Compact Oblivious Routing

Harald Räcke
Department of Informatics, TU München, Germany
raecke@in.tum.de

Stefan Schmid
Faculty of Computer Science, University of Vienna, Austria
stefan_schmid@univie.ac.at

Abstract

Oblivious routing is an attractive paradigm for large distributed systems in which centralized control and frequent reconfigurations are infeasible or undesired (e.g., costly). Over the last almost 20 years, much progress has been made in devising oblivious routing schemes that guarantee close to optimal load and also algorithms for constructing such schemes efficiently have been designed. However, a common drawback of existing oblivious routing schemes is that they are not compact: they require large routing tables (of polynomial size), which does not scale.

This paper presents the first oblivious routing scheme which guarantees close to optimal load and is compact at the same time – requiring routing tables of *polylogarithmic* size. Our algorithm maintains the polylogarithmic competitive ratio of existing algorithms, and is hence particularly well-suited for emerging large-scale networks.

2012 ACM Subject Classification Networks → Routing protocols; Theory of computation → Routing and network design problems; Networks → Network algorithms

Keywords and phrases Oblivious Routing, Compact Routing, Competitive Analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.72

1 Introduction

1.1 Motivation

With the increasing scale and dynamics of large networked systems, observing and reacting to changes in the workload and reconfiguring the routing accordingly becomes more and more difficult. Not only does a larger network and more dynamic workload require more fine-grained monitoring and control (which both introduce overheads), also the process of re-routing traffic itself (see e.g. [15]) can lead to temporary performance degradation and transient inconsistencies.

Oblivious routing provides an attractive alternative which avoids these reconfiguration overheads while being *competitive*, i.e., while guaranteeing route allocations which are almost as good as adaptive solutions. It is hence not surprising that oblivious routing has received much attention over the last two decades. Indeed, today, we have a good understanding of fast (i.e., polynomial-time constructable) and "competitive" oblivious routing algorithms (achieving a polylogarithmic approximation of the load, which is optimal).

However, while oblivious routing seems to be the perfect paradigm for emerging large networked systems, there is a fly in the ointment. Oblivious routing algorithms that aim to minimize congestion require large routing tables: namely *polynomial* in the network size. This is problematic, as fast memory in routers is expensive, not only in terms of monetary costs but also in terms of power consumption.

The goal of this paper is to design oblivious routing schemes which only require small routing tables (which are *compact*), and that at the same time still guarantee a close-to-optimal load.

1.2 The Problem in a Nutshell

The network is given as an undirected graph $G = (V, E)$ with n vertices. The edges E are weighted by a capacity function cap : $V \times V \to \mathbb{R}_0^+$; if $\{x, y\} \in E$, the function returns 0, otherwise a positive value.

The *oblivious routing problem* is to set up a unit flow for each source-target pair $(s, t) \in V \times V$ that determines how demand between s and t is routed in the network G. This unit flow is pre-specified without knowing the actual demands. When a demand vector $\vec{d}$ is given that specifies for each pair of vertices the amount of traffic to be sent, the demand-vector is routed by simply scaling the unit flow between a pair (s, t) by the corresponding demand d_{st} between the two vertices. This means that traffic is routed along *pre-computed paths* and that no path-selection is done dynamically.

The congestion $C_{\text{obl}}(G, \vec{d})$ resulting from a given oblivious routing scheme, is then compared to the optimal possible congestion $C_{\text{opt}}(G, \vec{d})$ that can be obtained for demand vector $\vec{d}$ in G. The *competitive ratio* of the oblivious routing scheme is defined as

$$\max_{\vec{d}} \frac{C_{\text{obl}(G,\vec{d})}}{C_{\text{opt}(G,\vec{d})}}$$

In this paper, we are interested in designing packet forwarding rules that allow the packets to follow a flow of an oblivious routing scheme with a good competitive ratio. Apart from the competitiveness of the underlying oblivious routing scheme one goal is to encode the forwarding rules *compactly* with small space requirement. In particular we require that for a vertex v the space requirement is only $O(\deg(v)\,\text{polylog}(n))$, where $\deg(v)$ is the degree of the vertex. In other words, this means we rougly require a polylogarithmic number of rules *per network link.* It seems unavoidable to let the memory requirement of a vertex depend on its degree as otherwise the routing scheme might not be able to efficiently utilize all network links.

In addition to the competitive ratio, the runtime, and the table size, we are also interested in the required vertex labels (i.e., their size) and the required packet header size.

1.3 Our Contributions

This paper presents the first *compact* oblivious routing scheme. Our approach builds upon an oblivious path selection scheme based on classic decomposition trees, which is then adapted to improve scalability, and in particular, to ensure small routing tables and message headers, while preserving polynomial runtime (for constructing the routing tables) and a polylogarithmic competitive ratio.

We present two different implementations of our approach and our results come in two different flavors accordingly (more detailed theorems will follow):

▶ **Theorem 1.** *There exist oblivious routing strategies that achieve a polylogarithmic competitive ratio w.r.t. the congestion and require routing tables of polylogarithmic size for*

1. *networks with arbitrary edge capacities which have a decomposition tree of bounded degree, and for*
2. *arbitrary networks with uniform edge capacities.*

Our algorithms only require small (polylogarithmic) header sizes and vertex labels. The routing tables can be constructed in polynomial time.

Networks for which there are decomposition trees of small degree include for example (constant-degree) grids. The exact requirements that a decomposition tree has to fulfill will be given later.

2 Algorithm and Analysis

This section describes an oblivious path selection scheme for general undirected networks that obtains close to optimal congestion and can be implemented with routing tables and routing headers of small size. In a nutshell, our algorithm leverages a path selection scheme for general networks that guarantees a good competitive ratio w.r.t. congestion, and then adapts it so that it can be implemented with small space requirements. We discuss the two phases of this algorithm in turn.

2.1 Oblivious Path Selection Scheme

There exist essentially two path selection schemes that could be used as a basis for our approach. First, there is the original result by Räcke [32] who showed that oblivious routing with a polylogarithmic competitive ratio is possible in general networks, using a *hierarchical path selection scheme* (cf. Section 2.1) that guarantees a competitive ratio of $O(\log^3 n)$. Second, there is a path selection scheme with an improved competitive ratio of $O(\log n)$ [33]. The latter scheme can be roughly viewed as a convex combination of spanning trees.[1] A path between a vertex s and a vertex t is chosen by sampling a random spanning tree and then choosing the path between s and t in this tree.

In this paper, we will build upon the original result [32] which we call the *hierarchical path selection scheme*. The challenge with implementing the path selection mechanism in [33] space-efficiently is that the number of spanning trees is quite large (polynomial in n). It seems difficult to avoid that a vertex in the graph has to store some information for every tree, which yields routing tables of polynomial size. The approach in [32] is based on a single tree which hence avoids the problem of [33].

The hierarchical path selection scheme is based on a hierarchical decomposition of the graph $G = (V, E)$. The vertex set V is recursively partitioned into smaller and smaller pieces until all pieces contain just single vertices of G. We will refer to the pieces/subsets arising during this partitioning process as *clusters*.

To such a recursive partitioning corresponds a decomposition tree $T = (V_T, E_T)$. A vertex x in this tree corresponds to cluster $V_x \subseteq V$ and there is an edge between a parent node p and a child node c if the cluster V_c arises from partitioning V_p. The root r of T corresponds to the subset $V_r = V$ and the leaf vertices correspond to singleton sets $\{v\}, v \in V$.

In order to simplify the notation and description we assume that all leaf vertices in T have the same distance to the root (this could e.g., be achieved by introducing dummy partitioning steps in which a set is partitioned into itself). We use h to denote the height of the tree. Let for a vertex $v \in V$, $a_\ell(v)$ denote the ancestor of $\{v\}$ on level ℓ of the tree, where the level of a vertex is its distance from the root. Here we use $\{v\}$ as a shorthand for 'the leaf node that corresponds to cluster $\{v\}$'. The *ℓ-weight of* v is the weight of all edges incident to v that leave the cluster $V_{a_\ell(v)}$. Formally $w_\ell(v) := \sum_{e=\{v,x\}:x\notin V_{a_\ell(v)}} \mathrm{cap}(e)$. We extend this definition to subsets of V by setting $w_\ell(U) := \sum_{u\in U} w_\ell(u)$ for every subset $U \subseteq V$.

We also introduce for every cluster S in the decomposition tree a weight function $w_S : S \mapsto \mathbb{R}_0^+$ and a weight function $\mathrm{out}_S : S \mapsto \mathbb{R}_0^+$. For a level ℓ-cluster S we define $w_S := w_{\ell+1}\lceil_S$ and $\mathrm{out}_S := w_\ell\lceil_S$, i.e., we define it as the restriction of $w_{\ell+1}$ and w_ℓ, respectively, to the vertex set of cluster S. Note that out_S counts edges that connect vertices

[1] This is not entirely correct as the trees are not proper spanning trees but the difference is not important for the above discussion.

of S to vertices outside of S while w_S also counts edges that connect different sub-clusters of S. We refer to w_S as the *cluster-weight* of S and to out_S as the *border-weight* of S.

Using this weight definition, we define a *concurrent multicommodity flow problem* (CMCF-problem) for every cluster S in the decomposition tree. For every (ordered) pair (u, v) there is a demand of $w_S(u)w_S(v)/w_S(S)$. Informally speaking, this means that every vertex injects a total flow that is equal to its w_S-weight and distributes this flow to the other vertices in S, proportionally to the w_S-weight of these vertices. We will use the decomposition tree T in [32] with the following properties:

- the height of T is $O(\log n)$, and
- for every cluster S in the decomposition tree, the CMCF-problem for S can be solved with congestion at most $C = O(\log^2 n)$ *inside* S.

Now suppose that we are given a decomposition tree with these properties. The path selection in [32] is then performed as follows. Suppose that we want to choose a path between vertices s and t in G. Let x_s and x_t denote the leaf vertices in T that correspond to singleton clusters $\{s\}$ and $\{t\}$, respectively. Let $x_s = x_1, x_2, \ldots, x_k = x_t$ denote the vertices in the tree on the path from x_s to x_t. We first choose a random vertex v_i from each cluster V_{x_i} according to the cluster-weight, i.e., the probability that v is chosen is $w_{V_{x_i}}(v)/w_{V_{x_i}}(V_{x_i})$. Note that $v_1 = s$ and $v_k = t$ as the corresponding clusters just contain a single vertex. It remains to select a path that connects the chosen vertices.

Suppose we want to connect two consecutive vertices v_p and v_c, where V_{x_p} is the parent cluster of V_{x_c}. We choose an intermediate vertex α inside V_{x_c} according to the border-weight of V_{x_c}, i.e., the probability that v is chosen is $\mathrm{out}_{V_{x_c}}(v)/\mathrm{out}_{V_{x_c}}(V_{x_c})$. We then consider the solution to the CMCF-flow problems for V_{x_c} and V_{x_p}. The first solution contains a flow $f(c, \alpha)$ between v_{x_c} and α, and the second contains a flow $f(p, \alpha)$ between v_{x_p} and α. We sample a random path from each flow. Concatenating these two paths, gives a flow between v_c and v_p. For the following analysis we call the sub-path between x_c and α the *lower sub-path* and the path between α and x_p the *upper sub-path.*

Concatenating all vertices v_i in the above manner gives a path between x_s and x_t. In the following we analyze the expected load generated on an edge due to this path selection scheme under the condition that an optimal algorithm can route the demand with congestion C_{opt}. For completeness and as we will need to modify this proof later, we repeat the following observations from [32].

▶ **Lemma 2.** *The expected load on an edge is at most $O(h \cdot C \cdot C_{\mathrm{opt}})$.*

Proof. Fix an edge e for which both end-points are contained in some cluster S. Let $S_1, \ldots, S_r$ denote the child-clusters of S. We first analyze the total demand that we have to route between a pair of vertices $(a, b) \in S \times S$ due to an upper sub-path where a is chosen as the intermediate vertex α and b is chosen as a random vertex from the parent cluster S. Assume $a \in S_i$ for some child cluster S_i. Then the probability that we choose a as α is $\Pr[a \text{ is chosen}] = \mathrm{out}_{S_i}(a)/\mathrm{out}_{S_i}(S_i)$. The probability that we choose b as the random end-point in S is $\Pr[b \text{ is chosen}] = w_S(b)/w_S(S)$. Note that any message for which we route between the child cluster S_i and the parent cluster S has to leave or enter the cluster S_i. Therefore the total demand for these messages can be at most $C_{\mathrm{opt}} \cdot \mathrm{out}_{S_i}(S_i)$, as otw. an optimum congestion of C_{opt} would not be possible. Hence, the expected demand for pair a

and b is only

$$\begin{aligned}\mathrm{out}_{S_i}(S_i)C_{\mathrm{opt}} \cdot \Pr[a \text{ is chosen}] \cdot \Pr[b \text{ is chosen}] &= \mathrm{out}_{S_i}(S_i)C_{\mathrm{opt}} \cdot \frac{\mathrm{out}_{S_i}(a)}{\mathrm{out}_{S_i}(S_i)} \cdot \frac{w_S(b)}{w_S(S)} \\ &= \frac{w_S(a) \cdot w_S(b)}{w_S(S)} \cdot C_{\mathrm{opt}} \ , \qquad (1)\end{aligned}$$

where we used the fact that $\mathrm{out}_{S_i}(a) = w_S(a)$, which holds since S_i is a direct child-cluster of S.

Now we analyze the demand that is induced for a pair $(a, b) \in S \times S$ due to the lower part of a message between S and its parent cluster. We assume that a is chosen as the intermediate vertex α and b is chosen as a random node in the child-cluster S. The probability that a is chosen as intermediate vertex is $\Pr[a \text{ is chosen}] = \mathrm{out}_S(a)/\,\mathrm{out}_S(S)$ and the probability that b is chosen is $\Pr[b \text{ is chosen}] = w_S(b)/w_S(S)$. Every such message has either to leave or enter cluster S. Hence, the total demand for these messages induced on pair (a, b) is at most

$$\begin{aligned}\mathrm{out}_S(S)C_{\mathrm{opt}} \cdot \Pr[a \text{ is chosen}] \cdot \Pr[b \text{ is chosen}] &= \mathrm{out}_S(S)C_{\mathrm{opt}} \cdot \frac{\mathrm{out}_S(a)}{\mathrm{out}_S(S)} \cdot \frac{w_S(b)}{w_S(S)} \\ &\le \frac{w_S(a) \cdot w_S(b)}{w_S(S)} \cdot C_{\mathrm{opt}} \ , \qquad (2)\end{aligned}$$

where we used the fact that $\mathrm{out}_S(a) \le w_S(a)$.

Combining Equation 1 and Equation 2 gives that the messages involving cluster S induce a demand of only $2w_S(a) \cdot w_S(b)/w_S(S) \cdot C_{\mathrm{opt}}$ between vertices a and b from S. Since we route this demand according to the multicommodity flow solution of the CMCF-problem for cluster S, the resulting load is at most $2C \cdot C_{\mathrm{opt}}$ on any edge *inside* cluster S, while edges not in S have a load of zero. Summing the load induced by messages for all clusters and exploiting the fact that an edge is at most contained in h different clusters, gives a maximum load of $2hC \cdot C_{\mathrm{opt}}$, i.e., a competitive ratio of $2hC$. ◀

Harrelson, Hildrum and Rao [21] present a decomposition tree in which the congestion for the CMCF-problem of clusters is not uniformly bounded by C but it is guaranteed that along a root-to-leaf path the congestion values of the respective flow problems sum up to at most $O(\log^2 n \log\log n)$. Then the expected load in Lemma 2 becomes $O(\log^2 n \log\log n \cdot C_{\mathrm{opt}})$. In addition the construction of this decomposition tree is polynomial time.

In the following description we base our oblivious routing scheme on the results in [32] as it slightly simplifies the write-up. For the theorems we also present the improved version obtained by plugging in the decomposition tree from Harrelson et al.

2.2 Implementation A: Decomposition Trees with Small Degree

We now present a space efficient implementation of the above path selection scheme. In the following, we will assume that the maximum degree of the decomposition tree T is small.

The basic building block for our implementation is a method that given a random starting point $v \in S$ chosen according to the cluster-weight of S (i.e., the probability of choosing v is $w_S(v)/w_S(S)$), routes to a random node $v_i \in S_i$ chosen according to the border weight of S_i. Here S_i is either a child-cluster of S (in case we want to communicate downwards in the tree) or $S_i = S$ (in case we want to communicate upwards). In the following we use $S_i, i \in \{1, \dots, r\}$ to denote the child-clusters of S and $S_0 = S$ to denote S itself. Let $G[S]$ denote the sub-graph induced by vertices in S.

For every $i \in \{0,\dots,r\}$ we compute a single commodity flow f_i in $G[S]$ as follows. We add a super-source s and connect it to every vertex $v \in S$ with an edge of capacity $w_S(v) \cdot \mathrm{out}_{S_i}(S_i)$ and a super-target t to which every vertex in $v \in S$ connects with capacity $\mathrm{out}_{S_i}(v) \cdot w_S(S)$. Note that all source edges together have the same capacity as the target edges.

We now compute a flow f_i between s_i and t_i that saturates all edges from s_i and to t_i. We can find such flows f_i (for all i) such that the *combined* congestion for these flows (on edges in S; edges from s_i's and to t_i's have congestion 1) is only $w_S(S) \cdot C$. To see this observe that in the CMCF-solution for cluster S the commodity (v_i, v) with $v_i \in S_i$ and $v \in S$ ships a flow of $\mathrm{out}_{S_i}(v_i) w_S(v)/w_S(S)$ between v_i and v. By 'merging' the flows of all commodities $(v_i, v) \in S_i \times S$ into a single commodity we obtain the desired flow (up to a $w_S(S)$-factor). Merging the commodities does not increase the congestion and, hence, the congestion is only $w_S(S) \cdot C$.

The flows f_i that we constructed so far may have fractional values that are difficult to store exactly. Therefore we slightly change the flows so that we can store them efficiently. For this we first scale every flow and the capacity of every edge up by a factor of r. Let f'_i and $\mathrm{cap}'(e)$ denote the scaled flows and capacities. Then every edge e makes a *capacity reservation* for every flow f_i. Suppose the (scaled) flow sends $f'_i(e)$ along edge e; then the edge e reserves a capacity of $\lceil f'_i(e) \rceil$ for the i-th flow. Note that the total capacity reservation is at most $\sum_i \lceil f'_i(e) \rceil \le \sum_i f'_i(e) + r \le w_S(S) \cdot C\, \mathrm{cap}'(e) + r \le 2w_S(S) \cdot C\, \mathrm{cap}'(e)$, because the scaled capacity of an edge is at least r.

Now we resolve every flow problem separately with the restriction that the flow should stay within its capacity reservation. This is clearly possible and since the capacity reservations and the demands are all integral we now have an *integral* flow f'_i. Undoing the scaling gives us flows f_i that can (concurrently) be routed with congestion at most $2w_S(S) \cdot C$, and where flow values are a multiple of $1/r$.

We now store the flows f_i in a distributed manner at the vertices of S, as follows. Fix $v \in S$. For every edge we store how much flow enters or leaves v. In order to route from the cluster-distribution of S to the border-distribution for S_i, $i \in \{0,\dots,r\}$, we choose random outgoing links (where a link is taken with probability proportional to the outgoing flow) until the chosen link is the super-target t. When we want to route from the border-distribution of S_i to the cluster-distribution of S, we take random incoming links (where a link is chosen with probability proportional to the incoming flow), until the chosen link corresponds to the super-source s. The proof of the following claim is analogous to Lemma 2.

▷ Claim 3. The expected load of an edge due to the path selection scheme is only $O(h \cdot C \cdot C_{\mathrm{opt}})$.

Proof. Suppose that the optimum congestion is C_{opt}. The total traffic that the scheme has to route between the cluster-distribution of S and the border-distribution of S_i is only $\mathrm{out}_{S_i}(S_i) \cdot C_{\mathrm{opt}}$. We route this traffic according to flow f_i of value $\mathrm{out}_{S_i}(S_i) w_S(S)$. Hence, the maximum load of an edge in $G[S]$ (according to original capacity) is $C \cdot C_{\mathrm{opt}}$.

Since an edge is contained in h different clusters the claim follows. ◀

▷ Claim 4. The path selection scheme can be implemented with routing tables of size $O(\deg(v)\deg(T)(\log m + \log W))$, labels of length $O(h \log(\deg(T)))$, and header length $O(h \log(\deg(T)))$.

Proof. Suppose that the capacities of the graph are integers in the range $\{1,\dots,W\}$. A flow value $f_i(e)$ along an edge is at most $w_S(S) \cdot W \cdot C$ (note that we assume that C is integral). Edges from s and to t have a capacity of $w_S(v)\,\mathrm{out}_{S_i}(S_i)$ and $w_S(S)\,\mathrm{out}_{S_i}(v)$, respectively.

Using the fact that $w_S(S)$ and $\mathrm{out}_{S_i}(S_i)$ are at most mW, and $d, C \leq m$ we get that a number describing the flow value along an edge can be encoded with

$$\log_2(m^2W^2r) = O(\log(m) + \log(W) + \log(r))$$

many bits (since flow values are a multiple of $1/r$). Hence, a node v has to store only $O(\deg(v)\deg(T)(\log m + \log W))$ many bits, where we used that $r \leq m$.

For the routing scheme we relabel the vertices. We number the children of a vertex in the tree and encode a leaf vertex by its path from the root. This generates labels of $O(h\log(\deg(T)))$ bits. The routing algorithm now only needs to have the label of the source vertex and the target vertex and a marker that marks where in the tree the routing currently is. ◀

In summary, we obtain the following theorem.

▶ **Theorem 5** (Decomposition Trees of Small Degree). *For decomposition trees of degree* $\deg(T)$ *one can construct an oblivious routing strategy that requires routing tables of size* $O(\deg(v)\deg(T)(\log(m) + \log(W)))$*, labels of length* $O(h\log(\deg(T)))$*, and header sizes of* $O(h\log(\deg(T)))$*. Depending on the decomposition tree used, we obtain two different competitive ratios:*

- *Using the decomposition tree from [32] the scheme guarantees a competitive ratio of* $O(hC) = O(\log^3(n))$ *w.r.t. congestion.*
- *Using the decomposition tree from [21] the scheme can be constructed in polynomial time and guarantees a competitive ratio of* $O(\log^2(n)\log\log(n))$*.*

2.3 Implementation B: Uniform Capacities

In this section we present a different implementation of the hierarchical routing scheme, for scenarios where the decomposition trees can be of arbitrary degree but where network capacities are uniform. Again the basic building block is to route from a node chosen according to the cluster-distribution of some cluster S to the border distribution of S_i where either $S_i = S$ or S_i is a child-cluster of S.

Assume that every edge in the graph G has capacity 1. We round the outgoing capacity $\mathrm{out}_{S_i}(S_i)$ of a child-cluster S_i, $i \geq 0$ to the next larger power of 2 and denote the rounded value with $\|S_i\|$. We also re-order the children w.r.t. this value, i.e., S_1 is the child-cluster with smallest $\|S_i\|$-value. Since there are at most m possible values for $\mathrm{out}_{S_i}(S_i)$, there are only $\log m$ possible values for $\|S_i\|$. There are only $\binom{r+\log m}{\log m}$ possibilities to choose the $\|S_i\|$-values of the r children of cluster S. Hence, we can store these with $O(\log(m)\cdot\log(r))$ many bits. In addition we store the value of $\|S_0\|$, which requires $O(\log\log m)$ bits, and the value of $w_S(S)$ which requires $O(\log m)$ bits.

In order to design the routing scheme for an individual cluster, we embed a hypercube of dimension $d := \lceil\log_2(\sum_{i\geq 0}\|S_i\|)\rceil$. We first order the hypercube nodes in an arbitrary way and then reserve a range of $\|S_i\|$ consecutive hypercube nodes for every $i \geq 0$ (the i-th range). Note that we store the (rounded) size of all children and that it is straightforward to compute the ranges assigned for any i from this information.

Then we map the hypercube nodes to nodes of S. First we map hypercube nodes in the i-th range to nodes with $\mathrm{out}_{S_i}(v) > 0$ such that each node receives at least $\mathrm{out}_{S_i}(v)$ and at most $2\,\mathrm{out}_{S_i}(v)$ hypercube nodes. Hypercube nodes that remain unmapped after this step (i.e., nodes that do not fall within any range) are mapped arbitrarily subject to the constraint that a cluster node v does not receive more than $8w_S(v)$ hypercube nodes. This can easily be done as the number of hypercube nodes (2^d) is at most $2\sum_i \|S_i\| \le 4\sum_i \sum_{v\in S_i} \mathrm{out}_{S_i}(v) = 4(w_S(S) + \mathrm{out}_S(S)) \le 8w_S(S)$.

▷ Observation 6. There are at most $8w_S(v)$ hypercube nodes mapped to any graph node.

For the embedding we set up a concurrent multicommodity flow problem as follows. For every edge $\{x, y\}$ of the hypercube that is mapped to endpoints $\{v_x, v_y\}$, we introduce a demand of 1 between v_x and v_y in both directions. Then every node sends and receives a total traffic of at most $8d \cdot w_S(v)$. By adding fake traffic we can turn this instance into a balanced multicommodity flow instance in which every vertex sends and receives a traffic of exactly $8d \cdot w_S(v)$.

We can solve this multicommodity flow instance with congestion at most $16dC$ inside the cluster S by using Valiant's trick [38, 25] of sending to random intermediate destinations and using the fact that each flow can send a traffic of $w_S(v)$ to random destinations with congestion C.

2.3.1 Using the Hypercube

How do we exploit the embedded hypercube? If during the routing scheme we are required to send a message from a cluster node v_p to a cluster node $v_c \in S_i$ we proceed as follows. Instead of choosing an intermediate node α according to probability distribution $\mathrm{out}_{S_i}(v)/\mathrm{out}_{S_i}(S_i)$ we choose a random hypercube node from the i-th range. Then we route a message inside the hypercube to this node. For this we let the message start at a random hypercube node from the nodes that are mapped to v_p.

Note that this means that the probability that the message is sent to node α lies between $\mathrm{out}_{S_i}(\alpha)/\|S_i\|$ and $2\,\mathrm{out}_{S_i}(\alpha)/\|S_i\|$ as the hypercube nodes in the i-th range are not mapped completely uniformly.

For the second part of the message we proceed analogously in the hypercube of S_i. We let the message start at a random hypercube node mapped to α and choose a random hypercube node as its target.

Again due to the non-uniform mapping, the target distribution on S_i (i.e., $w_{S_i}(v)/w_{S_i}(S_i)$) is not reached exactly, but deviations by a constant factor might occur. This only influences the congestion of a single step by a constant factor, but it could be problematic if we used this approach along a path in the tree: in each step the distribution would change by a constant factor.

Therefore, we add an additional step that fixes the distribution over S_i. We embed an additional hypercube H_S for every cluster S with dimension $\lceil \log_2(w_S(S)) \rceil$. The mapping is done such that each cluster-vertex $v \in S$ receives exactly $w_S(v)$ hypercube nodes among the first $w_S(S)$ nodes from H_S (the remaining nodes are distributed uniformly). Since every node in the cluster S stores the value of $w_S(S)$, we can route from a node $v \in S$ to a random node chosen according to distribution $w_S(v)/w_S(S)$, by just selecting a random hypercube node from the first $w_S(S)$ nodes.

2.3.2 Analysis

We showed that the congestion for sub-messages that involve cluster S is small. There are two types of such messages:

1. messages that start at an intermediate node (distributed according to the border weight of S_i for some $i \ge 0$) and are sent to a random node $v \in S$ distributed according to the cluster-weight of S; and

2. messages that start at a random node $v \in S$ and are sent to some intermediate node.

It was shown that the total traffic that is sent between a pair v_i and v, where v is distributed according to the cluster weight of S and v_i is distributed according to the border weight of S_i, is only $\mathrm{out}_{S_i}(v_i) w_S(v)/w_S(S) \cdot C_{\mathrm{opt}}$.

In our new scheme this changes slightly. For messages of the second type the source is distributed as before but the target may have a slightly different distribution (as we choose a random hypercube node in the i-th range). For messages of the first type already the source may have a slightly different distribution (as we choose a random hypercube node from some range in the hypercube for a child- or parent-cluster). Also the target distribution is slightly skewed as we choose a random hypercube node as the target.

But since the distributions are only changed by a constant factor this difference does not really influence our analysis. We still have the property that the traffic between v_i and v_S is $\Theta(\mathrm{out}_{S_i}(v_i) w_S(v)/w_S(S) \cdot C_{\mathrm{opt}})$.

The second difference is that the traffic is not sent according to the CMCF-problem for cluster S but it is instead sent along the hypercube. Note that due to the embedding of the hypercube, a cluster node $v \in S_i$ has $\Theta(\mathrm{out}_{S_i}(v)) = \Theta(w_S(v))$ hypercube nodes in the i-th range mapped to it (i.e., hypercube nodes are balanced perfectly up to constant factors). Hence the demand between v_i and v will be split among $\Theta(\mathrm{out}_{S_i}(v_i) w_S(v))$ pairs in the cube. Therefore the demand for every pair in the cube is only $\Theta(C_{\mathrm{opt}}/w_S(S)) = \Theta(C_{\mathrm{opt}}/2^d)$. This means that at most a traffic of $O(C_{\mathrm{opt}})$ starts and ends at every vertex and routing this traffic using Valiant's trick gives a congestion of $O(dC_{\mathrm{opt}})$ in the hypercube. Since we embedded the hypercube with congestion $O(dC)$, the congestion of a graph edge will be $O(d^2 C \cdot C_{\mathrm{opt}})$ (as each hypercube node has degree d), which gives rise to the following lemma.

▶ **Lemma 7.** *Implementation B guarantees a maximum expected load of $O(hd^2 C \cdot C_{\mathrm{opt}})$.*

Proof. The lemma follows by applying the previous argument for each level of the tree.

It remains to bound the edge-load induced by the re-randomization steps. The total traffic that is send to a cluster S in the tree is at most $(\sum_i \mathrm{out}(S_i)) \cdot C_{\mathrm{opt}} = \Theta(w_S(S) \cdot C_{\mathrm{opt}})$. For each such message we require a re-randomization, because in our current scheme, it is only distributed approximately according to the cluster-weight of S.

However by design each vertex receives exactly a $w_S(v)/w_S(S)$-fraction of the re-randomization messages, and a $\Theta(w_S(v)/w_S(S))$-fraction of messages start at v, since the messages are approximately distributed according to cluster-weight. Sending these messages along the hypercube introduces congestion $\Theta(d \cdot C_{\mathrm{opt}})$ in the cube and $\Theta(d^2 C \cdot C_{\mathrm{opt}})$ due to the embedding. ◀

▶ **Lemma 8.** *Implementation B requires space $O(hC \log(m) \log\log(m) \deg(v))$ bits at every vertex and a label and header length of $O(h \log(\deg(T)))$.*

Proof. We will use the following helper lemma:

▶ **Lemma 9.** *Let $X_1, \dots, X_n$ denote a set of* negatively correlated *random variables taking values in the range $[0, 1]$. Let X denote their sum and let $\mu \leq E[X]$ denote a lower bound on the expectation of X. Then for any $\delta \geq 1$*

$$\Pr[X \geq (1 + \delta)\mu] \leq e^{-\delta\mu/3} \ .$$

A vertex $v \in S$ has to store the approximate size $\|S_i\|$ of the child-clusters of S. Summing this over all levels gives $O(h \log(m) \cdot \log(r))$ bits. In addition one has to encode the embedding of the hypercubes. The congestion of the solution to the concurrent multicommodity flow problem for embedding a hypercube is $O(dC)$. This fractional solution will encode a flow for every hypercube edge. Using a standard randomized rounding approach, we can route the flows to paths with a congestion of $O(dC + \log(m)) = O(dC)$. This is done as follows. For every pair $\{x, y\}$ we take the unit flow and first decompose this unit flow into flow-paths. Then we choose for every pair one of the flow-paths at random (proportional to its weight). Let $X_i(e)$ denote the random variable that describes whether the flow path for the i-th pair includes edge e. By design the above process guarantees $E[X_i(e)] = f_i(e)$, where $f_i(e)$ is the flow for pair i that goes through edge e. The total load on edge e is $\sum_i X_i(e)$. This is a sum of negatively correlated random variables with expectation $\mu = O(dC)$. Using Lemma 10 with $\delta = 3\ln(m)/\mu$ gives that with constant probability, no edge exceeds load $O(dC + \log m)$.

Therefore only $O(\deg(v)dC)$ paths traverse a vertex v (recall that $C \geq \log m$). For every path, we need to store the outgoing edge and the id of the paths on this edge. This requires $(\log_2(\deg(v)) + \log_2(dC))$ bits for every path and $O(d\deg(v)C\log(d\deg(v)C))$ bits in total. Multiplying with the height and using $d = \Theta(\log m)$ gives $O(hC\log(m)(\log(\deg v) + \log\log m)\deg(v)) = O(hC\log^2(m) \cdot \deg(v))$ bits.

The header- and label-length is analogous to Implementation A. We just use the root-to-leaf path in the tree as a label and a header consists of the source-label, the target-label, and a marker. ◀

In summary we derived the following result:

▶ **Theorem 10** (Compact Oblivious Routing for Uniform Capacities). *For arbitrary networks of uniform capacities, there exists an oblivious routing strategy which requires label and header length of $O(h\log(\deg(T)))$, and which comes in two flavors, depending on the decomposition tree used:*

- *Using the decomposition tree from [32] the scheme requires $O(hC\log^2(m)\deg(v)) = O(\log^5(n)\deg(v))$ bits at every vertex and guarantees a competitive ratio of $O(hC\log^2(n)) = O(\log^5(n))$ w.r.t. congestion.*
- *Using the decomposition tree from [21] the scheme requires $O(\log^2 m\log^2(n)\log\log n)\deg(v)) = O(\log^4 n\log\log n)\deg(v))$ bits at every vertex and guarantees a competitive ratio of $O(\log^4(n)\log\log(n))$ w.r.t. congestion. The oblivious routing scheme can be constructed in polynomial time.*

3 Related Work

The drawbacks of ***adaptive routing*** have been discussed intensively in the literature, see e.g., [34] for a survey. In particular, adaptive routing schemes need global information about the routing problem in order to calculate the best paths, and even if it were possible to collect such information sufficiently fast, it can still take much time to compute a (near-)optimal solution to that problem (large linear programs may have to be solved).

One of the first and well-known results on ***oblivious routing*** is due to Borodin and Hopcroft [8] who showed that competitive oblivious routing algorithms require randomization, as deterministic algorithms come with high lower bounds: given an unweighted network with n nodes and maximum degree Δ, there exists a (permutation) routing instance such that the congestion induced by a given deterministic oblivious routing scheme is at least $\Omega(\sqrt{n}/\Delta^{3/2})$. This result was improved by Kaklamanis et al. [22] to a lower bound of $\Omega(\sqrt{n}/\Delta)$.

For randomized algorithms Valiant and Brebner [38] showed how to obtain a polylogarithmic competitive ratio for the hypercube by routing to random intermediate destinations. Räcke [32] presented the first oblivious routing scheme with a polylogarithmic competitive ratio of $O(\log^3 n)$ in general networks. The paper by Räcke was also the first to propose designing oblivious routing schemes based on cut-based *hierarchical decompositions*. However, Räcke's result is non-constructive in the sense that only an exponential time algorithm was given to construct the hierarchy. This approach has subsequently been used to obtain approximate solutions for a variety of cut-related problems that seem very hard on general graphs but that are efficiently solvable on trees, see e.g. [2, 3, 5, 10, 14, 23, 26, 33]. Polynomial-time algorithms for constructing the hierarchical decomposition were given by Bienkowski et al. [7] and Harrelson et al. [21]. However, none of these results provide an (asymptotically) optimal competitive ratio.

Azar et al. [4] gave a polynomial time algorithm that for a given graph computes the optimal oblivious routing via a linear programming approach, i.e., without using a hierarchical decomposition.

An optimal competitive ratio of $O(\log n)$ (which matches a known lower bound from grids [6, 30]) was first presented by Räcke [33] . Instead of considering a single tree to approximate the cut-structure of a graph G, [33] proposes to use a convex combination of decomposition trees. The paper relies on multiplicative weight updates and the proof technique is similar to the technique used by Charikar et al. [9] for finding a probabilistic embedding of a metric into a small number of dominating tree metrics.

More recently, inspired by the ideas on cut matching games introduced by Khandekar, Rao, and Vazirani [24], Räcke et al. [35] presented a *fast* construction algorithm for hierarchical tree decompositions, i.e., for a single tree: given an undirected graph $G = (V, E, c)$ with edge capacities, a single tree $T = (V_T, E_T, c_T)$ can be computed whose leaf nodes correspond to nodes in G and which approximates the cut-structure of G up to a factor of $O(\log^4 n)$ (i.e., the faster runtime comes at the price of a worse approximation guarantee). In particular, the authors present almost linear-time cut-based hierarchical decompositions, by establishing a connection between approximation of max flow and oblivious routing. This overcomes the major drawback of earlier algorithms such as [21] and even [29] which required high running times for constructing the decomposition tree (or the distribution over decomposition trees). The bound has been improved further by Peng in [31].

Previous results on ***compact routing*** focus on routing strategies that aim to minimize the *path length* instead of the congestion (see e.g. [13, 27, 39]). There are two variants: *labeled* (the designer is free to name the nodes according to the topology and the edge weights of the graph) and *name-independent* (name determined by an adversary). The research community has derived many interesting results on compact shortest path routing on special graphs, e.g., characterizing hypercubes, trees, scale-free networks, and planar graphs [17, 18, 19, 28, 37]. A well-known recent result on compact routing in the name-independent model on *general graphs* is by Abraham et al. [1].

However, it is also known that it is impossible to implement shortest path routing with routing tables whose size in all network topologies grows slower than linear with the increase of the network size [16, 20]. As a resort, compact routing research studies algorithms to decrease routing table sizes at the price of letting packets to be routed along suboptimal paths. In this context, suboptimal means that the forwarding paths are allowed to be longer than the shortest ones, but the length increase is bounded by a constant stretch factor. A particularly interesting result is by Thorup et al. [37] who presented compact routing schemes for general weighted undirected networks, ensuring small routing tables, small headers and low stretch. The approach relies on an interesting shortest path routing scheme for trees of arbitrary degree and diameter that assigns each vertex of an n-node tree a label of logarithmic size. Given the label of a source node and the label of a destination it is possible to compute, in constant time, the port number of the edge from the source that heads in the direction of the destination. An interesting recent work by Retvari et al. [36] generalizes compact routing to arbitrary routing policies that favor a broader set of path attributes beyond path length. Using routing algebras, the authors identify the algebraic requirements for a routing policy to be realizable with sublinear size routing tables.

There also exist interesting works focusing on *scalable* oblivious traffic engineering, e.g., for confluent (per-destination) routing schemes [11, 12]. However, we are not aware of any results on compact oblivious routing which provides polylogarithmic approximation ratios along both dimensions, table size and congestion.

4 Conclusion

Given the fast growth of communication networks (e.g., due the advent of novel paradigms such as Internet-of-Things), the high costs of network equipment (e.g., fast memory is expensive and power hungry), as well as the increasing miniaturization of communication-enabled devices, we in this paper initiated the study of oblivious routing schemes which only require small routing tables. In particular, we presented the first *compact* oblivious routing scheme, requiring polylogarithmic tables only (as well as polylogarithmic packet headers and vertex labels).

We believe that our work opens an interesting avenue for future research. In particular, while our algorithms provide poly-logarithmic routing tables and competitive ratios, it may be possible to further improve these results by logarithmic factors. Furthermore, it would be interesting to generalize our results to non-uniform network capacities, as well as to explore whether our results can be improved for special network topologies arising in practice.

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.

2 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Seffi Naor. A general approach to online network optimization problems. *ACM Transactions on Algorithms (TALG)*, 2(4):640–660, 2006.

3 Konstantin Andreev, Charles Garrod, Daniel Golovin, Bruce Maggs, and Adam Meyerson. Simultaneous source location. *ACM Transactions on Algorithms (TALG)*, 6(1):16, 2009.

4 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.

5 Nikhil Bansal, Uriel Feige, Robert Krauthgamer, Konstantin Makarychev, Viswanath Nagarajan, Joseph Naor, and Roy Schwartz. Min-max graph partitioning and small set expansion. In *Proc. IEEE 52nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 17–26. IEEE, 2011.

6 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.

7 Marcin Bienkowski, Miroslaw Korzeniowski, and Harald Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proc. 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 24–33. ACM, 2003.

8 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.

9 Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge Plotkin. Approximating a finite metric by a small number of tree metrics. In *Proc. 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 379–388. IEEE, 1998.

10 Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. The all-or-nothing multicommodity flow problem. In *Proc. 36th Annual ACM Symposium on Theory of Computing (STOC)*, pages 156–165. ACM, 2004.

11 Marco Chiesa, Gábor Rétvári, and Michael Schapira. Lying your way to better traffic engineering. In *Proc. ACM 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, pages 391–398, 2016.

12 Marco Chiesa, Gábor Rétvári, and Michael Schapira. Oblivious routing in ip networks. *IEEE/ACM Transactions on Networking (TON)*, 26(3):1292–1305, 2018.

13 Lenore J Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 38(1):170–183, 2001.

14 Roee Engelberg, Jochen Könemann, Stefano Leonardi, and Joseph Seffi Naor. Cut problems in graphs with a budget constraint. In *Proc. Latin American Symposium on Theoretical Informatics (LATIN)*, pages 435–446. Springer, 2006.

15 Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio. Survey of consistent software-defined network updates. In *IEEE Communications Surveys and Tutorials (COMST)*, 2018.

16 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.

17 Pierre Fraigniaud and Cyril Gavoille. Routing in trees. In *Proc. International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 757–772. Springer, 2001.

18 Greg N Frederickson and Ravi Janardan. Designing networks with compact routing tables. *Algorithmica*, 3(1-4):171–190, 1988.

19 Cyril Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News*, 32(1):36–52, 2001.

20 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.

21 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.

22 Christos Kaklamanis, Danny Krizanc, and Thanasis Tsantilas. Tight bounds for oblivious routing in the hypercube. *Mathematical Systems Theory*, 24(1):223–232, 1991.

23 Rohit Khandekar, Guy Kortsarz, and Vahab Mirrokni. On the advantage of overlapping clusters for minimizing conductance. *Algorithmica*, 69(4):844–863, 2014.

24 Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *Journal of the ACM (JACM)*, 56(4):19, 2009.

25 Petr Kolman and Christian Scheideler. Improved bounds for the unsplittable flow problem. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 184–193. Society for Industrial and Applied Mathematics, 2002.

26 Jochen Könemann, Ojas Parekh, and Danny Segev. A unified approach to approximating partial covering problems. In *Proc. European Symposium on Algorithms (ESA)*, pages 468–479. Springer, 2006.

27 Dmitri Krioukov, Kevin Fall, Arthur Brady, et al. On compact routing for the internet. *ACM SIGCOMM Computer Communication Review (CCR)*, 37(3):41–52, 2007.

28 Dmitri Krioukov, Kevin Fall, and Xiaowei Yang. Compact routing on internet-like graphs. In *Proc. IEEE INFOCOM*. IEEE, 2004.

29 Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *Proc. 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 245–254. IEEE, 2010.

30 Bruce M Maggs, F Meyer auf der Heide, Berthold Vocking, and Matthias Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proc. 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 284–293. IEEE, 1997.

31 Richard Peng. Approximate undirected maximum flows in o(m polylog (n)) time. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1862–1867. Society for Industrial and Applied Mathematics, 2016.

32 Harald Räcke. Minimizing congestion in general networks. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 43–52, 2002. doi:10.1109/SFCS.2002.1181881.

33 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.

34 Harald Räcke. Survey on oblivious routing strategies. In *Proc. 5th Conference on Computability in Europe: Mathematical Theory and Computational Practice (CiE)*, pages 419–429, 2009.

35 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.

36 Gábor Rétvári, András Gulyás, Zalán Heszberger, Márton Csernai, and József J Bíró. Compact policy routing. *Distributed computing*, 26(5-6):309–320, 2013.

37 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. ACM, 2001.

38 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.

39 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.