Minimizing Congestion in General Networks

Harald Räcke*
Heinz Nixdorf Institute and
Department of Mathematics and Computer Science
Paderborn University, Germany
harry@upb.de

Abstract

A principle task in parallel and distributed systems is to reduce the communication load in the interconnection network, as this is usually the major bottleneck for the performance of distributed applications. In this paper we introduce a framework for solving on-line problems that aim to minimize the congestion (i.e. the maximum load of a network link) in general topology networks.

We apply this framework to the problem of on-line routing of virtual circuits and to a dynamic data management problem. For both scenarios we achieve a competitive ratio of $O(\log^3 n)$ with respect to the congestion of the network links.

Our on-line algorithm for the routing problem has the remarkable property that it is oblivious*, i.e., the path chosen for a virtual circuit is independent of the current network load. Oblivious routing strategies can easily be implemented in distributed environments and have therefore been intensively studied for certain network topologies as e.g. meshes, tori and hypercubic networks. This is the first oblivious path selection algorithm that achieves a polylogarithmic competitive ratio in general networks.*

1. Introduction

In large parallel and distributed systems, such as networks of workstations or the Internet, the bandwidth of the interconnection network usually is the major bottleneck for the performance of distributed applications. This is due to the fact that it is often more expensive or more difficult to increase the bandwidth of the interconnection network than to increase processor speed and memory capacities at individual nodes.

Therefore, a principle task for these systems is to design distributed applications in such a way that they cause as little communication overhead as possible. However, it may not be sufficient to simply reduce the total communication load, i.e., the total traffic in the network, as this can result in bottlenecks. In addition, the load has to be distributed evenly among all network resources. This corresponds to minimizing the congestion, i.e., the maximum taken over all network links of the amount of data transmitted by the link divided by the respective bandwidth.

In this paper we introduce a general framework for minimizing the congestion produced by applications in distributed environments. This framework can be applied to many online problems that aim to minimize the congestion. If successful it yields a solution to the online problem on arbitrary networks that is fully distributed, i.e., all decisions are made using only local knowledge of the network load. The framework is based on representing all network links and thus the whole topology of the network as a tree. Since a tree topology is very simple many online problems can be solved efficiently for trees. By combining a tree solution of an online problem and the tree representation of the network, the framework yields a solution that minimizes the congestion of the network links. The framework is evaluated by applying it to the following two fundamental problems of distributed computing.

The first one is the well known online routing problem (see e.g. [2, 10]). In this problem routing requests consisting of a source and a destination point, arrive online. For each request the routing algorithm has to specify a path from the source to the destination in the network. The goal is to minimize the congestion of the network links.

The second problem is a data management problem that was introduced in [11]. Here, the nodes of the network issue read and write requests to shared data objects. A data management algorithm for this problem has to decide where to place copies of shared objects in the network and how to access these copies such that the link congestion is minimized.

*Partially supported by DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen" and by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT)

1.1. Definition of the model

We model the network by a weighted graph $G = (V,E)$ of $|V| = n$ nodes and $|E|$ edges. The nodes represent the processors, the edges represent the links and the function $\mathrm{b} : E \to \mathbb{R}^+$ represents the bandwidth of the links. A sequence σ of requests, e.g. read or write requests to shared objects, are issued at the nodes of the network and have to be served by an online algorithm. In order to obtain a simple and unambiguous model we assume that these requests are issued one after the other, i.e., σ_{i+1} is not issued before σ_i is not fully processed by the online algorithm. Note however that the algorithms that we will develop for this sequential model can handle parallel and overlapping requests, too.

The performance of an online algorithm A is evaluated by comparing it to an *optimal offline algorithm OPT*, which has full knowledge of the input sequence in advance and can process it optimally. Given an input sequence σ, let $C_A(\sigma)$ and $C_{OPT}(\sigma)$ denote the congestion produced by A and OPT, respectively, when processing σ. Algorithm A is called c-competitive if there exists a constant a (independent of σ) such that $C_A(\sigma) \le c \cdot C_{OPT}(\sigma) + a$, for any sequence σ. The algorithm is said to be strictly competitive if $a = 0$. In general we assume that the online algorithm may use randomization. In this case it must fulfill the above bound with high probability.[1]

In the framework of competitive analysis the input sequence is viewed as chosen by an adversary in order to reflect that the above bound has to hold for all input sequences. In the case of randomized algorithms one has to specify the exact power that is given to the adversary. Throughout this paper we assume that the adversary is *oblivious*, i.e., the request sequence is not allowed to depend on the random choices made by the online algorithm.

For a given algorithm and an input sequence σ we define the *(absolute) load* of a link $e \in E$ to be the amount of data that must be transferred by this connection when σ is processed by the algorithm. We define the *relative load* of a connection to be its load divided by its bandwidth. Finally, we define the *congestion* to be the maximum over the relative loads of all links in the network. The cost metrics for the online routing and the data management problem are defined as follows.

Online routing. In the online routing problem the request sequence consists of routing requests between pairs of nodes. For each request a path connecting the corresponding nodes in the network has to be specified. This increases the load on each edge of the path by one.

Data management. In the data management problem the request sequence consists of read and write requests to shared data objects. A data management strategy is allowed to migrate, create and invalidate copies at execution time. We assume that any message, including messages for migrating copies, messages for locating copies in the network as well as messages for invalidating copies, increases the load on every edge of the respective path by one.

Note that the above uniform definition of the load associated with messages is only chosen for keeping the description of strategies as simple as possible. All results can easily be extended to non-uniform definitions, as well.

1.2. Previous work and new results

A centralized, $O(\log n)$-competitive online routing algorithm was first presented by Aspnes et al. in [2]. This algorithm is based on the use of an exponential cost function. Each edge e is assigned a length that is exponential in the current load of e. If a routing request occurs the algorithm chooses a shortest path between source and destination with respect to the length assigned to the edges. The competitive ratio of this algorithm is optimal due to a lower bound provided in the same paper. The drawback of this algorithm is that it is centralized and that the current network load must be known in order to make a routing decision.

Awerbuch and Azar [4] improve on this result by developing an algorithm that is not centralized, i.e., there exist several agents that make routing decisions in the network. But also in their scenario it is still difficult for a distributed implementation to realize the up-to-date knowledge of the state of the network for the agents.

In this paper we show the somehow surprising result that knowledge of the current load in the network is not needed in order to achieve a polylogarithmic competitive ratio, w.r.t. the congestion. We present an *oblivious* online routing algorithm, i.e., a routing algorithm where the path selected for the *i-th* request σ_i does not depend on routing decisions made for other requests σ_j, $j \neq i$. Hence, the path may only depend on the source of the request, its destination and on some random input. Note that randomization is essential, for the design of an oblivious routing algorithm with low competitive ratio, since there exist large lower bounds for deterministic routing protocols (see [5, 7]). Oblivious routing strategies can easily be implemented in distributed environments and therefore many oblivious strategies have been designed for special network topologies (see e.g. [12, 14]). In [1] the authors provide a lower bound for oblivious routing on the hypercube.

To our knowledge this work is the first analytical treatment of oblivious routing strategies for general networks.

The online data management problem with the goal of minimizing the congestion of the network links was investigated in [11]. There the authors present a 3-competitive online algorithm for trees. Furthermore, they show how to achieve an $O(\log n)$-competitive algorithm for meshes via a

[1] Throughout the paper, congestion C, w.h.p. (with high probability) means congestion at most C+x, with probability at least $1 - 2^{-\Omega(x)}$.

bi-simulation between a mesh and a tree network. In [13] it is claimed that this approach can be extended to other networks by providing a suitable, hierarchical decomposition for the target network. In this paper we define the exact properties that the hierarchical decomposition has to fulfill and we prove that such a hierarchical decomposition exists for any network. This gives an $O(\log^3 n)$-competitive data management strategy for general graphs.

1.3. Outline of the framework

In the following we sketch the general concept for solving online problems in the congestion based model for arbitrary networks. The framework is based on a bi-simulation between a target network $G = (V,E)$ and an associated tree network $T_G = (V_t, E_t)$, the so-called *decomposition tree* which is suitably constructed out of G. Both networks are weighted with $\mathrm{b} : E \to \mathbb{R}^+$ and $\mathrm{b}_t : E_t \to \mathbb{R}^+$ describing the bandwidth of edges in E and E_t, respectively.

In a first simulation step it is shown that the tree T_G can simulate the network G, i.e., any request sequence σ for an online problem that produces congestion C when it is processed on G can be processed on T_G with congestion C, too. For this simulation we will present a mapping $\pi_1 : V \to V_t$ from the nodes of G to the nodes of T_G. A processor $v \in V$ of G is then simulated by its counterpart $\pi_1(v)$ in T_G. Note that this first simulation step is completely independent of the online problem that has to be solved.

In a second simulation step we show for certain online problems that the network G can simulate the tree T_G in such a way that the congestion is only a small factor c larger than the congestion on the tree. More concretely, if a request sequence σ can be processed on the tree with congestion C_t, then it can be processed on G with congestion $c \cdot C_t + K$, w.h.p., where K is a constant independent of σ. For this simulation a node $v \in V$ may have to simulate several nodes of V_t, since usually $|V_t| > |V|$. We denote the (randomized) mapping used for the second simulation step $\pi_2 : V_t \to V$. Additionally for this simulation step we describe how the routing between host nodes in G that simulate adjacent tree nodes is performed. (Note that this is easy for the first simulation since T_G is a tree.) The exact description of the routing and the definitions of π_1 and π_2 will be given in Section 3. For the purpose of this section it is only important that $\pi_2(\pi_1(v)) = v$ holds for each $v \in V$.

How do these simulation results help us in solving congestion-based online problems on general networks? A condition for our framework is that the online problem can be solved efficiently for trees. Thus, suppose that there is a c_t-competitive algorithm for the tree T_G (for some value c_t).

We get an online strategy for G as follows. Suppose we are given a sequence σ of requests for the online problem. Let $C_{\mathrm{opt}}(G)$ and $C_{\mathrm{opt}}(T_G)$ denote the optimal congestion if σ is processed on G and T_G, respectively. (For processing σ on T_G we assume that a request originally issued at a node $v \in V$ is issued at $\pi_1(v) \in V_t$, instead.) Furthermore, let $C_{\mathrm{onl}}(T_G)$ denote the congestion produced by the online tree strategy when processing σ on T_G. The first simulation step and the c_t-competitiveness yield

$$C_{\mathrm{opt}}(G) \geq C_{\mathrm{opt}}(T_G) \geq \frac{1}{c_t} \cdot C_{\mathrm{onl}}(T_G) - K_t \ ,$$

where K_t is some constant that does not depend on σ.

Now, we can simulate this online tree strategy on the network G and we get an online strategy for G. Let $C_{\mathrm{onl}}(G)$ denote the congestion of this strategy for request sequence σ. Note that since $\pi_2(\pi_1(v)) = v$ for every $v \in V$ the bi-simulation does not reorder the nodes of V and in particular does not change the request sequence σ. (For the first simulation step requests from a node $v \in V$ are mapped to $\pi_1(v) \in V_t$. For the second simulation such a request is mapped to $\pi_2(\pi_1(v)) = v$.)

Let $C_{\mathrm{onl}}(G)$ denote the congestion of this strategy for request sequence σ and let c denote the factor between the congestion in G and T_G for the second simulation step. We get

$$C_{\mathrm{onl}}(G) \leq c_t \cdot c \cdot C_{\mathrm{opt}}(G) + K_t + K \ .$$

Hence, the online strategy for G is $c_t \cdot c$-competitive.

The crucial part of the proof is to choose T_G in such a way that the second simulation step can be performed with a small factor c. The remainder of the paper is organized as follows. In the following section we will describe the general method for choosing T_G and define certain parameters for T_G that mainly influence the quality of the simulations. In Section 3 we will then show that given a decomposition tree T_G with "good" parameters both simulations can be performed with a small factor c. Finally we will show in Section 4 that for any network G there is a tree T_G such that all parameters relevant for the simulations have "good" values.

2. The decomposition tree

In this section we describe the general method for choosing the decomposition tree T_G for a network $G = (V,E)$ in such a way that both simulations work properly. The decomposition tree T_G corresponds to a decomposition of the node set V of G into a system of subsets of V. We need the following notations to characterize certain properties of such a set system. Two subsets X and Y are called *intersecting* if neither of $X \setminus Y$, $Y \setminus X$ and $X \cap Y$ are empty. A set system is called *laminar* if it contains no intersecting pair. Furthermore, a laminar set system of subsets of V is *complete* if it contains the set V and all sets $\{v\}, v \in V$.

Given a complete laminar system $\mathcal{S}$ containing subsets of V we construct a decomposition tree $T_G = (V_t, E_t)$ as fol-

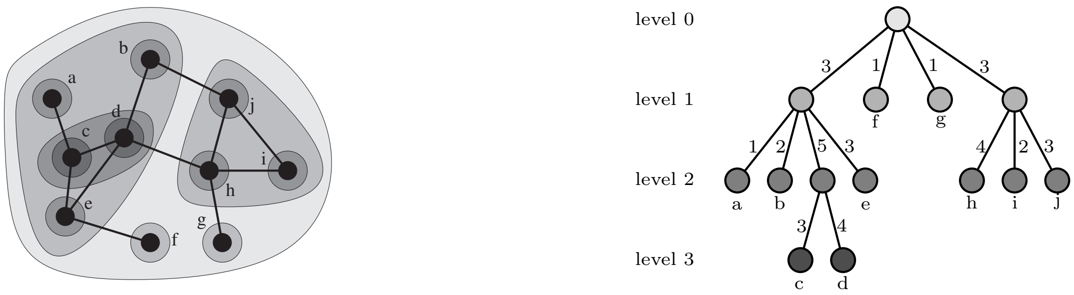

Figure 1. A set system for a graph and the associated decomposition tree. Edge labels in the right figure indicate the bandwidth of the respective edge. (Edges in the left figure have bandwidth 1.)

lows. For each $S \in \mathcal{S}$ the set V_t contains a node v_t. We call S the set or the cluster corresponding to v_t, and v_t the (tree) node corresponding to S. In the following the cluster corresponding to a node $v_t \in V_t$ will be denoted with S_{v_t}. Two nodes u_t and v_t in T_G are connected if $S_{u_t} \subset S_{v_t}$ or $S_{u_t} \supset S_{v_t}$ and if there is no $S \in \mathcal{S}$ such that $S_{u_t} \subset S \subset S_{v_t}$ or $S_{u_t} \supset S \supset S_{v_t}$, respectively. Note that by this definition T_G is indeed a tree, since $\mathcal{S}$ is a laminar system. We assume T_G to be rooted at the node corresponding to the cluster V, that contains all nodes in the network. By this definition the leaves of T_G correspond to sets $\{v\}, v \in V$, i.e., there is a one-to-one relation between the nodes of G and the leaf nodes of T_G.

We define the root to be on level 0 of T_G and all nodes whose parents are on level ℓ are defined to be on level $\ell+1$. It remains to define the bandwidths for the edges in E_t. Therefore, we first define the function $\text{cap} : 2^V \times 2^V \rightarrow \mathbb{R}^+$ which for two subsets $X, Y \subseteq V$ describes the total bandwidth that is available between nodes of X and nodes of Y. It is defined as follows:

$$\text{cap}(X,Y) := \sum_{x \in X, y \in Y} \mathsf{b}((x,y)) \ .$$

Recall that $\mathsf{b}((x,y))$ denotes the bandwidth for edge $(x,y) \in E$. For a set $X \subseteq V$ we denote the total bandwidth of edges leaving set X in G with $\text{out}(X) = \text{cap}(X, \overline{X})$, where $\overline{X} := V \setminus X$. Now, we define the bandwidth of an edge $(u_t, v_t) \in E_t$ connecting a level ℓ node v_t and a level $\ell-1$ node u_t. The bandwidth for such an edge is defined as $\mathsf{b}_t((u_t,v_t)) = \text{out}(S_{v_t})$. Figure 1 gives an example of a complete laminar system and the corresponding decomposition tree.

Now, we describe the parameters of the decomposition tree that are significant for the quality of the bi-simulation between G and T_G. The first important parameter is simply the height of T_G which will be denoted with $h(T_G)$. For specifying the further parameters we need the following notation. Let for a decomposition tree T_G and $\ell \in \{0, \ldots, h(T_G)\}$ the set $V_t^\ell \subset V_t$ denote the set of all level ℓ nodes of T_G. The clusters that correspond to nodes in V_t^ℓ form a sub-partition of V, i.e., a set of pairwise disjoint subsets of V. We define a weight function $\mathrm{w}_\ell : 2^V \rightarrow \mathbb{R}^+$ according to this sub-partition as follows:

$$\mathrm{w}_\ell(X) := \text{cap}(X,V) - \sum_{v_t \in V_t^\ell} \text{cap}(X \cap S_{v_t}, S_{v_t}) \ .$$

Informally speaking, the weight function $\mathrm{w}_\ell(X)$ counts for a subset X, the bandwidth of all edges that are adjacent to nodes in X and that do not connect nodes of the same cluster with respect to the sub-partition corresponding to V_t^ℓ. In order to define w_ℓ properly for all $\ell \in \{0, \ldots, h(T_G)+1\}$ we add the definition $\mathrm{w}_{h(T_G)+1}(x) := \text{cap}(X,V)$.

We mention some basic properties of w_ℓ that will be used intensively throughout the remainder of the paper. First of all w_ℓ is *additive*, i.e., for a set $X = X_1 \uplus X_2$, $\mathrm{w}_\ell(X) = \mathrm{w}_\ell(X_1) + \mathrm{w}_\ell(X_2)$. Furthermore, for a level ℓ cluster S_{v_t} we have $\mathrm{w}_\ell(S_{v_t}) = \text{out}(S_{v_t})$. Finally, $\mathrm{w}_{\ell-1}(X) \leq \mathrm{w}_\ell(X)$ holds for any $\ell \in \{1, \ldots, h(T_G)+1\}$.

Based on this weight function, we define for a level ℓ node v_t of the decomposition tree, the *bandwidth-ratio* λ_{v_t} as

$$\lambda_{v_t} := \max_{\substack{U \subset S_{v_t} \\ |U| \leq |S_{v_t}|/2}} \left\{ \frac{\text{out}(U)}{\text{cap}(U, S_{v_t} \setminus U)} \right\}$$

and the *weight-ratio* δ_{v_t} as

$$\delta_{v_t} := \max_{\substack{U \subset S_{v_t} \\ |U| \leq |S_{v_t}|/2}} \left\{ \frac{\mathrm{w}_{\ell+1}(U)}{\text{cap}(U, S_{v_t} \setminus U)} \right\} \ .$$

For these definitions we use the conventions $\frac{0}{0} = 0$ and $\frac{x}{0} = \infty$ for $x > 0$. Observe that thereby a tree node v_t corresponding to a cluster S_{v_t} that contains only a single node has $\lambda_{v_t} = 0$ and $\delta_{v_t} = 0$. Further, if the graph induced by the nodes of S_{v_t} is disconnected we have $\lambda_{v_t} = \infty$ and $\delta_{v_t} = \infty$.

These parameters will turn out to be a good measure for the congestion needed to route messages from the children of S_{v_t} to S_{v_t}. The intuition is as follows.

The bandwidth ratio λ_{v_t} describes the ratio between the capacity of edges that leave a set U (i.e., $\mathrm{out}(U)$) and the capacity of edges that connect U with the rest of the cluster (i.e., $\mathrm{cap}(U, S_{v_t} \setminus U)$). The simulation will mainly use edges of the latter type when leaving the set U. Hence, this ratio is an important parameter. The weight-ratio describes for a set U the ratio between $\mathrm{w}_{\ell+1}(U)$ and the capacity of edges connecting U with the rest of the cluster. In the simulation $\mathrm{w}_{\ell+1}(U)$ will be proportional to the probability that a message has to be routed to the set U. Therefore, the weight-ratio is an important parameter, as well.

Let $\delta(T_G) = \max_{v_t \in V_t}\{\delta_{v_t}\}$ and $\lambda(T_G) = \max_{v_t \in V_t}\{\lambda_{v_t}\}$ denote the maximum weight-ratio and maximum bandwidth-ratio, respectively, taken over all nodes of T_G. In the following section it is shown that the quality of the bi-simulation between a network G and an associated decomposition tree T_G can be expressed by the parameters $\delta(T_G)$, $\lambda(T_G)$ and $h(T_G)$. For convenience we will use the notation λ, δ, and h to denote $\lambda(T_G)$, $\delta(T_G)$ and $h(T_G)$, respectively, when there is no ambiguity.

3. Simulation results

In this section we show how the bi-simulation between G and T_G works and present the results on the congestion of the online routing strategy and the data management strategy based on this simulation.

3.1. Simulating G on T_G

The simulation of the network G on the decomposition tree T_G is straightforward. A node $v \in V$ is simulated by the leaf node in T_G that corresponds to cluster $\{v\}$. A message that is sent between nodes u and v in G is sent along the unique shortest path connecting the respective counterparts in T_G. The following theorem states that this simulation does not increase the congestion.

Theorem 1 *For any request sequence σ for an online problem on network G that can be processed with congestion C the straightforward simulation on T_G yields congestion $C_t \leq C$.*

Proof. Let $e_t = (u_t, v_t)$ denote an edge of T_G that connects a level ℓ node v_t to a level $\ell - 1$ node u_t and that has relative load C_t, when processing σ on T_G. Then, the absolute load of e_t is $C_t \cdot \mathrm{b}_t(e_t)$.

Now, consider how σ is processed on G. Any message that crosses edge e_t in T_G has either to leave or to enter the cluster S_{v_t}. The total bandwidth of all edges leaving cluster S_{v_t} is $\mathrm{out}(S_{v_t}) = \mathrm{b}_t(e_t)$. Thus, one of those edges must have relative load $\frac{C_t \cdot \mathrm{b}_t(e_t)}{\mathrm{out}(S_{v_t})} = C_t$. Hence, $C \geq C_t$. ■

3.2. Simulating T_G on G

In this section we show that the decomposition tree T_G can be simulated on the network G. In contrast to the above simulation of G on T_G this simulation is not completely independent of the definition of the online problem. Therefore, we split the description into two parts. In the first part we give a general, i.e., problem-independent, simulation result that relates the expected relative load of any edge of G to the congestion on the tree T_G. In the second part we show for the data management and the online routing problem that this simulation can be adopted such that the relative load of any edge e in G does not deviate too much from its expectation. This gives the desired simulation of T_G on G.

The problem-independent simulation is done as follows. A level ℓ node v_t of T_G is simulated by a random node of the corresponding cluster S_{v_t}, i.e., v_t is mapped to node $v \in S_{v_t}$ with probability $\frac{\mathrm{w}_{\ell+1}(v)}{\mathrm{w}_{\ell+1}(S_{v_t})}$. Note that by this definition a leaf node corresponding to a cluster $\{v\}$, $v \in V$, is simulated by v. It remains to describe the path selection strategy used for the routing between host nodes of G that simulate adjacent tree nodes of T_G.

Consider a level $\ell - 1$ node u_t of T_G with d children v_i, $i \in \{1, \ldots, d\}$. In an initialization phase we solve a concurrent multicommodity flow problem (CMCF-problem) on the cluster S_{u_t} and afterwards we select the routing paths for the tree edges (u_t, v_i) according to this solution.

The CMCF-problem is defined as follows. We define $|S_{u_t}|^2$ commodities $f_{u,v}$ for $u, v \in S_{u_t}$. The source of commodity $f_{u,v}$ is u, its sink is v and its demand is

$$d_{u,v} := \frac{\mathrm{w}_{\ell+1}(v)}{\mathrm{w}_{\ell+1}(S_{v_i})} \cdot \mathrm{out}(S_{v_i}) \cdot \frac{\mathrm{w}_{\ell}(u)}{\mathrm{w}_{\ell}(S_{u_t})} \quad , \tag{1}$$

where S_{v_i} denotes the level ℓ cluster that contains v. We solve the CMCF-problem on S_{u_t} while respecting the capacities, i.e., the bandwidth of the edges in S_{u_t}. Note that we restrict the solution in such a way that it only may use edges inside S_{u_t}, i.e., the flow is not allowed to leave the cluster. For a solution of the CMCF-problem define the *throughput fraction* q as the minimum over all commodities, of the fraction of the commodity's demand that is met. An optimal solution maximizes the value of q.

Altogether, after solving the CMCF-problem we have for each commodity $f_{u,v}$ a flow of size $q \cdot d_{u,v}$ from u to v. Furthermore, the total flow that traverses an edge e inside cluster S_{u_t} is smaller than the bandwidth $\mathrm{b}(e)$ of e.

We can now choose the routing path according to the solution of the CMCF-problem. Suppose that u_t is simulated by some node $u \in S_{u_t}$ and one of its children v_i is simulated by $v \in S_{v_i}$. Whenever a message is sent along edge (u_t, v_i) in the tree a path between u and v is chosen randomly in such a way that the expected load on any edge equals the flow of commodity $f_{u,v}$ along that edge.

The following theorem states that this simulation achieves a low expected load on any edge of G.

Theorem 2 *The expectation of the relative load $L(e)$ of an edge $e \in E$ due to the simulation of a tree strategy on G is bounded by*

$$E(L(e)) \leq O(\log n \cdot h \cdot \max\{\delta, \lambda\} \cdot C_t) \ ,$$

where C_t is the congestion on T_G. If G is planar or of constant genus this result improves to $E(L(e)) \leq O(h \cdot \max\{\delta, \lambda\} \cdot C_t)$.

Proof. Let q denote the minimum throughput fraction that was achieved when solving the multicommodity flow problems during the initialization phase. We first relate the expected load $E(L(e))$ of an edge e to the value of q and then we will give a lower bound on q in terms of the parameters $\delta(T_G)$ and $\lambda(T_G)$.

Lemma 3 *For any edge e of G, $E(L(e)) = O(h \cdot C_t/q)$.*

Proof. Let $L_\ell(e)$ denote the load of an edge e due to the simulation of edges connecting nodes on level ℓ to nodes on level $\ell - 1$ of T_G. We show that $E(L_\ell(e)) = O(C_t/q)$, which yields the lemma.

Fix a level ℓ. Let (u_t, v_t) denote an edge of T_G and assume that u_t is on level $\ell - 1$ and v_t is on level ℓ. The simulation of such a tree edge induces load on $e = (x, y)$ only if both endpoints x and y of e lie in the cluster S_{u_t}. This holds because the routing paths between the nodes simulating u_t and v_t do not leave the cluster S_{u_t}. Hence, let u_t denote a level $\ell - 1$ node such that $x, y \in S_{u_t}$. (If no such node exists $E(L_\ell(e)) = 0$.) Further, let d denote the degree of u_t and let v_i, $i \in \{1, \ldots, d\}$ denote the children of u_t.

We have to analyze the number of messages that traverse e due to the simulation of the tree edges (u_t, v_i), $i \in \{1, \ldots, d\}$. Assume for a worst case scenario that every edge (u_t, v_i) has relative load C_t. Then the absolute load of this edge is $C_t \cdot \mathrm{b}_t((u_t, v_i)) = C_t \cdot \mathrm{w}_\ell(S_{v_i})$. We say that this tree edge is mapped to a pair $(u, v) \in V \times V$ iff u_t is simulated by u and v_i is simulated by v. In this case $C_t \cdot \mathrm{w}_\ell(S_{v_i})$ messages have to be routed between u and v for the simulation of this edge. For simulating a tree node v_i the node v must be contained in the corresponding cluster S_{v_i}. Therefore, only one edge of $(u_t, v_i), i \in \{1, \ldots, d\}$ comes into question for being mapped to a fixed pair (u, v). Consequently, the expected number of messages that have to be routed between u and v for simulation of level ℓ edges is

$$\left(\frac{\mathrm{w}_{\ell+1}(v)}{\mathrm{w}_{\ell+1}(S_{v_i})}\right) \cdot \left(\frac{\mathrm{w}_\ell(u)}{\mathrm{w}_\ell(S_{u_t})}\right) \cdot C_t \cdot \mathrm{out}(S_{v_i}) \ ,$$

because $\mathrm{w}_{\ell+1}(v)/\mathrm{w}_{\ell+1}(S_{v_i})$ is the probability that v simulates v_i and $\mathrm{w}_\ell(u)/\mathrm{w}_\ell(S_{u_t})$ is the probability that u simulates u_t.

This number equals $C_t \cdot d_{u,v}$, where $d_{u,v}$ is the demand for commodity $f_{u,v}$ in the definition of the CMCF-problem (see Equation 1). This means that if at most $q \cdot d_{u,v}$ messages would be routed between any pair (u, v) the expected load on any edge e would be smaller than the capacity $\mathrm{b}(e)$, because the routing paths are chosen in such a way that the expected load of e equals the flow that passes e in the solution of the CMCF-problem.

Consequently, if we route $C_t \cdot d_{u,v}$ messages on expectation between any pair (u, v), then the expected (absolute) load is at most $\mathrm{b}(e) \cdot C_t/q$ for these messages on any edge e. Thus, $E(L_\ell(e)) = O(C_t/q)$ holds for any edge e. This yields the lemma. ■

Now, we derive a bound on the throughput fraction of the multicommodity flow problems that are solved during the initialization phase. Let u_t denote a level $\ell - 1$ node of T_G with d children v_i, $i \in \{1, \ldots, d\}$. We show a bound for the throughput fraction of the flow problem on cluster S_{u_t}.

A cut in S_{u_t} is a partition of S_{u_t} into two disjoint sets U and $S_{u_t} \setminus U$. The *capacity* of the cut is $\mathrm{cap}(U, S_{u_t} \setminus U)$, i.e., the bandwidth of edges between both sets. The *sparsity* of the cut is its capacity divided by the total demand that has to *cross* the cut, i.e., the sum of the demands of commodities for which source and destination lie in different portions of the cut. A cut with minimum sparsity is called sparsest cut. It is widely known that the value of a sparsest cut, i.e., its sparsity, and the maximum throughput fraction are strongly related (see e.g. [9]). Now, we first give a lower bound on the value of a sparsest cut in S_{u_t}, and then we will utilize this relationship to derive a lower bound on the maximum throughput fraction of the CMCF-problem.

Lemma 4 *The value of the sparsest cut of the CMCF-problem on cluster S_{u_t} is at least $\frac{1}{5\delta + 2\lambda}$.*

Proof. Fix a non-empty subset $X \subset S_{u_t}$ and let $Y := S_{u_t} \setminus X$ denote the other part of S_{u_t}. Further let $X_i := X \cap S_{v_i}$ and $Y_i := Y \cap S_{v_i}$ denote the part of X and Y, respectively, that lies in cluster S_{v_i}. The sparsity $\phi(X, Y)$ of the cut induced by X and Y is defined as $\phi(X, Y) = \frac{\mathrm{cap}(X, Y)}{\mathrm{dem}(X, Y)}$, where $\mathrm{dem}(X, Y)$ is the demand that has to cross the cut, i.e., the sum of all demands of commodities for which source and destination lie in different portions of the cut.

Let $\mathrm{abs}(U)$ for a subset $U \subset S_{u_t}$ denote the total demand that is *absorbed* in subset U (i.e., the total demand of commodities that have their sink in U). The following technical claim is proved in a full version of the paper.

Claim 5 *For any subset U_i of a level ℓ cluster S_i, $\mathrm{abs}(U_i) \leq 2 \cdot \mathrm{w}_\ell(U_i) + (\delta + 2\lambda) \cdot \mathrm{cap}(U_i, S_i \setminus U_i)$.*

This claim can be used to estimate $\mathrm{dem}(X,Y)$ as follows:

$$\begin{aligned}\mathrm{dem}(X,Y) &= \sum_i \mathrm{abs}(X_i)\cdot\frac{\mathrm{w}_\ell(Y)}{\mathrm{w}_\ell(S_{u_t})} + \sum_i \mathrm{abs}(Y_i)\cdot\frac{\mathrm{w}_\ell(X)}{\mathrm{w}_\ell(S_{u_t})}\\ &\le 4\cdot\frac{\mathrm{w}_\ell(X)\cdot\mathrm{w}_\ell(Y)}{\mathrm{w}_\ell(S_{u_t})} + (\delta+2\lambda)\cdot\mathrm{cap}(X,Y)\ .\end{aligned}$$

For this step we utilized $\sum_i \mathrm{cap}(X_i,Y_i) \le \mathrm{cap}(X,Y)$ and $\sum_i \mathrm{w}_\ell(X_i) = \mathrm{w}_\ell(X)$.

One of the sets X and Y contains at most $|S_{u_t}|/2$ nodes. W.l.o.g. we assume $|X| \le |S_{u_t}|/2$. By combining the definition of the weight ratio δ with the above inequality we get

$$\begin{aligned}\mathrm{dem}(X,Y) &\le 4\delta\cdot\mathrm{cap}(X,Y) + (\delta+2\lambda)\cdot\mathrm{cap}(X,Y)\\ &\le (5\delta+2\lambda)\cdot\mathrm{cap}(X,Y)\ ,\end{aligned}$$

which yields the lemma. ■

In [3] it is shown that any CMCF-problem can be satisfied up to $q = \Omega(\phi/\log k)$, where ϕ is the value of the sparsest cut and k is the number of commodities. Combining this with Lemma 3 and 4 yields $E(L(e)) \le O(\log n\cdot h\cdot \max\{\delta,\lambda\}\cdot C_t)$.

For the improved result we utilize that a uniform multicommodity flow on graphs of constant genus can be satisfied up to a factor $q = \Omega(\phi)$ (see [8]). In order to apply this result we transform the CMFC-problem into a uniform multicommodity flow problem with nearly the same value of a sparsest cut. Then Lemma 3 and 4 yield the theorem. The details are omitted due to space limitations. ■

So far, we have shown that T_G can be simulated on G such that for every edge of the network the expected load remains small. In order to obtain a result on the congestion of the simulation we have to show that this simulation technique can be adopted such that the load of any edge does not deviate too much from its expectation. This is done in the proofs of the following theorems.

Theorem 6 *Given a graph G and an associated decomposition tree T_G there exists an oblivious online routing algorithm that is (strictly) $O(\log n\cdot h\cdot\max\{\delta,\lambda\})$-competitive with respect to the congestion.*

Remark 7 *By utilizing the results of Section 4 on the parameters of T_G this theorem gives an $O(\log^3 n)$-competitive online routing algorithm for general networks.*

Proof. In order to apply the framework as sketched in Section 1.3 we need an efficient solution for the online routing problem on trees. This point is easy since in a tree the routing path between two nodes is unique.

The simulation technique can be adopted as follows. In the online routing problem the internal nodes of the decomposition tree do not store any information. They are only used during the simulation to ensure that appropriate routing paths are selected. Therefore, for each routing request a new random embedding of these nodes in the network G can be used. Let $L_i(e)$ denote a 0-1 random variable describing the load on edge $e\in E$ due to the routing of the *i-th* request σ_i. (The routing algorithm can easily ensure that each edge is traversed at most once for each request. Hence, we can assume that $L_i(e)$ is a 0-1 random variable.)

The variables $L_i(e)$ are independent and hence, the load $L(e)$ of an edge e, is a sum of independent random variables. By applying a Chernoff bound (see e.g. [6]) to this sum we get $L(e) = O(E(L(e)))$, w.h.p. As this holds for any edge e we get the desired result on the congestion. Note that the independence of different routing requests means that the routing algorithm is oblivious as stated in the theorem. ■

Similarly, one can prove the following theorem for the data management problem. The proof will appear in a full version of the paper.

Theorem 8 *There is an $O(\log^3 n)$-competitive online algorithm for the data management problem on general networks.*

The above results are obtained by utilizing the results of Section 4 which hold for general networks. For special networks these results can be significantly improved. For example if G is a mesh then there is a decomposition tree T_G with $\lambda(T_G) = O(1)$, $\delta(T_G) = O(1)$ and $h(T_G) = O(\log n)$. This yields a data management strategy and a routing strategy with competitive ratio $O(\log n)$. Hence, our framework achieves the same results for the mesh as shown in [11].

4. The graph decomposition

In this section we show that for any graph G there is a decomposition tree T_G with good parameters $\lambda(T_G)$, $\delta(T_G)$ and $h(T_G)$. This completes the proof of the competitive ratios for the data management strategy and the online routing strategy presented in Section 3. The main theorem is as follows.

Theorem 9 *For any graph $G = (V,E)$ with $n = |V|$ nodes there exists a decomposition tree T_G that has height $h(T_G) = O(\log n)$, maximum bandwidth-ratio $\lambda(T_G) = O(\log n)$ and maximum weight-ratio $\delta(T_G) = O(\log n)$.*

Proof. For the following proof we define $\lambda := 20\cdot\log n + 1$ and $\delta := 24\cdot\lambda$. We construct a decomposition tree T_G with $\lambda(T_G)\le\lambda$ and $\delta(T_G)\le\delta$. We say that a subset $S\subset V$ fulfills the bandwidth-property and the weight-property iff

$$\lambda \ge \max_{\substack{U\subset S\\ |U|\le\frac{3}{4}|S|}}\left\{\frac{\mathrm{out}(U)}{\mathrm{cap}(U,S\setminus U)}\right\} \quad\text{and}\quad \delta \ge \max_{\substack{U\subset S\\ |U|\le|S|/2}}\left\{\frac{\mathrm{w}_{\ell+1}(U)}{\mathrm{cap}(U,S\setminus U)}\right\},$$

respectively. Obviously, a tree T_G in which any cluster S_{v_t} fulfills both properties has bandwidth-ratio at most λ

```
BandwidthPartition(R)
  𝒫_R := {R}
  while ∃R_i ∈ 𝒫_R not fulfilling the bandwidth-property do
    find U ⊂ R_i with |U| ≤ (3/4)|R_i|
    and λ · cap(U, R_i \ U) < out(U)
    𝒫_R := 𝒫_R \ {R_i}
    𝒫_R := 𝒫_R ∪ {U, R_i \ U}
  end
  return 𝒫_R
```

Figure 2. The algorithm BandwidthPartition

```
Partition(S)
  compute (1/6)-balanced mincut (A, B) of S
  𝒫_S := {{v} | v ∈ S}
  while ∃U violating the sharpened weight condition do
    /* i.e. |U| ≤ (2/3)|S| and ∃Q ⊂ 𝒫_S with U = ⊎_{S_i∈Q} S_i */
    for each S_i ∈ Q do 𝒫_S := 𝒫_S \ {S_i}
    𝒫_S := 𝒫_S ∪ BandwidthPartition(U ∩ A)
    𝒫_S := 𝒫_S ∪ BandwidthPartition(U ∩ B)
  end
  return 𝒫_S
```

Figure 3. The algorithm Partition.

and weight-ratio at most δ. Note that the definition of the bandwidth-property slightly differs from that of the bandwidth-ratio as the maximum is taken over all sets that contain at most $\frac{3}{4}|S|$ nodes. This will be needed later in the proof.

We prove the existence of a tree T_G with good parameters by presenting an algorithm for constructing the tree. The running time of this algorithm is not polynomial as it solves several NP-complete problems during the construction. Observe, however, that the decomposition has to be computed only once for a network and can then be used by any online algorithm that aims to minimize the congestion.

A key observation for the construction is that the above properties are somehow local to a cluster. This means that, e.g., whether a cluster fulfills the bandwidth-property or not, only depends on that cluster but not on the rest of the decomposition tree. Similarly, the weight-property for a level ℓ cluster S_{v_t} only depends on this cluster and on clusters corresponding to children of v_t in T_G. This holds since these child-clusters completely define the weight function $w_{\ell+1}$ for subsets of S_{v_t}. Let v_i, $i \in I$ denote the children of v_t. Then for a subset $U \subset S_{v_t}$ we can write $w_{\ell+1}(U) = \sum_{i\in I} \mathrm{cap}(U \cap S_{v_i}, \overline{S_{v_i}})$, which means that $w_{\ell+1}(U)$ and hence, the weight-ratio of S_{v_t}, only depends on the "child-clusters" S_{v_i}. Because of the observed locality of the desired properties we can conclude the theorem from the following lemma.

Lemma 10 *Given a level ℓ cluster $S \subset V$ that fulfills the bandwidth property, it is possible to partition S into sub-clusters S_i with the following characteristics.*

1. *The sets S_i are disjoint and form a partitioning of S, i.e., $\biguplus_i S_i = S$.*
2. *Each sub-cluster S_i fulfills the bandwidth-property.*
3. *For each sub-cluster S_i we have $|S_i| \le \frac{5}{6} \cdot |S|$.*
4. *The cluster S fulfills the weight-property, i.e., for each subset $U \subset S$ with $|U| \le |S|/2$ we have $\sum_i \mathrm{cap}(U \cap S_i, \overline{S_i}) \le \delta \cdot \mathrm{cap}(U, S \setminus U)$.*

Now, we first argue that the above lemma yields the theorem. Obviously, the cluster V that contains all nodes in the network fulfills the bandwidth-property, because $\mathrm{out}(V) = 0$. This is the set corresponding to the root of T_G. By applying the above lemma to V we get a partitioning of V. The sets of this partitioning correspond to the level 1 nodes of T_G. Since these sets again fulfill the bandwidth-property we can apply the lemma recursively until the leaf nodes of T_G correspond to singleton sets $\{v\}, v \in V$.

It is easy to verify that by this construction every cluster of T_G fulfills both properties. Further, the height of T_G is logarithmic because of Property 3 of the lemma. This yields the theorem.

Proof of Lemma 10. The algorithm for partitioning S according to the requirements of Lemma 10 uses a subroutine that is described in the proof of the following lemma.

Lemma 11 *For any subset $R \subset V$ it is possible to partition R into disjoint subsets $R_i \subset R$, such that each R_i fulfills the bandwidth-property and $\sum_i \mathrm{out}(R_i) \le 2 \cdot \mathrm{out}(R)$.*

Proof. The following straightforward algorithm computes a partitioning $\mathcal{P}_R$ of R with the above characteristics. In the beginning $\mathcal{P}_R$ contains only the set R. As long as $\mathcal{P}_R$ contains a set R_i that does not fulfill the bandwidth-property this set is removed from $\mathcal{P}_R$ and replaced by two subsets U and $R_i \setminus U$ of R_i that are chosen as follows. U is selected as an arbitrary subset that violates the bandwidth-property, i.e., $|U| \le \frac{3}{4}|R_i|$ and $\lambda \cdot \mathrm{cap}(U, R_i \setminus U) < \mathrm{out}(U)$. An outline of the algorithm is given in Figure 2.

Obviously, after the algorithm has finished the set system $\mathcal{P}_R$ contains only sets R_i that fulfill the bandwidth-property and these sets partition R, i.e., $\biguplus_i R_i = R$. It remains to show that $\sum_i \mathrm{out}(R_i) \le 2 \cdot \mathrm{out}(R)$.

For the analysis we introduce the following notion. Consider a set R_i that is partitioned into two sets U and $R_i \setminus U$ during the algorithm. By the partitioning the term $\sum_i \mathrm{out}(R_i)$ increases by $2 \cdot \mathrm{cap}(U, R_i \setminus U)$. For such a step of the algorithm we say that the algorithm "*buys*" the new bandwidth $2 \cdot \mathrm{cap}(U, R_i \setminus U)$ and that it "*pays*" with $\mathrm{cap}(U, \overline{R_i})$, i.e., the bandwidth of edges leaving U that do not contribute to $\mathrm{cap}(U, R_i \setminus U)$.

Now, we derive an upper bound on the amount of new bandwidth the algorithm can buy. For this we need the following two claims.

Claim 12 *If in a step the algorithm buys bandwidth B the price for this bandwidth (i.e., $\mathrm{cap}(U, \overline{R_i})$) is at least $\frac{\lambda-1}{2} \cdot B$.*

Proof. U is selected as a set that violates the bandwidth property. Therefore, $\lambda \cdot \mathrm{cap}(U, R_i \setminus U) < \mathrm{out}(U) = \mathrm{cap}(U, \overline{R_i}) + \mathrm{cap}(U, R_i \setminus U)$. The new bandwidth B equals $2 \cdot \mathrm{cap}(U, R_i \setminus U)$. This yields $\frac{2}{\lambda-1} \cdot \mathrm{cap}(U, \overline{R_i}) > B$. ∎

Claim 13 *The bandwidth of an edge e can be used at most $5 \cdot \log(n)$ times for buying new bandwidth.*

Proof. Let $e = (u, v)$ denote an edge of G. Suppose that e is used for buying new bandwidth when the algorithm partitions a set R_i into sets U and $R_i \setminus U$ with $|U| \leq \frac{3}{4}|R_i|$. Either u or v must be contained in the set U because otherwise e would not contribute to $\mathrm{cap}(U, \overline{R_i})$, i.e., the price for the new bandwidth. The set U that now contains one end-point of e has many nodes less than R_i. This observation can be utilized as follows.

Let for the running time of the algorithm R_u and R_v denote the set in the partition $\mathcal{P}_R$ that contain the node u and v, respectively. Whenever the algorithm uses e for buying new bandwidth either the set R_u or the set R_v changes, and thereby $|R_u|$ or $|R_v|$ is reduced by a factor of $\frac{3}{4}$. Obviously, this can happen only $2 \cdot \log_{4/3}(n) \leq 5 \cdot \log n$ times. ∎

We get an upper bound on the amount of new bandwidth as follows. In the beginning when $\mathcal{P}_R$ contains only the set R the only bandwidth that can be used for buying new bandwidth is the bandwidth of edges leaving cluster R, i.e., $\mathrm{out}(R)$. We denote the value of this bandwidth with B_0. Using the above observations we can estimate the bandwidth B_1 the algorithm can buy for the bandwidth B_0. This is at most

$$B_1 \leq 5 \log(n) \cdot \frac{2}{\lambda - 1} B_0 = \frac{B_0}{2} \ .$$

For this bandwidth B_1 the algorithm can buy again some new bandwidth B_2 which can be shown to be smaller than $\frac{B_1}{2}$. Proceeding in this manner we get that the total new bandwidth is at most $\sum_{i=1}^{\infty} B_i \leq B_0$.

Altogether, $\sum_i \mathrm{out}(R_i)$ increases at most by B_0 during the algorithm. Hence, in the end we have $\sum_i \mathrm{out}(R_i) \leq 2 \cdot \mathrm{out}(R)$ which yields the lemma. ∎

Now, we describe the algorithm for partitioning S according to the requirements of Lemma 10. The general idea of the algorithm is to maintain a partitioning $\mathcal{P}_S$ of the set S that fulfills requirements 1, 2 and 3 of the lemma and then to change this partitioning successively, until Requirement 4 is fulfilled, as well. It is relatively easy to maintain Requirement 1 that $\mathcal{P}_S$ is a partitioning, i.e., $\biguplus_{S_i \in \mathcal{P}_S} S_i = S$.

For maintaining Requirement 2 the algorithm uses the algorithm BANDWIDTHPARTITION as a subroutine. Before a new set is added to $\mathcal{P}_S$ this set is decomposed by BANDWIDTHPARTITION into subsets that fulfill the bandwidth-property. Then these subsets are added to $\mathcal{P}_S$, instead.

Requirement 3 is fulfilled as follows. In the beginning the algorithm calculates a $\frac{1}{6}$-balanced mincut for the subgraph induced by S. This means S is partitioned into two subsets A and $B := S \setminus A$, such that the capacity $\mathrm{cap}(A, B)$ between both sets is minimized but each set contains at least $|S|/6$ nodes. Then the algorithm guarantees that each set S_i in the partitioning is either a subset of A or a subset of B. Hence, Requirement 3 holds.

In order to fulfill the 4-*th* requirement the algorithm has to fulfill the condition $\sum_i \mathrm{cap}(U \cap S_i, \overline{S_i}) \leq \delta \cdot \mathrm{out}(U)$ for all subsets U of S with at most $|S|/2$ nodes. For doing this the algorithm uses the following trick. It tries to ensure a *sharpened* condition that only must hold for certain subsets U which are composed from sub-clusters of the current partitioning, i.e., U can be written as $U = \biguplus_{S_i \in \mathcal{Q}} S_i$ with $\mathcal{Q} \subset \mathcal{P}_S$. Formally, we say a subset $U \subset S$ violates the *sharpened weight condition* iff

$$\sum_i \mathrm{cap}(U \cap S_i, \overline{S_i}) > 6 \cdot \mathrm{out}(U) \ ,$$

and $|U| \leq \frac{2}{3}|S|$ and $\exists \mathcal{Q} \subset \mathcal{P}_s$ such that $U = \biguplus_{S_i \in \mathcal{Q}} S_i$. Later we will show that if no subset violates this condition then the cluster S fulfills the weight-property, i.e., Requirement 4 holds.

The partitioning algorithm works as follows. In the beginning $\mathcal{P}_S = \{\{v\} | v \in S\}$ contains all singleton subsets of v. As long as there exists a subset $U = \biguplus_{S_i \in \mathcal{Q}} S_i$ that violates the sharpened weight condition the algorithm does the following. Firstly, it removes all sub-clusters $S_i \in \mathcal{Q}$ from $\mathcal{P}_S$. Then it does not directly insert U into $\mathcal{P}_S$ but first it ensures that requirements 2 and 3 will be fulfilled afterwards. Therefore, U is partitioned into $U \cap A$ and $U \cap B$, and then these sets are again partitioned with BANDWIDTHPARTITION. Finally, all subsets resulting from this partitioning are inserted into $\mathcal{P}_S$. A description of the algorithm is shown in Figure 3.

It is clear from the above description that after the algorithm has finished, requirements 1, 2, and 3 are fulfilled and furthermore, no set violates the sharpened weight condition. It remains to show that the algorithm fulfills Requirement 4, as well, and last but not least we have to show that the algorithm terminates.

Lemma 14 *The algorithm* PARTITION *terminates.*

Proof. We show that in each iteration of the while loop, the term $\sum_{S_i \in \mathcal{P}_S} \mathrm{out}(S_i)$ decreases. In each iteration the algorithm first removes all sets $S_i \in \mathcal{Q}$ from $\mathcal{P}_S$. Hence, $\sum_{S_i \in \mathcal{P}_S} \mathrm{out}(S_i)$ decreases by $\sum_{S_i \in \mathcal{Q}} \mathrm{out}(S_i) = \sum_{S_i \in \mathcal{Q}} \mathrm{cap}(U \cap S_i, \overline{S_i}) > 6 \cdot \mathrm{out}(U)$.

Now, we show that the increment of $\sum_{S_i \in \mathcal{P}_S} \mathrm{out}(S_i)$, when inserting the sets resulting from the partitioning of U, is smaller than this value. Therefore, we need the following claim which is proved in the full version.

Claim 15 *For a subset U with $|U| \leq \frac{2}{3}|S|$,* $\mathrm{cap}(A \cap U, B \cap U) \leq \mathrm{out}(U)$.

From this claim we get

$$\begin{aligned}\mathrm{out}(A \cap U) + \mathrm{out}(B \cap U) \\ &\leq \mathrm{cap}(A \cap U, \overline{U}) + 2 \cdot \mathrm{cap}(A \cap U, B \cap U) \\ &\quad + \mathrm{cap}(B \cap U, \overline{U}) \leq 3\,\mathrm{out}(U) \quad .\end{aligned}$$

Hence, when partitioning the sets $A \cap U$ and $B \cap U$ with BANDWIDTHPARTITION we get a collection of subsets U_i with $\sum_i \mathrm{out}(U_i) \leq 6 \cdot \mathrm{out}(U)$, according to Lemma 11.

Consequently, when inserting these sets into $\mathcal{P}_S$ the value of $\sum_{S_i \in \mathcal{P}_S} \mathrm{out}(S_i)$ increases by less than $6 \cdot \mathrm{out}(U)$ and therefore it decreases during each iteration of the while loop. This yields the lemma. ■

Lemma 16 *If no subset of S violates the sharpened weight condition then S fulfills the weight-property.*

Proof. Fix an arbitrary nonempty subset $X \subset S$ with $|X| \leq |S|/2$ and let $Y := S \setminus X$ denote the other part of S. Further let $X_i := X \cap S_i$ and $Y_i := Y \cap S_i$ denote the part of X and Y, respectively, that lies in cluster $S_i \in \mathcal{P}_S$. We have to show that $\frac{\sum_i \mathrm{cap}(X_i, \overline{S_i})}{\mathrm{cap}(X,Y)} \leq \delta$.

We partition the sets X_i into so-called *small* and *large* sets, as follows. A set X_i is called *large* iff $|X_i| \leq \frac{3}{4}|S_i|$ and *small*, otherwise. Let $\mathcal{S}$ and $\mathcal{L}$ denote the set of small and large sets, respectively, and let I_s and I_l denote the corresponding index sets, i.e., $\mathcal{S} = \{X_i \mid i \in I_s\}$ and $\mathcal{L} = \{X_i \mid i \in I_l\}$. Further, we introduce a set $U := \biguplus_{i \in I_l} S_i$, which is the union of all sub-clusters of the current partitioning that have a *large* intersection with X. Note that the definition of *small* and *large* and the fact that $|X| \leq |S|/2$ ensures that $|U| \leq \frac{2}{3}|S|$. The idea of the proof is to find a relation between $\mathrm{cap}(X,Y)$ and $\mathrm{out}(U)$ and then to exploit that U must fulfill the sharpened weight-property. The following claim is proved in a full version of the paper.

Claim 17 $\mathrm{cap}(X,Y) \geq \frac{1}{4\lambda}\left(\mathrm{out}(U) + \sum_{i \in I_s} \mathrm{out}(X_i)\right)$.

Using this claim, the quotient $\sum_i \mathrm{cap}(X_i, \overline{S_i}) / \mathrm{cap}(X,Y)$ can be estimated, as follows.

$$\begin{aligned}\frac{\sum_i \mathrm{cap}(X_i, \overline{S_i})}{\mathrm{cap}(X,Y)} &= \frac{\sum_{i \in I_s} \mathrm{cap}(X_i, \overline{S_i}) + \sum_{i \in I_l} \mathrm{cap}(X_i, \overline{S_i})}{\mathrm{cap}(X,Y)} \\ &\leq 4\lambda \cdot \frac{\sum_{i \in I_s} \mathrm{out}(X_i) + \sum_{i \in I_l} \mathrm{cap}(S_i, \overline{S_i})}{\sum_{i \in I_s} \mathrm{out}(X_i) + \mathrm{out}(U)} \\ &\leq 4\lambda \cdot \frac{\sum_{i \in I_l} \mathrm{cap}(S_i, \overline{S_i})}{\mathrm{out}(U)}\end{aligned}$$

For the last inequality we utilized $\sum_{i \in I_l} \mathrm{cap}(S_i, \overline{S_i}) \geq \mathrm{out}(U)$.

Since, U fulfills the sharpened weight-property we have $\sum_{i \in I_l} \mathrm{cap}(S_i, \overline{S_i}) \leq 6 \cdot \mathrm{out}(U)$. Plugging this into the above inequality yields $\frac{\sum_i \mathrm{cap}(X_i, \overline{S_i})}{\mathrm{cap}(X,Y)} \leq 24\lambda \leq \delta$. This means that the cluster S fulfills the weight-property. ■

This completes the proof of Lemma 10. As previously claimed Lemma 10 directly implies the theorem. ■

Acknowledgments

I am very grateful to Baruch Awerbuch and Bruce Maggs for their helpful comments on a preliminary version of this paper.

References

[1] W. Aiello, F. T. Leighton, B. M. Maggs, and M. Newman. Fast algorithms for bit-serial routing on a hypercube. *Mathematical Systems Theory*, 24(4):253–271, 1991.

[2] J. Aspens, Y. Azar, A. Fiat, S. A. Plotkin, and O. Waarts. Online load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*, pages 623–631, 1993.

[3] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate mincut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998.

[4] B. Awerbuch and Y. Azar. Local optimization of global objectives: Competitive distributed deadlock resolution and resource allocation. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 240–249, 1994.

[5] A. Borodin and J. E. Hopcroft. Routing, merging and sorting on parallel models of computation. *Journal of Computer and System Sciences*, 30(1):130–145, 1985.

[6] T. Hagerup and C. Rüb. A guided tour of Chernoff bounds. *Information Processing Letters*, 33(6):305–308, 1990.

[7] C. Kaklamanis, D. Krizanc, and T. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Proceedings of the 2nd ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 31–36, 1990.

[8] P. N. Klein, S. A. Plotkin, and S. B. Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*, pages 682–690, 1993.

[9] F. T. Leighton and S. B. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 422–431, 1988.

[10] S. Leonardi. On-line network routing. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *LNCS*, pages 242–267. Springer, 1998.

[11] B. M. Maggs, F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Exploiting locality for networks of limited bandwidth. In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 284–293, 1997.

[12] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th ACM Symposium on Theory of Computing (STOC)*, pages 263–277, 1981.
[13] B. Vöcking. *Static and Dynamic Data Management in Networks*. PhD thesis, Universität Paderborn, Dec. 1998.
[14] B. Vöcking. Almost optimal permutation routing on hypercubes. In *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC)*, pages 530–539, 2001.

A. Appendix

A.1 Proof of Claim 5

By definition of the multicommodity flow problem we have

$$\mathrm{abs}(U_i) = \frac{\mathrm{w}_{\ell+1}(U_i)}{\mathrm{w}_{\ell+1}(S_i)}\,\mathrm{w}_\ell(S_i) \quad .$$

In order to estimate $\mathrm{abs}(U_i)$ we distinguish two cases according to the cardinality of U_i.

- Suppose $|U_i| \le |S_i/2|$.

  By applying the definition of the weight-ratio we get

  $$\mathrm{abs}(U_i) \le \mathrm{w}_{\ell+1}(U_i) \le \delta \cdot \mathrm{cap}(U_i, S_i \setminus U_i) \quad .$$

- Now, suppose $|U_i| \ge |S_i/2|$.

  In this case we distinguish two sub-cases according to the value of $\mathrm{w}_\ell(U_i)$.

  – Suppose $\mathrm{w}_\ell(U_i) \ge \mathrm{w}_\ell(S_i)/2$. Then

  $$\mathrm{abs}(U_i) \le \mathrm{w}_\ell(S_i) \le 2 \cdot w_\ell(U_i) \quad .$$

  – Suppose $\mathrm{w}_\ell(S_i \setminus U_i) \ge \mathrm{w}_\ell(S_i)/2$. Then we get

  $$\begin{aligned}\mathrm{abs}(U_i) &\le \mathrm{w}_\ell(S_i) \le 2 \cdot w_\ell(S_i \setminus U_i) \\ &\le 2 \cdot \mathrm{out}(S_i \setminus U_i) \le 2\lambda \cdot \mathrm{cap}(U_i, S_i \setminus U_i),\end{aligned}$$

  by applying the definition of the bandwidth-ratio for the set $S_i \setminus U_i$.

Altogether, this yields the claim.

A.2 Proof of Claim 15

Proof. Assume for contradiction that $\mathrm{cap}(A \cap U, B \cap U) > \mathrm{out}(U)$. Then,

$$\begin{aligned}\mathrm{cap}(A,B) &\ge \mathrm{cap}(A \setminus U, B \setminus U) + \mathrm{cap}(A \cap U, B \cap U) \\ &> \mathrm{cap}(A \setminus U, B \setminus U) + \mathrm{out}(U) \\ &\ge \mathrm{cap}(A \setminus U, B \setminus U) + \mathrm{cap}(U, B \setminus U) \\ &= \mathrm{cap}(A \cup U, B \setminus U) \\ &= \mathrm{cap}(A \cup U, S \setminus (A \cup U)) \quad .\end{aligned}$$

Analogously, it can be shown that $\mathrm{cap}(A,B) > \mathrm{cap}(B \cup U, S \setminus (B \cup U))$. This means that the cuts induced by $A \cup U$ and $B \cup U$ have smaller capacity than $\mathrm{cap}(A,B)$. Since $A|B$ is the minimum $\frac{1}{6}$-balanced cut we conclude that $A \setminus U$ and $B \setminus U$ both contain less than $|S|/6$ nodes. But then U must contain more than $\frac{2}{3}|S|$ nodes. This is a contradiction. ■

A.3 Proof of Claim 17

The set X fulfills the bandwidth-property since $|X| \le |S|/2$. Hence,

$$\lambda \cdot \mathrm{cap}(X,Y) \ge \mathrm{out}(X) \ge \mathrm{cap}(U \cap X, V \setminus S) \quad . \tag{2}$$

Similarly, we can apply the bandwidth-property for sets X_i, $i \in I_s$ and Y_i, $i \in I_l$. This yields

$$\begin{aligned}\mathrm{cap}(X,Y) &\ge \sum_{i \in I_s} \mathrm{cap}(X_i,Y_i) + \sum_{i \in I_l} \mathrm{cap}(X_i,Y_i) \\ &\ge \frac{1}{\lambda}\sum_{i \in I_s} \mathrm{out}(X_i) + \frac{1}{\lambda}\sum_{i \in I_l} \mathrm{out}(Y_i) \\ &\ge \frac{1}{\lambda}\sum_{i \in I_s} \mathrm{out}(X_i) + \frac{1}{\lambda}\sum_{i \in I_l} \mathrm{cap}(Y_i, V \setminus S) \\ &= \frac{1}{\lambda}\sum_{i \in I_s} \mathrm{out}(X_i) + \frac{1}{\lambda}\mathrm{cap}(U \setminus X, V \setminus S) \quad .\end{aligned}$$

By combining this result with Inequality 2 we get

$$2\lambda \cdot \mathrm{cap}(X,Y) \ge \mathrm{cap}(U, V \setminus S) + \sum_{i \in I_s} \mathrm{out}(X_i) \quad . \tag{3}$$

Now, we relate $\mathrm{cap}(X,Y)$ to $\mathrm{cap}(U, S \setminus U)$.

$$\begin{aligned}&\mathrm{cap}(U, S \setminus U) \\ &\quad = \mathrm{cap}(Y \cap U, S \setminus U) + \mathrm{cap}(X \cap U, X \cap (S \setminus U)) \\ &\qquad + \mathrm{cap}(X \cap U, Y \cap (S \setminus U)) \\ &\quad = \mathrm{cap}(\uplus_{i \in I_l} Y_i, S \setminus U) + \mathrm{cap}(X \cap U, \uplus_{i \in I_s} X_i) \\ &\qquad + \mathrm{cap}(X \cap U, Y \cap (S \setminus U)) \\ &\quad \le \sum_{i \in I_l} \mathrm{out}(Y_i) + \sum_{i \in I_s} \mathrm{out}(X_i) + \mathrm{cap}(X \cap U, Y) \\ &\quad \le \lambda \sum_{i \in I_l} \mathrm{cap}(X_i,Y_i) + \lambda \sum_{i \in I_s} \mathrm{cap}(X_i,Y_i) + \mathrm{cap}(X,Y) \\ &\quad \le (\lambda+1) \cdot \mathrm{cap}(X,Y) \\ &\quad \le 2\lambda \cdot \mathrm{cap}(X,Y)\end{aligned}$$

Summing the above inequality and Inequality 3 gives

$$\begin{aligned}4\lambda \cdot \mathrm{cap}(X,Y) &\ge \mathrm{cap}(U, S \setminus U) + \mathrm{cap}(U, V \setminus S) \\ &\quad + \sum_{i \in I_s} \mathrm{out}(X_i) \\ &= \mathrm{out}(U) + \sum_{i \in I_s} \mathrm{out}(X_i) \quad .\end{aligned}$$

This yields the claim.