - \mu = \text{bal}_f$. Then there exists a deterministic TS that routes $\mu \xrightarrow{\mu(V)} \mu'$ with congestion $\mathcal{O}(\text{cong}(f))$.*

The routing table of node v has size $\mathcal{O}(\deg(v) \log(nW))$, and packet headers have length $\mathcal{O}(\log(nW))$. At a node v there are $N_v := \mu(v)$ valid path ids.

Proof. First, we transform f to be integral and acyclic.

Let $F := \lceil \text{cong}(f) \rceil$. We consider the single-commodity flow problem where we add a source s, a sink t and edges (s, v), (v, t) for $v \in V$ with respective capacities $\mu(v)$ and $\mu'(v)$. We retain the other edges, scaling their capacities by F. All capacities are integral and f is a solution with flow value $\mu(V)$, so there is also an integral, acyclic solution f', which by construction has $\mu' - \mu = \text{bal}_{f'}$ and $\text{cong}(f') \leq F$.

For each source $v \in V$ (i.e., a node that has $k := -\text{bal}_{f'}(v) > 0$) we put k tokens into v. These have labels $a + 1, a + 2, ..., a + k$ where a is chosen s.t. the labels are disjoint for all nodes. We store a and k in v, which takes $\mathcal{O}(\log(nW))$ space.

Now we repeatedly move those tokens according to f', iteratively constructing the TS in the process. Each token represents a unit of flow.

As f' is acyclic, we iterate over the nodes based on the topological ordering given by f'. Let u denote the current node. We assume the invariant that no previous node contains any tokens, which is true in the beginning and will hold inductively for all other iterations.

Therefore, u has tokens precisely equal to the flow over incoming edges of u (plus $-\text{bal}_{f'}(u)$ if u is a source), which, due to flow conservation, is the same as the flow over outgoing edges (plus $\text{bal}_{f'}(u)$ if u is a sink). We distribute the tokens by sorting them and assign each outgoing edge (u, v) an amount of consecutive tokens, according to its flow $f'(u, v)$. These tokens are sent over that edge. Exactly $\max\{\text{bal}_{f'}(v), 0\}$ tokens remain, which we remove from the graph. Hence we deleted all tokens from u and added tokens only to its successors, so our invariant still holds.

After the last iteration, all nodes are empty, so each token has been routed. To construct a TS, we encode the path of all tokens, by storing an *interval* $I(u, v)$ of ids for each edge (u, v), which contains the tokens which were routed over that edge. Note that the interval may be larger than the number of tokens that use e; all tokens that traverse u and are inside $I(u, v)$ use edge $e = (u, v)$, but the interval may contain tokens that do not traverse u. In total we just need to store two ids of length $\mathcal{O}(\log(nW))$ per incident edge. (Recall that the maximum load over any edge is at most $FW \in \mathcal{O}(\text{poly}(nW))$.)

As we send exactly $|f'(e)|$ tokens over an edge e we get the same congestion as f'.

We want to construct a deterministic TS, so the input label at a node v contains a path id $l \in \{1, ..., \mu(v)\}$. As v stores the offset a from above, we can map the path id to $a + l$, one of the tokens starting at v. Afterwards, the routing algorithm simply needs to check which interval contains the token, to simulate their movement. This is deterministic. □

We use Lemma 1 typically to route between distributions which are 'close' to our weights c. As we can solve the PMCF with low congestion, this will have low congestion as well. The following lemma encapsulates that argument.

Lemma 2. *Let $\mu_{\text{in}}, \mu_{\text{out}}$ denote distributions with $\mu_{\text{in}}, \mu_{\text{out}} \leq c$ and $\mu_{\text{in}}(V) = \mu_{\text{out}}(V)$. Then there is a flow f with $\text{bal}_f = \mu_{\text{out}} - \mu_{\text{in}}$ and $\text{cong}(f) \leq 2C$.*

Proof. We construct f based on the PMCF, using Valiant's trick.

Let $(f_{u,v})_{u,v \in V}$ denote a solution to the PMCF with congestion at most C, where $f_{u,v}$ is a unit u-v-flow, i.e., it sends one unit of flow from u to v and has to be scaled by the corresponding demand.

Intuitively, we route from u to v by sending a packet to an intermediate node w, picked randomly weighted by c. The route itself uses the flows $f_{u,w}$ and $f_{w,v}$, which we can concatenate by adding them. Hence,

$$f := \sum_{u,v \in V} f_{u,v} \cdot \mu_{\text{in}}(u)\bar{c}(v) + \sum_{u,v \in V} f_{u,v} \cdot \bar{c}(u)\mu_{\text{out}}(v)$$

As both $\mu_{\text{in}}(u)\bar{c}(v)$ and $\bar{c}(u)\mu_{\text{out}}(v)$ are at most $c(u)c(v)/c(V)$, the demand of the PMCF, we have $\text{cong}(f) \leq 2C$. The flow to and from the intermediate nodes cancels, so a node v sends $\mu_{\text{in}}(v)$ packets and receives $\mu_{\text{out}}(v)$, yielding $\text{bal}_f = \mu_{\text{out}} - \mu_{\text{in}}$. □

Routing the PMCF Our first goal is to create a transformation scheme for the PMCF, for which we use the technique of cut-matching games introduced in [16]. We will not discuss it in detail, but instead encapsulate the properties of interest and refer to the

original proofs. We need modify the technique slightly [3] , so for those parts we briefly show how the proofs can be adapted.

Consider the following game. We are given a finite set of nodes V', with $|V'|$ even. There are $N \in \mathcal{O}(\log^2 |V'|)$ rounds and two players. In round k,

- Player 1 (the "cut player") chooses a partition $A_1 \dot\cup A_2 = V'$ with $|A_1| = |A_2|$,
- Player 2 (the "matching player") chooses a bijection $M : V' \to V'$ *respecting the partition*, i.e., it maps A_1 to A_2 and vice versa, and
- Player 2 chooses a partition B of V'.

At the end of the game, we define a random walk on V' consisting of N steps. In step k, a packet

- moves from node v to either v or $M(v)$ with probability $\frac{1}{2}$, and then
- moves from the resulting node v' to a node in $B_{v'}$ uniformly at random, where $v' \in B_{v'} \in B$ is the group of v' in the partition B.

The game is won by Player 1 if this random walk is *mixing*, i.e., for any $u, v \in V'$ it moves from u to v with probability between $1/|V'| \pm \varepsilon$, where $\varepsilon = 1/|V'|^2$.

Lemma 3 (KRV)**.** *Player 1 has a winning strategy.*

Proof. See [16, Section 3.1]. The proofs have to be changed slightly to work here, which we will now do, using the original notation.

While the original result uses perfect matchings instead of bijections for M, it extends directly by slightly modifying the proof of Lemmata 3.1 and 3.3. In particular, we can decompose M into two perfect matchings $m_1 = M \restriction A_1$ and $m_2 = M \restriction A_2$. We write

$$\psi_M(t) := \sum_{(i,j)\in M} \left\| \frac{P_i(t) + P_j(t)}{2} - \frac{\mathbf{1}}{n} \right\|^2$$

for the resulting potential after adding M to the random walk. Note that a perfect matching, such as m_1, changes the potential to $2\psi_{m_1}(t)$, as we need to count each node twice. We now have $\psi_M(t) = (2\psi_{m_1}(t) + 2\psi_{m_2}(t))/2$, and the original result guarantees the reduction for both halves.

Moving from v' to a node in $B_{v'}$ corresponds to averaging the probability vectors of the nodes in that cell, which does not increase the potential function: For any set of vectors $v_1, ..., v_k$ the Cauchy-Schwarz inequality yields

$$k \left\| \frac{1}{k} \sum_i v_i \right\|^2 \le \frac{1}{k} \Big(\sum_i 1 \cdot \|v_i\| \Big)^2 \le \frac{1}{k} \Big(\sum_i 1^2 \cdot \sum_i \|v_i\|^2 \Big) = \sum_i \|v_i\|^2$$

The original result just guarantees $\varepsilon \le 1/2|V'|$, but we can increase the number of iterations by a constant factor. □

[3] In particular, we use a bijection instead of a perfect matching, allow the matching player to choose subsets which will be shuffled randomly, and require a stronger bound on the error.

These random walks can be made deterministic, by storing whether to switch sides at each step, provided that we can send packets along our chosen bijection M. Additionally, while we need a lot of nodes in V' to match the weights c, a single node v can simulate many in V' with only little bookkeeping. Here we use that "mixing" nodes of V' at each step is not problematic, a notion made precise by choosing the appropriate partition B. The bijections M will be stored implicitly using single-commodity flows.

In total we get short descriptions of the possible paths taken by the random walk, enabling us to circumvent one of the key problems arising in weighted graphs — the inability to store paths directly within the graph due to too many paths going over a single edge.

Lemma 4 (PMCF transformation scheme)**.** *There exists a deterministic CTS that routes* $\mathbf{1}_v \xrightarrow{c(v)} c$ *for each* $v \in V$ *with approximation* $1 + \mathcal{O}(n^{-1})$ *and congestion* $\mathcal{O}(C \log^2 n)$*.*

The routing table of node v *has size* $\mathcal{O}(\deg(v) \log^3(nW))$*, while path ids and packet headers have length* $\mathcal{O}(\log^3(nW))$*.*

Proof. We want to play the game described above Lemma 3, so we define a set of "virtual" nodes $V' := \{1, ..., 2nc(V)\}$ [4] and choose an embedding $\varphi : V' \to V$ which assigns each node virtual nodes according to its weight, i.e. $|\varphi^{-1}(v)| = 2nc(v)$.

In each turn we have $A_1 \cup A_2 = V', |A_1| = |A_2|$ and can choose any bijection M respecting that partition. We want to simulate the random walk, so we also need a way to send packets according to M. In other words, we need a CTS that routes $\mathbf{1}_{\varphi(v')} \to \mathbf{1}_{M(\varphi(v'))}$ for each $v' \in V'$. It is difficult to construct such a CTS *given* a specific M, so instead we build the transformation scheme first, and then define M accordingly.

We construct two deterministic transformation schemes TS_1 and TS_2, routing from A_1 to A_2 and vice versa. Let us first consider TS_1.

A node v sends out a packet for each virtual node in A_1 assigned to it, so $\mu_1(v) := |\varphi^{-1}(v) \cap A_1|$ in total. Analogously, it receives $\mu_2(v) := |\varphi^{-1}(v) \cap A_2|$ packets. Hence we want to route $\mu_1 \xrightarrow{nc(V)} \mu_2$.

We have $\mu_1, \mu_2 \leq c$ and $\mu_1(V) = \mu_2(V) = nc(V)$, so Lemma 2 yields a flow f with $\mathrm{bal}_f = \mu_2 - \mu_1$ and $\mathrm{cong}(f) \leq 2nc(V)$. (We scale μ_1, μ_2 by $1/n$ and the resulting flow by n.)

Using f, we apply Lemma 1 to get a deterministic transformation scheme TS_1 that routes $\mu_1 \xrightarrow{nc(V)} \mu_2$ with congestion $\mathcal{O}(nC)$.

At each node v there are now $\mu_1(v)$ path ids to route a packet using TS_1. The transformation scheme is deterministic, so each path id corresponds to a single target node. Consider giving $\mu_1(v)$ packets to each node v, one for each path id, and routing them accordingly. Then, v will receive $\mu_2(v)$ packets. By mapping the outgoing packets to $\varphi^{-1}(v) \cap A_1$ and the incoming packets to $\varphi^{-1}(v) \cap A_2$ in some fashion, we get a bijection from A_1 to A_2.

We construct TS_2 in the same manner, and combine the two mappings to get the desired bijection M from V' to V'.

[4] We would like to have $V' = \{1, ..., c(V)\}$, but we need to make sure that $|V'|$ is even and at least n.

It remains to be shown that we can indeed route this bijection efficiently, i.e., without encoding any arbitrary mappings between virtual nodes and path ids.

Instead, we will simply choose a *random* path id for either TS_1 or TS_2, weighted such that each path id has the same probability. This corresponds to moving to a random virtual node assigned to v in each iteration, i.e. choosing our partition as $B := \{\varphi^{-1}(v) : v \in V\}$ in each round.

To summarize, we route the random walk as follows. In each iteration $k = 1, ..., N$ we flip a fair coin to decide whether we move from node v according to M or not. If yes, we pick a number i u.a.r. in $\{1, ..., 2nc(v)\}$. The first $\mu_1(v)$ numbers stand for path ids of TS_1, the others for path ids of TS_2. Then we route using the given transformation scheme and path id.

As we want our transformation scheme to be deterministic, these random choices will not be made while routing the packet, but encoded in the path id. There is a small technical issue in that the set $\{1, ..., 2nc(v)\}$ from which we sample i depends on v, but needs to be encoded in a path id chosen u.a.r. from some *fixed* range of integers. Instead, we will sample $i' \in \{1, ..., 2n^2c(V)N\}$ u.a.r and set $i := i' \pmod{2nc(v)}$. (Recall that N is the number of rounds.) So i is not quite uniform, but the probabilities differ by at most a factor of $1 + 1/nN$ in each round, and $1 + 1/n$ in total.

Thus we can store all random choices for the $\mathcal{O}(\log^2 nc(V)) = \mathcal{O}(\log^2(nW))$ iterations using $\mathcal{O}(\log^3(nW))$ bits. These are the path ids of our transformation scheme. As there is no randomness apart from the choices encoded in the path id, the transformation scheme is deterministic. The packet headers need to include the headers from Lemma 1, as well as our path ids; the length of the latter dominates.

Recall that we want to have a CDS that routes $\mathbf{1}_v \xrightarrow{c(v)} c$ for each $v \in V$. Hence, in aggregate the input distribution is c.

Let us now analyze the congestion. TS_1 routes $\mu_1 \xrightarrow{nc(V)} \mu_2$ and TS_2 does $\mu_2 \xrightarrow{nc(V)} \mu_1$, both with congestion $\mathcal{O}(nC)$. If we add them and scale the demand by $1/n$ we route $c \xrightarrow{c(V)} c$ with congestion $\mathcal{O}(C)$. So the distribution of packets does not change in an iteration, except for the factor of $1 + 1/nN$ above, and the total congestion is $\mathcal{O}(C \log^2(nW))$.

Due to Lemma 3, the random walk moves to any virtual node with probability between $1/2nc(V) \pm 1/|V'|^2$. We have $|V'|^2 \geq 2n^2c(V)$, so scaling by the total amount of flow $c(V)$ yields a value in $1/2n \pm 1/2n^2$. A node $v \in V$ has $2nc(v)$ virtual nodes, so it receives between $c(v)(1 \pm 1/n)$ packets in the random walk, or between $c(v)(1 \pm 2/n)$ in the actual transformation scheme. Hence the output distribution is c, with an approximation of $1 + \mathcal{O}(n^{-1})$. □

We remark that this CTS has input distribution $\mathbf{1}_v$ for commodity $v \in V$, which means that the source node of a packet already encodes its commodity.

Mixing CTS To close out section we prove that we can implement the mixing step with the tools we have. To start, we need a small helper lemma.

Lemma 5 (Routing distributions similar to c)**.** *Let $\mu_{\text{in}}, \mu_{\text{out}}$ denote integral distributions with $\mu_{\text{in}}, \mu_{\text{out}} \leq c$ and set $M := \min\{\mu_{\text{in}}(V), \mu_{\text{out}}(V)\}$. Then there exists a deterministic TS that routes $\mu_{\text{in}}(V) \xrightarrow{M} \mu_{\text{out}}(V)$ with congestion $\mathcal{O}(C)$. The routing table of node v has size $\mathcal{O}(\deg(v) \log(nW))$, while path ids and packet headers have length $\mathcal{O}(\log(nW))$.*

Proof. From Lemma 2 we get a flow f with congestion $2C$, by scaling the distributions to $M\bar{\mu}_{\text{in}}$ and $M\bar{\mu}_{\text{out}}$. Scaling both f and the distributions by $\alpha := \mu_{\text{in}}(V)\mu_{\text{out}}(V)$, the latter are now integral again and we can apply Lemma 1. This routes $\alpha M\bar{\mu}_{\text{in}} \xrightarrow{\alpha M} \alpha M\bar{\mu}_{\text{in}}$ with congestion $\mathcal{O}(\alpha C)$, or equivalently $\mu_{\text{in}} \xrightarrow{M} \mu_{\text{in}}$ with congestion $\mathcal{O}(C)$. □

Now we fix a cluster S with children $S_1, ..., S_r$. As we will later change the numbering of children, it is important that the following lemma works for an arbitrary one.

Lemma 6 (Mixing CTS)**.** *There exists a CTS that routes $w_{S_i} \xrightarrow{w_S(S_i)} w_S$ for each $i = 1, ..., r$ with congestion $\mathcal{O}(C \log^2 n)$ and approximation $1 + \mathcal{O}(n^{-1})$. The routing table of node v has size $\mathcal{O}(\deg(v) \log^3(nW))$, while path ids and packet headers have length $\mathcal{O}(\log^3(nW))$.*

Proof. For each S_i we route $w_{S_i} \xrightarrow{w_S(S_i)} \text{out}_{S_i}$ *within* S_i using Lemma 5 for weights $c := w_{S_i}$. This has congestion $\mathcal{O}(C)$, as $w_S(S_i) = \text{out}(S_i)$. It uses space only within S_i, so $\mathcal{O}(\deg(v) \log(nW))$ per node $v \in S$.

In total, the packets are now in distribution $\sum_i \text{out}_{S_i} = w_S$ and we apply Lemma 4 with $c := w_S$. (Here we do not use that Lemma 4 gives a deterministic CTS.) As all source nodes route to w_S concurrently, we route $\text{out}_{S_i} \xrightarrow{w_S(S_i)} w_S$ for each $i = 1, ..., r$ (with approximation $1 + \mathcal{O}(n^{-1})$). This has congestion $\mathcal{O}(C \log^2 n)$.

For the bounds on space per node and length of packet headers, the costs of the latter step dominate. □

As for Lemma 4, the source node of a packet already determines the commodity, so there is no need to specify an encoding for it.

3.2 Constructing an Unmixing CTS

General Graph Embedding Up until now, we have not used that we have only edges of distinct classes. The next two lemmas concern randomized rounding, which we use to select a small number of paths from a flow without increasing congestion. This uses a probabilistic argument to prove existence, but the choice of paths is fixed and not subject to randomness.

Consider some flow f sending k packets from a source node u to a node v with congestion 1, where the flow involves only edges of weight 1. It is obvious that taking a single path with weight k from that flow uniformly at random increases the congestion to k, while taking k paths with weight one should intuitively work quite well, giving a congestion $1 + \mathcal{O}(\log k)$. This intuition is correct, which we now prove formally.

We call a multi-set of u-v-paths a *path system*. The *class of a path* is the minimum class of its edges, and the *class of a path system* P is the minimum class amongst its paths. To send a packet using a path system P we choose a path uniformly at random. Therefore,

if we have multiple path systems $P = \{P_1, ..., P_k\}$ with demands $d = \{d_1, ..., d_k\}$, then their total congestion is $\text{cong}(P, d) := \text{cong}(\{d_i p/|P_i| : i \in \{1, ..., k\}, p \in P_i\})$.

Lemma 7 (Randomized rounding)**.** *Let $P = \{P_1, ..., P_k\}$ denote a set of path systems with demands d. Then there exists a set $P' = \{P'_1, ..., P'_k\}$, with $P'_i \subseteq P_i$ and $|P'_i| \leq \lceil 2^{-l} d_i \rceil$ for each P_i of class l. The congestion is $\text{cong}(P', d) \in \mathcal{O}(\text{cong}(P, d) + \log n)$.*

Proof. We use the probabilistic method, so we will choose P'_i by picking the appropriate number of paths from P_i randomly, and then show that the congestion is low enough with positive probability. The latter part uses the following bound:

Lemma 8 (adapted from [22, lemma 10])**.** *Let $X_1, ..., X_n$ denote a set of negatively correlated random variables taking values in $[0, 1]$. Let X denote their sum, and let $\delta \geq \mathbb{E}(X)$. Then $\Pr(X \geq \mathbb{E}(X) + \delta) \leq e^{-\delta/3}$.*

First we consider the congestion on a particular edge e. For path system P_i of class l we sample $N_i := \lceil 2^{-l} d_i \rceil$ paths (with replacement), so we define random variables $X_{i,p,j}$ for each $p \in P_i, j \in \{1, ..., N_i\}$ as the congestion induced on e. More precisely, if $e \in p$ and p is picked as the j-th path, then $X_{i,p,j} = d_i / N_i w(e)$, else $X_{i,p,j} = 0$. Note that an $e \in p \in P_i$ has class at least l, so $X_{i,p,j} \leq 1$.

Finally, $X_e := \sum X_{i,p,j}$ is the total congestion of edge e. Of course, each path still has the same probability, so $\mathbb{E}(X_e)$ equals the original congestion on e w.r.t. P and demands d. Choosing $\delta := 6 \ln m + \mathbb{E}(X_e)$ for Lemma 8, we get $\Pr(X_e \geq 2\mathbb{E}(X_e) + 6 \ln m) \leq 1/m^2$.

There are m edges in total, so by union bound the probability of *any* edge having congestion larger than $\mathcal{O}(\text{cong}(P, d) + \log n)$ is strictly less than 1. □

Having the tool of randomized rounding at our disposal, we now turn to the most involved lemma in our construction. If we want to route small demands we can already do so using Lemmata 4 (to get a path system) and 7 (to pick a small number of paths to store). However, routing a demand of size, say, W from node u to v, we would have to pick $2^{-l}W$ paths from a path system $P_{u,v}$ connecting u and v, to ensure low congestion. Here, l is the class of $P_{u,v}$.

Hence, if l is small we would need to route a large number of paths. Instead, we find a cut consisting only of small (i.e., class l) edges separating each path in $P_{u,v}$. [5] These can be used to store routing information.

So we take all pairs (u_i, v_i) in the same situation, that is, connected by a class l path system, and take the union of all the cuts consisting of class l edges. Then we route a single-commodity flow from the nodes u_i to this cut. Of course, the packets from u may have ended up at an edge belonging to some other u_i, so it may not be possible to route to v directly. Instead we send the packets back through the single-commodity flow.

Again, the packets from u may now reside in a different node u'. However, on the way they passed through a class l edge, which we can use for storing the path from u' to v. (To be precise, we use one of the adjacent nodes for storage.) But now we have a new problem—while both $P_{u,v}$ and $P_{u',v'}$ are class l path systems, $P_{u',v}$ need not be. If it has

[5] Note that this is not a cut of the *graph*, which might still be connected, but of the *path system*.

class at least l, all is well and we can route the packet with a single path. Though if it has a class $l' < l$ we have to route using multiple paths again.

The fact that the class keeps decreasing allows us to solve this problem recursively. At each class we split the packet into smaller ones and find an edge to store the routing information for each of them. This stops when the packet is small enough to route directly, at the latest once it has reached size 1.

When we refer to storing the routing information for a node u in some previous node v on the path of a packet, we are using shorthand for a slightly elaborate transformation scheme, which we will refer to as *anticipative routing*. When the packet arrives at node v, the node checks the packet header and adds the stored routing information to it, before sending the packet on its way normally. Then, once the packet has reached the node u the routing information is extracted from the packet header and used.

Lemma 9 (General graph embedding)**.** *Let $G' = (V, A, d)$ denote a weighted, directed graph, where the total weight of incoming and outgoing arcs of a node v is at most $c(v)$, and $\deg_{G'}(v) \in \mathcal{O}(\deg(v) \log^2 n)$. Then there is a CTS that routes $\mathbf{1}_v \xrightarrow{d(u,v)} \mathbf{1}_v$ for each $(u, v) \in A$ with congestion $\mathcal{O}(C \log^2 n \log^2 W)$.*

The routing table of node v has size $\mathcal{O}(\deg(v) C \log^2 n \log W \log^3(nW))$, while packet headers have length $\mathcal{O}(\log^3(nW))$. Commodity $(u, v) \in A$ is encoded as $l \in \{1, ..., \deg_{G'}(u)\}$.

Proof. As mentioned above, the problematic demands are those which need multiple paths to route with low congestion. We will refer to those demands as *large*. More precisely, we call an arc $(u, v) \in A$ *l-large* if $d(u, v) > 2^l$ and l is the class of $P_{u,v}$ (defined below).

The proof will proceed in three parts.

(a) First, we construct path systems $P = \{P_{u,v} : u, v \in V\}$, s.t. all paths in $P_{u,v}$ have the same class and can be routed by storing a $\mathcal{O}(\log^3(nW))$ path id. For any demands d' where both the total incoming and outgoing demand of a node v are at most $c(v)$, we have $\text{cong}(P, d') \in \mathcal{O}(C \log^2 n \log W)$.

(b) Then we show that we can partially route arcs $a \in A$ which are l-large, replacing them with $2^{-l} d(u, v)$ arcs of weight 2^l. This does not change either the total outgoing or incoming demand of any node.

(c) Finally, we construct the CTS and derive the resulting bounds.

Part (a). We use Lemma 4 to construct a *deterministic* concurrent transformation scheme TS_1 routing the PMCF. Hence we can have $P'_{u,v}$ denote the path system containing the paths from u to v, one for each path id. Then we employ Valiant's trick and define $P^*_{u,v}$ as $\bigcup_{w \in V} P'_{u,w} \circ P'_{w,v}$, where $P \circ P'$ is a concatenation of path systems P, P' given by $P \circ P' := \{p \circ p' : p \in P, p' \in P'\}$. That means that we can split a path in $P^*_{u,v}$ into its first and second part.

Now consider some demands d', where the incoming or outgoing demand of any node v is at most $c(v)$. As $\bigcup_{w \in V} P'_{u,w}$ are all outgoing paths of u, sampling one u.a.r. is equivalent to sending a packet with TS_1 from u. So the first parts create the same

congestion as TS_1, given that $\sum_v d(u,v) \leq c(v)$. To be precise, the congestion increases by a factor of $1 + \mathcal{O}(n^{-1})$, the approximation guaranteed by Lemma 4. This is only a constant factor, so we are going to disregard it.

The intermediate node w follows distribution c. The second parts, i.e., the paths from $P'_{w,v}$ then have weight $\sum_u c(w)d(u,v) \leq c(w)c(v)$. Routing a packet from w using TS_1 chooses a path from $P'_{w,v}$ with weight $\bar{c}(v)$, so we also bound the congestion based on TS_1.

In total we get $\text{cong}(P^*, d') \leq 2\,\text{cong}(TS_1, c) \in \mathcal{O}(C \log^2 n)$. Finally, we want to modify $P^*_{u,v}$ so that it only contains paths of one class. We simply pick a class l with the maximum number of paths in $P^*_{u,v}$, and set $P_{u,v} := \{p \in P^*_{u,v} : p \text{ has class } l\}$. As there are N_{class} classes, we have $|P^*_{u,v}| \leq |P_{u,v}| N_{\text{class}}$ and the congestion increases by $\mathcal{O}(\log W)$.

Each path in $P_{u,v}$ is the concatenation of two paths from TS_1, so we can store two path ids of TS_1 to route it.

Part (b). We choose the highest class l where the set A' of l-large arcs is non-empty. Additionally, we introduce $\text{str} : A \to V$, which is the node that will be used to store the routing information for an arc. Initially, $\text{str}(u,v) = u$.

For any arc $a \in A'$ we define $d'(a)$ as the largest multiple of 2^l s.t. $d'(a) \leq d(a)$, and then set $d := d - d'$. So when routing a, a coin is flipped. With weight $d(a)$ we route using a (how precisely is yet to be determined), with weight $d'(a)$ we do the following procedure.

For all $(u,v) \in A'$ the path system $P_{u,v}$ has class l. Using Lemma 7 with demands d', we find a set of $d'(u,v)2^{-l}$ class l paths from u to v for $(u,v) \in A$, with congestion $\mathcal{O}(C \log^2 n \log W)$. We let M denote the set of prefixes of these paths, up to (and including) their first class l edge. By treating M them as a flow f, we can construct a transformation scheme TS, using Lemma 1, which has the same congestion.

Going back to the arc a we want to route, we send a packet using TS, with path id chosen uniformly at random. The necessary information for this is stored in $\text{str}(a)$. There are $N := d'(a)2^{-l}$ paths in M for a (and thus path ids of TS). If we put that number of tokens into the source of a and apply TS (one path id per token), they end up at nodes $z_1, ..., z_N$. A node z_i may receive tokens from other demands $a' \in A'$, but at most $\mathcal{O}(\deg(z_i) C \log^2 n \log W)$ in total, as each path ending in z_i in M induces a load of 2^l on a class l edge adjacent to z_i, i.e., a congestion of 1.

We also construct a transformation scheme based on the reverse flow $-f$, to send the tokens back. This does *not* use a random path id, instead a node z_i stores a mapping from incoming to outgoing path ids (any mapping is fine). We remark that the tokens of a may not end up where they started, as routing through a single-commodity flow mixes packets arbitrarily.

To summarize, an arc $a =: (u,v)$ has sent out N tokens, each of which corresponds to 2^l flow from $d'(a)$. Each token traversed an intermediate node z_i to end up at a node u'. Node z_i was passed by a low number of tokens in total. So now we add a new arc $a' := (u', v)$ to A, with demand $d(a') := 2^l$. Crucially, the routing information for a' is stored in z_i, i.e., $\text{str}(a') := z_i$.

As a technical detail we note that we allow for parallel arcs in G'. It is important that

we do not merge multiple small demands into a larger one, as we have already ensured sufficient storage space for each, which would be lost.

The tokens are routed through f and then $-f$, so the number of tokens starting and ending at u is the same. This implies that both the total outgoing and incoming demand of any node remain unchanged.

As mentioned above, this procedure uses anticipative routing. For demand a we send N packets from u, each of which follows a deterministic path. So the intermediate node z_i assigns the packet the specific path id sending it to u' as well as the (yet to be determined) information on how to proceed from there. At u' the node does not have to look up the packet header in its routing table, but merely execute the information contained within.

Part (c). First, we apply (b) at most N_{class} times to eliminate all large arcs. Note that while (b) introduces new arcs, these have demand 2^l, where l is maximum class s.t. l-large arcs exist. So the new demands can only be l'-large for an $l' < l$.

Now we route the remaining arcs. Those are not large, so we can use Lemma 7 to pick a single path from $P_{u,v}$ for each $(u, v) \in A$. Based on our construction in (a), each path in $P_{u,v}$ can be routed using a $\mathcal{O}(\log^3(nW))$ path id. This will be stored in $\text{str}(u, v)$.

For the initial arcs, we store their path ids within their respective source nodes together with their (encoded) commodity.

Finally, we analyze the congestion and space requirements.

Each use of (b) creates congestion of $\mathcal{O}(C \log^2 n \log W)$, due to embedding two flows. Routing the non-large arcs at the end creates the same congestion (though only once). So in total we have a congestion of $\mathcal{O}(C \log^2 n \log^2 W)$.

In total, each node v is used at most $\deg_{G'}(v) \in \mathcal{O}(\deg(v) \log^2 n)$ times for storage due to our initial demands, and then at most $\cdot\mathcal{O}(C \log^2 n \log W)$ times for each adjacent class l edge when executing (b) for class l. Storing routing information for a large arc needs $\mathcal{O}(\log(nW))$ additional space to store the number of tokens and the range of path ids for them. This is dominated by the $\mathcal{O}(\log^3(nW))$ sized path id we need for both large and non-large arcs. (Recall that a large demand is first split into a fractional part and a multiple of 2^l.)

To embed the flows in (b) using Lemma 1, we need a total of $\mathcal{O}(\deg(v) \log(nW) \log W)$ space per node v, and transformation scheme in (a) from Lemma 4 uses $\mathcal{O}(\deg(v) \log^3(nW)))$ space. Summing everything up, we get $\mathcal{O}(\deg(v) C \log^2 n \log^2 W \log^3(nW))$.

Regarding packet headers, we need packet headers of Lemmata 1 and 4, as well as some additional space for our anticipative routing (at most $\mathcal{O}(\log^3(nW))$). In total we get $\mathcal{O}(\log^3(nW))$. □

We want to remark on a slight technicality in the previous proof. Usually, scaling the routed distributions by some constant factor will scale the congestion by the same and nothing of importance has changed. However, the proof argues that there is a bound on the space used for each node, based on the congestion. Scaling the routed distribution to decrease congestion does actually affect this bound, so we could try scaling the congestion even lower. Though, as it turns out it is not possible to get a congestion below $\mathcal{O}(C \log^2 n \log W)$ as that is the minimum when fixing a single path provided by

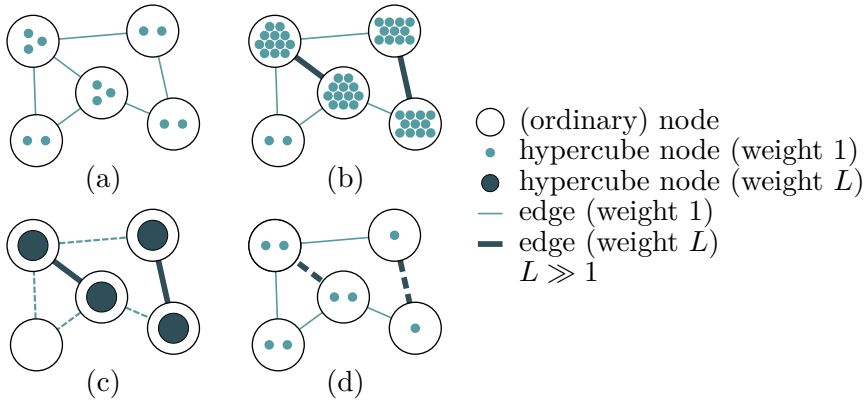

Figure 4: Embedding a hypercube in a cluster S. If all edges have weight 1, assigning hypercube nodes according to w_S ensures that no node receives more than its degree (a). However, if edges with large weights exist, this no longer works (b). Instead, we assign hypercube nodes according to edges of one class, here either edges with weight L (c) or weight 1 (d).

part (a). Using a fractional path would indeed have lower congestion, but not take up less space.

Hypercube embedding Now we move on to the hypercube embedding. Consider some cluster S with children $S_1, ..., S_r$. The general idea is that we assign each node v some w_S hypercube ids, by giving each child cluster S_i an interval of $w_S(S_i) = \text{out}_{S_i}(S_i)$ hypercube ids, distributed according to out_{S_i}. (Recall that $w_S = \sum_i \text{out}_{S_i}$.) Of course, this does not quite make a hypercube, so we have to skew the distributions by at most some constant factor so that everything ends up in a power of two.

Then there is a second problem, illustrated in Figure 4. The main reason for embedding a hypercube is to reduce the number of paths a node has to store to reach any target. Within the hypercube, each node has logarithmic degree but we can still route to any node due to the special structure of the hypercube. We then leverage that to route to the interval assigned to child cluster S_i, effectively routing to out_{S_i}.

However, we need to ensure that a node v is assigned $\tilde{\mathcal{O}}(\deg(v))$ ids. If there are much more than that, we cannot store routing information in v for its adjacent hypercube edges. The original result in [22] had $w_S(v) \leq \deg(v)$ due to unit weights, but we do not. Instead we will assign a node v roughly $w_S^{(l)}(v)/2^l$ hypercube ids, meaning one for each adjacent class l edge contributing to w_S.

There are, of course, at most $\deg(v)$ such edges, so we do not run into storage problems. But if the distributions $w_S^{(l)}$ and w_S are too dissimilar then we cannot route between them with low congestion. (The PMCF only ensures that distributions close to w_S can be routed well.) Hence we need to choose the class l carefully, so that $w_S^{(l)}(S_i)$ contains

enough edges and we do not have to put too much flow on any single node.

This creates another complication, as different child clusters may necessitate different choices of l. While child clusters may have different classes, there are only N_{class} many classes. So we will implement a hypercube for each of them, and later have a flow for each class which sends the data into the initial distribution for the specific hypercube.

Lemma 10 (Hypercube embedding)**.** *Let S be an arbitrary cluster with children $S_1, ..., S_r$. There exists a compact CTS that routes $\text{maj}_S^{(l)} \xrightarrow{w_S(S_i)} \text{out}_{S_i}^{(l)}$ for each S_i of class l with approximation 2 and congestion $\mathcal{O}(C \log^3 n \log^3 W)$.*

The routing table of node v has size $\mathcal{O}(\deg(v) C \log^2 n \log W \log^3(nW))$, while packet headers have length $\mathcal{O}(\log^3(nW))$. There exists a numbering of $S_1, ..., S_r$ s.t. commodity S_i is encoded as integer i.

Proof. In the same manner as Räcke and Schmid [22], we embed a hypercube. However, we use a hypercube for each class l of edges and each hypercube id we assign has weight 2^l, i.e., a node $v \in S$ gets roughly $\text{maj}_S^{(l)}(v)/2^l$ hypercube ids.

The construction proceeds in a similar manner as the one of Räcke and Schmidt up until the embedding of the hypercube edges, where we use Lemma 9 instead of simple randomized rounding. We start by arguing that we can renumber the child clusters s.t. given the index we can determine both the class and the approximate weight of a child.

Storing child class and weight. For a child cluster S_i with class l, let $\|S_i\|$ denote the smallest power of two with $\text{out}_S^{(l)}(S_i)/2^l \leq \|S_i\| \leq 2\,\text{out}_S^{(l)}(S_i)/2^l$. This is the number of hypercube nodes that we assign to S_i.

We store the number of children of each class, which takes $\mathcal{O}(\log r \log W)$ bits, in each node in S. Additionally, for each class l we store the number of child cluster of that class which have a specific value of $\|S_i\|$. There are at most $1 + \log_2 m$ different values for $\|S_i\|$, so we need $\mathcal{O}(\log r \log n)$ bits. In total, this uses $\mathcal{O}(\log r \log n \log W)$ bits in each node in S.

For our renumbering, we sort child clusters S_i by class and, within a class, by their value of $\|S_i\|$. Given an index i based on this sorting, we can determine both class of S_i and $\|S_i\|$.

Constructing the class l hypercube. Fix some class l. We will now describe the construction of the class l hypercube, then analyze at the end the congestion for all classes at once.

The hypercube has dimension d, with $d \in \mathbb{N}$ minimal s.t. $2^d \geq \sum_{L(S_i)=l} \|S_i\|$, where $L(S_i)$ is the class of S_i. Each class l child S_i gets a range of $\|S_i\|$ ids, distributed such that a node $v \in S_i$ gets between $\text{out}_S^{(l)}(v)/2^l$ and $2\,\text{out}_S^{(l)}(v)/2^l$ hypercube ids. These ids are stored in v. As the order of children is fixed and stored within each node, we can recompute the range of any child cluster during routing.

We have assigned $\sum_{i=1}^{r} \|S_i\|$ hypercube ids in total, which may be less than 2^d. Hence we distribute the other hypercube ids evenly across the nodes of class l child clusters S_i, s.t. a node $v \in S_i$ receives at most $2\,\text{out}_S^{(l)}(v)/2^l$ additional hypercube ids, and thus between $\text{out}_S^{(l)}(v)/2^l$ and $4\,\text{out}_S^{(l)}(v)/2^l$ in total. These other hypercube ids will only be used during routing as intermediate nodes.

Congestion within the hypercube. Now consider some packet at a node $u \in S$ that we want to route to S_i. First we pick a hypercube node x u.a.r. among those assigned to u (they are stored in u). Then we pick a hypercube node y u.a.r. from the range assigned to S_i (which we can recompute). Then we route from x to z, a random intermediate node in the hypercube, then from z to y.

We remark that, in a hypercube, the PMCF with weights $c := 1$ can be solved with congestion $\mathcal{O}(1)$ and that this bound is achieved by routing in the usual manner, i.e., fixing an order for the bits and sending the packet along the edge according to the first bit different between source and target. As we are using Valiant's trick, the congestion is determined by the maximum incoming or outgoing amount of flow for a single node.

For the congestion, we consider routing $\mathrm{maj}_S^{(l)} \xrightarrow{\mathrm{out}_S^{(l)}(S_i)} \mathrm{out}_{S_i}^{(l)}$ for S_i with class l, i.e. $\mathrm{out}_{S_i}^{(l)}(S_i)$ units of flow instead of $w_S(S_i) = \mathrm{out}_{S_i}(S_i)$. As S_i has class l, we have $\mathrm{out}_{S_i}^{(l)}(S_i) \geq \mathrm{out}_{S_i}(S_i)/N_{\text{class}}$ and the congestion increases by a factor of at most N_{class}.

Summing up all $\mathrm{out}_S^{(l)}(S_i)$ we get $\mathrm{maj}_S^{(l)}(S)$, so a node $v \in S_i$ sends out $\mathrm{out}_S^{(l)}(v)$ packets, and each hypercube node x of v sends at most 2^l of them. For commodity i there are $\|S_i\| \geq \mathrm{out}_S^{(l)}(S_i)/2^l$ hypercube nodes, so each receives at most 2^l packets.

Both outgoing and incoming flow of a hypercube node are at most 2^l, so the load on a hypercube edge is also at most $\mathcal{O}(2^l)$.

While we send to a hypercube node from the range of S_i u.a.r., a node $v \in S_i$ is assigned between $\mathrm{out}_S^{(l)}(v)/2^l$ and $2\,\mathrm{out}_S^{(l)}(v)/2^l$ of them. Hence the target distribution is only within an approximation of 2.

Embedding into the original graph. Finally, we embed the hypercube using Lemma 9. A node $v \in S_i$ for S_i of class l has at most $4\,\mathrm{maj}_S^{(l)}(v)/2^l \leq 4\deg(v)$ hypercube ids. So there are at most $8m$ nodes in the hypercube in total, and the degree of each node is $\mathcal{O}(\log n)$. Let $d_l(u, v)$ denote the number of edges connecting u and v in the class l hypercube, for $u, v \in S$, and $d := \sum_l 2^l d_l$. Setting $A := \{(u, v) : d(u, v) > 0\}$ we embed the graph $G' := (S, A, d)$.

As the load on an edge of the class l hypercube is at most 2^l, in total $d(u, v)$ packets are sent from u to v. A node v in a class l child has outgoing and incoming demand at most $\mathcal{O}(k2^l \log n) \subseteq \mathcal{O}(w_S(v) \log n)$, where k is the number of class l edges incident to v. The congestion of Lemma 9 increases by $\mathcal{O}(\log W)$ due to decreasing the total number of packets earlier in our analysis, and $\mathcal{O}(\log n)$ due to the outgoing and incoming demand of a node.

While we use an additional $\mathcal{O}(\log r \log n \log W)$ space per node v to store the sizes of clusters, and $\mathcal{O}(\deg(v) \log n)$ to store the hypercube ids of nodes assigned to v, this is dominated by the cost of Lemma 9, which also determines the sizes of packet headers. □

Unmixing CTS Given the hypercube embedding from the last lemma, we can now construct the unmixing CTS. At the beginning we need to ensure that we move to the distribution for the correct class, then we move through the (class specific) hypercube, and finally we go to the target distribution.

Lemma 11 (Unmixing CTS)**.** *There exists a CTS that routes* $w_S \xrightarrow{w_S(S_i)} w_{S_i}$ *for each* $i = 1, ..., r$ *with congestion* $\mathcal{O}(C \log^3 n \log^3 W)$*. The routing table of node* v *has size* $\mathcal{O}(\deg(v) C \log^2 n \log W \log^3(nW))$*, while packet headers have length* $\mathcal{O}(\log^3(nW))$*. There exists a numbering of* $S_1, ..., S_r$ *s.t. commodity* S_i *is encoded as integer* i*.*

Proof. The numbering of child clusters and our path ids are the same as for Lemma 10. Therefore we can determine the class l of S_i based on its index, as shown in the proof of that lemma.

For a child S_i with class l we want to route $w_S \to \text{maj}_S^{(l)} \to \text{out}_{S_i}^{(l)} \to \text{out}_{S_i}$.

(1) For each class l let $M \subseteq S$ denote the union of class l child clusters. We route $w_S \xrightarrow{w_S(M)} \text{maj}_S^{(l)}$ using Lemma 5 with congestion $C \cdot w_S(M) / \text{maj}_S^{(l)}(M)$. This is at most CN_{class}, as $\text{maj}_S^{(l)}(S_i) = \text{out}_{S_i}^{(l)}(S_i) \geq w_S(S_i)/N_{\text{class}}$ for each child S_i with class l.

(2) We use Lemma 10 once, to route $\text{maj}_S^{(l)} \xrightarrow{w_S(S_i)} \text{out}_{S_i}^{(l)}$, with congestion $\mathcal{O}(C \log^3 n \log^3 W)$.

(3) For each S_i we route $\text{out}_{S_i}^{(l)} \xrightarrow{w_S(S_i)} \text{out}_{S_i}$ *within* S_i using Lemma 5. Here we have congestion $C \cdot w_S(S_i) / \text{out}_{S_i}^{(l)}(S_i) \leq CN_{\text{class}}$.

Note that (1) has to be implemented on the whole cluster for each class, so its total congestion is $\mathcal{O}(C \log^2 W)$ (but still lower than step (2)). For the bounds on space per node and length of packet headers, the costs of step (2) dominate. □

3.3 Combining the Results

Lemma 10 can be used directly as a drop-in replacement in the original result in [22]. However, we have organized things slightly differently and thus feel it necessary to repeat the analysis.

The key idea is routing between two nodes u and v using the decomposition tree, spreading out a packet according to distribution w_S in each cluster. This ensures that routing within a cluster can be done with low congestion. Moving through the tree, the congestion is determined by the bottlenecks out_S. However, the optimal algorithm has to send the packets through these bottlenecks as well, so we remain competitive.

Theorem 12. *There exists a compact oblivious routing scheme with competitive ratio* $\mathcal{O}(\log^6 n \log^3 W)$*, using a routing table of length* $\mathcal{O}(\deg(v) \log^5 n \log W \log^3(nW))$ *for a node* $v \in V$*, packet headers of length* $\mathcal{O}(\log^3(nW))$*, and node labels of length at most* $\mathcal{O}(\text{height}(T) \log \deg(T))$*.*

Proof. The analysis is mostly analogous to [22, Lemma 2], apart from the slight change that Lemmata 6 and 11 route directly between w_S and w_{S_i} instead of splitting into an upper and lower sub-path.

To route from node u to v, we determine the clusters S_u, S_v containing just these nodes, i.e., $S_u = \{u\}$ and $S_v = \{v\}$. Let p denote the path in the decomposition tree from

$\{u\}$ to $\{v\}$, which has length $k \in \mathcal{O}(\log n)$. We start in distribution $\bar{w}_{S_u}(V) = \mathbf{1}_u$, and want to end at $\bar{w}_{S_v}(V) = \mathbf{1}_v$. This is done by going through the sequence of distributions $w_{p_1}, w_{p_2}, ..., w_{p_k}$, routing from w_{p_i} to $w_{p_{i+1}}$ using Lemma 6 if p_{i+1} is the parent of p_i, and Lemma 11 otherwise.

We accumulate a slight multiplicative error of $1 + \mathcal{O}(n^{-1})$ at each step, which is bounded by a constant factor in total, as we have at most 2 height$(T) \in \mathcal{O}(\log n)$ steps. The final distribution is $\mathbf{1}_v$ and remains unchanged by any error, so this merely increases congestion by a constant.

It is necessary to determine the path through the decomposition tree, hence the label of a node v consists of the path in the decomposition tree, encoded as a sequence of child cluster indices (given by Lemma 10. These are enough to determine the full path, by looking at the node labels of the start and end node.

Now we analyze the competitive ratio. Let $d : V \times V \to \mathbb{R}$ denote demands.

Fix any edge $e \in E$. Load on e is generated only when routing between distributions w_{S_i} and w_S for some cluster S with child cluster S_i, where S contains both endpoints of e. This uses that the routing between the two distributions happens inside of S, and does not generate load on any edge not fully contained. Sending a packet from u to v involves routing between distributions w_{S_i} and w_S only if one of u, v is not in S_i and the other one is, so the total demand for these is $\lambda(i) := \sum_{u \in S_i} \sum_{v \notin S_i} (d(u, v) + d(v, u))$.

However, the demand $\lambda(i)$ must enter or leave S_i (and thus pass over an edge in out_{S_i}) regardless of our specific routing scheme. So there are $\lambda(i) \leq C_{\mathrm{opt}} \mathrm{out}_{S_i}(S_i)$ such packets at most, where C_{opt} is the optimal congestion for demands d. Using $w_S(S_i) = \mathrm{out}_{S_i}(S_i)$ we get $w_S(S_i)/\lambda(i) \leq C_{\mathrm{opt}}$.

Applying Lemmata 6 and 11 with $C \in \mathcal{O}(\log^2 n)$ then results in a congestion of at most $\mathcal{O}(C_{\mathrm{opt}} \log^5 n \log^3 W)$, and for each node $v \in S$ it uses $\mathcal{O}(\deg(v) \log^4 n \log W \log^3(nW))$ space, as well as packet headers of length $\mathcal{O}(\log^3(nW))$.

Both edges and nodes can be contained in at most $T_{\mathrm{h}} \in \mathcal{O}(\log n)$ clusters, giving the final bounds on congestion and space per node. For a packet we need to store the path through the decomposition tree, so $\mathcal{O}(\log n)$ path ids of length $\mathcal{O}(\log \deg(T))$ and the length of a packet header does not increase.

As mentioned above, we store the cluster indices in the label of a node v, for each cluster in which v is contained, resulting in node labels of length $\mathcal{O}(T_{\mathrm{h}} \log \deg(T))$. □

Corollary 13. *Assume $W \in \mathcal{O}(\mathrm{poly}(n))$. Then there exists a compact oblivious routing scheme with competitive ratio $\mathcal{O}(\log^9 n)$, using a routing table of length $\mathcal{O}(\deg(v) \log^9 n)$ for a node $v \in V$, packet headers of length $\mathcal{O}(\log^3 n)$ and node labels of length $\mathcal{O}(\log^2 n)$.*

References

[1] Ittai Abraham, Cyril Gavoille, and Dahlia Malkhi. On space-stretch trade-offs: Upper bounds. In *Proc. 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 217–224, 2006. URL: `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/agm-spaaub.pdf`, `doi:10.1145/1148109.1148144`.

[2] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Räcke. Optimal oblivious routing in polynomial time. *Journal of Computer and System Sciences*, 69(3):383–394, 2004. URL: https://ttic.uchicago.edu/~harry/pdf/optimal_oblivious_journal.pdf, doi:10.1145/780542.780599.

[3] Yair Bartal and Stefano Leonardi. On-line routing in all-optical networks. In *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 516–526. Springer, 1997.

[4] Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the 15th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 24–33, 2003. URL: https://ttic.uchicago.edu/~harry/ps/constructing_hierarchy.ps, doi:10.1145/777412.777418.

[5] Allan Borodin and John E. Hopcroft. Routing, merging, and sorting on parallel models of computation. *Journal of computer and system sciences*, 30(1):130–145, 1985. URL: http://www.cs.toronto.edu/~bor/Papers/routing-merging-sorting.pdf, doi:10.1145/800070.802209.

[6] Marco Chiesa, Gábor Rétvári, and Michael Schapira. Oblivious routing in ip networks. *IEEE/ACM Transactions on Networking (TON)*, 26(3):1292–1305, 2018. URL: http://lendulet.tmit.bme.hu/~retvari/publications/ton_2018_2.pdf, doi:10.1109/TNET.2018.2832020.

[7] Lenore J Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001. URL: https://www.cs.tufts.edu/~cowen/sodasap.pdf, doi:10.1006/jagm.2000.1134.

[8] Paul Erdős. Extremal problems in graph theory. In *Proceedings of the Symposium on Theory of Graphs and its Applications*, pages 29–36, 1963. doi:10.1002/jgt.3190010206.

[9] Pierre Fraigniaud and Cyril Gavoille. Memory requirement for universal routing schemes. In *Proc. 14th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 223–230. ACM, 1995. doi:10.1145/224964.224989.

[10] Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 757–772. Springer, 2001. doi:10.1007/3-540-48224-5_62.

[11] Greg N Frederickson and Ravi Janardan. Designing networks with compact routing tables. *Algorithmica*, 3(1-4):171–190, 1988. URL: https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1595&context=cstech, doi:10.1007/BF01762113.

[12] Cyril Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News*, 32(1):36–52, 2001. URL: https://www.cs.princeton.edu/~jrex/teaching/spring2005/reading/Gav01.pdf, doi:10.1145/568438.568451.

[13] Cyril Gavoille and Stéphane Pérennès. Memory requirement for routing in distributed networks. In *Proc. 15th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 125–133. ACM, 1996. URL: https://dl.acm.org/doi/pdf/10.1145/248052.248075, doi:10.1145/248052.248075.

[14] Chris Harrelson, Kirsten Hildrum, and Satish Rao. A polynomial-time tree decomposition to minimize congestion. In *Proc. 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 34–43, 2003. URL: https://dl.acm.org/doi/pdf/10.1145/777412.777419, doi:10.1145/777412.777419.

[15] Christos Kaklamanis, Danny Krizanc, and Thanasis Tsantilas. Tight bounds for oblivious routing in the hypercube. *Mathematical Systems Theory*, 24(1):223–232, 1991. URL: https://dl.acm.org/doi/pdf/10.1145/97444.97453, doi:10.1145/97444.97453.

[16] Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM (JACM)*, 56(4):19, 2009. URL: https://people.eecs.berkeley.edu/~vazirani/pubs/partitioning.pdf, doi:10.1145/1538902.1538903.

[17] M. Kodialam, T.V. Lakshman, J.B. Orlin, and S. Sengupta. Oblivious routing of highly variable traffic in service overlays and ip backbones. *IEEE/ACM Transactions on Networking (TON)*, 17(2):459–472, 2009. doi:10.1109/TNET.2008.927257.

[18] Dmitri Krioukov, Kevin Fall, and Xiaowei Yang. Compact routing on internet-like graphs. In *Proc. IEEE INFOCOM*. IEEE, 2004. URL: https://arxiv.org/pdf/cond-mat/0308288.pdf, doi:10.1109/INFCOM.2004.1354495.

[19] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. Semi-oblivious traffic engineering: The road not taken. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 157–170, Renton, WA, April 2018. USENIX Association. URL: https://www.usenix.org/conference/nsdi18/presentation/kumar.

[20] Harald Racke. Minimizing congestion in general networks. In *Proc. 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 43–52. IEEE, 2002. URL: https://ttic.uchicago.edu/~harry/pdf/min_congestion.pdf, doi:10.1109/SFCS.2002.1181881.

[21] Harald Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proc. 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 255–264. ACM, 2008. URL: http://www.cs.cornell.edu/~abrahao/tdg/papers/p255.pdf, doi:10.1145/1374376.1374415.

[22] Harald Räcke and Stefan Schmid. Compact oblivious routing. In *Proceedings of the 27th European Symposium on Algorithms (ESA)*, 2019. URL: https://drops.dagstuhl.de/opus/volltexte/2019/11196/pdf/LIPIcs-ESA-2019-75.pdf, doi:10.4230/LIPIcs.ESA.2019.75.

[23] Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 227–238. Society for Industrial and Applied Mathematics, 2014. URL: https://epubs.siam.org/doi/pdf/10.1137/1.9781611973402.17, doi:10.1137/1.9781611973402.17.

[24] Gábor Rétvári, András Gulyás, Zalán Heszberger, Márton Csernai, and József J Bíró. Compact policy routing. *Distributed computing*, 26(5-6):309–320, 2013. URL: http://lendulet.tmit.bme.hu/~retvari/publications/podc_2011.pdf, doi:10.1007/s00446-012-0181-9.

[25] Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proceedings of the 13th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, SPAA 01, pages 1–10, New York, NY, USA, 2001. Association for Computing Machinery. URL: https://www.cs.bgu.ac.il/~elkinm/teaching/distr_comp/autumn16/tz_routing.pdf, doi:10.1145/378580.378581.

[26] Brian Towles and William J Dally. Worst-case traffic for oblivious routing functions. In *Proc. 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. ACM, 2002. doi:10.1109/L-CA.2002.12.

[27] Leslie G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982. doi:10.1137/0211027.

[28] Leslie G. Valiant and Gordon J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th ACM Symposium on Theory of Computing (STOC)*, pages 263–277, 1981. doi:10.1145/800076.802479.

[29] Jan van Leeuwen and Richard B Tan. Compact routing methods: A survey. In *Proc. Colloquium on Structural Information and Communication Complexity (SICC)*, pages 99–109, 1995.