Data Management and Routing in General Networks

Dissertation of Harald Räcke

Contents

CHAPTER 1

Introduction

In recent years, heterogeneous distributed systems have more and more displaced traditional parallel computers as the systems of choice for so-called grand challenge applications, like, e.g., weather prediction, protein folding, or cell modeling. In contrast to conventional parallel systems, with their specialized processors and custom interconnection networks, these novel architectures are characterized by connecting a (usually) huge amount of standard workstations or clusters, via a relatively unstructured communication network as, e.g., the Internet.

While this approach, in principle, enables high performance computing at comparatively low cost, it poses new challenges for the design of algorithms, as these have to achieve a high scalability even in irregular networks. In fact the main goal of the currently rapidly evolving areas of Grid computing and Peer to Peer computing is the development of basic services for such systems.

Basic services like routing, load balancing, data management, or monitoring are essential for an efficient use of parallel computers as the user, i.e., the programmer of a parallel application, cannot cope with all details and difficulties of particular parallel machines.

This thesis provides a theoretical analysis of algorithms for routing and data management services in large distributed systems. Our algorithms are close to optimal for any given topology, even for very irregular ones, which means that they are likely to perform well in modern parallel systems.

A *routing service* is very fundamental and provides the basic functionality for exchanging information between nodes of the network. We consider *unicast* routing algorithms (in which information is sent from one source to one destination), as well as *multicast* routing algorithms (in which information is sent from one source to possibly several destinations).

A *data management service* provides access to shared data object that can be read and written by nodes in the system. Examples for shared data objects are global variables in a parallel program, files in a distributed files system, or pages in the WWW. Compared to routing, a data management service is much more sophisticated. For example a data management algorithm has to deal with the following questions.

- Which node should receive a replica of a shared object?
- When should replicas be deleted or migrated to other nodes?
- How can replicas be located efficiently in a dynamic scenario, where the distribution of replicas in the network keeps changing?
- From which replicas should a given read request for a shared object be served?

In particular the latter issue shows that data management is somehow a generalization of routing.

We develop strategies for two entirely different scenarios with entirely different optimization goals. In the first scenario the goal is to optimize the *performance* of a parallel application. For this, it is of major importance to reduce the communication overhead in the interconnection network of the parallel system, as this is usually the main bottleneck, and thus has a crucial impact on the overall performance. We do this by minimizing the congestion in the network, that is the maximum amount of data transmitted by a network link. This cost-measure guarantees that the communication load is distributed evenly among all network resources.

We present routing and data management algorithms that for any given network topology, and any given sequence of requests obtain close to optimum congestion. Further, our algorithms serve all requests using only local information stored at the network nodes. This allows an easy and efficient implementation in a distributed environment. The results for the first scenario are presented in Chapter 2.

In the second scenario, presented in Chapter 3, we investigate so called utility computing models [JGN99], in which resource usage is not for free, but a fee is charged dependent on the bandwidth and storage capacity utilized by an algorithm. These models are mainly motivated by the emerging Grid technology which aims at providing standardized and easy-to-use methods for sharing resources between different institutions and organizations. Currently, these organizations are mainly from the scientific and educational sector, and therefore the incentive for sharing resources is usually assumed to be the common interest of all participants. However, with the growing commercial interest in Grid computing, resources sharing methods based on monetary transactions become more and more important.

In such an environment a parallel application may not only be judged upon its performance, but also upon the monetary cost it induces. We develop data management strategies that try to minimize the total cost. We consider static data management strategies, i.e., strategies that for a given request pattern, calculate a *static* placement of objects which does not change during the running time of the parallel application.

We show how to compute a cost-optimal placement for tree networks in polynomial time. For general networks the problem is MaxSNP-hard, which means that there is no PTAS unless P equals NP. Therefore, we investigate approximation algorithms for this case. We present an algorithm that calculates a constant factor approximation of the optimum solution in polynomial time.

1.1 Bibliographical Notes

Most of the results presented in this thesis have been previously published in preliminary form in various conference proceedings.

In [Räc02] it was proved that for any network topology there exist data management and routing strategies that always (i.e., for any request sequence) obtain close to optimum congestion. We improved this result in [BKR03] where we showed that the data management and routing strategies can be computed in polynomial time. These two publications form the basis for the results given in Section Section 2.3.

In Section Section 2.4 it is shown that for the routing problem even the optimum strategy (i.e., the strategy with best possible competitive ratio) can be computed efficiently. This result has been previously published as joint work in [ACF$^+$03].

The results of the second scenario, where the goal is to minimize the monetary cost induced by a data management strategy are presented in Chapter Chapter 3. These results have been previously published in [KRW03].

CHAPTER 2

Communication-efficient Data Management Strategies

In this section we develop data management and routing strategies that are communication-efficient in the sense that they produce only little communication overhead in the interconnection network of a parallel or distributed system. This approach is motivated by the observation that, e.g., in most parallel computers the communication network is the major bottleneck for the performance of the system, because of an imbalance between low communication bandwidth on the one side, and relatively high processor speed on the other side.

This imbalance has always been an important issue in the area of parallel and distributed computing and its relevance has even increased in recent years, since, for economic reasons, much more industrial effort is made to increase processor speed than to improve network bandwidth and latency. In particular, todays low-cost parallel computers that are mostly constructed out of commodity components, often have serious bandwidth limitations. Another yet even more important example of bandwidth restricted networks is the Internet, where the performance of applications as, e.g., the WWW, is nearly completely determined by network characteristics.

In order to obtain a good performance in such systems it is mandatory to design applications in a communication-sensitive manner. However, simply minimizing the total number of messages in the system, or the *total communication load*, i.e., the sum, taken over all messages, of the size of the message multiplied by the length of the routing path, is not sufficient, as this may result in bottlenecks. In addition, the communication load has to be distributed evenly among all network resources. This corresponds to minimizing the congestion which is the maximum amount of data transmitted by a single network link. Previous work on routing [LMRR94, SV98] and data management [MMVW97] shows that reducing the congestion is very important in order to get a good communication performance. However, most existing methods for congestion minimization focus on specific network topologies like fat trees, meshes, hypercubes, etc. For applications in highly unstructured networks, like the Internet, these methods do not yield satisfactory solutions.

We propose a basic framework for minimizing the congestion in general topology networks. At the heart of our framework is a powerful decomposition technique that makes it possible to transform any given network into a tree network with nearly equivalent communication characteristics (this means the tree has, e.g., the same bandwidth between node-pairs, the same bottlenecks, etc.). Due to the simple structure of tree networks many congestion-related problems can be solved quite elegantly and efficiently for trees. Our decomposition technique allows it, to transfer such a tree solution to a solution for general topologies with only a little loss in efficiency. The framework can be applied for any problem that aims to minimize the congestion and that has an efficient tree solution.

We evaluate our framework on the basis of the following three fundamental problems from the area of distributed computing: virtual circuit routing, multicast routing, and data management. For all these problems we get solutions that work totally distributed and produce only little communication overhead, i.e., the created congestion is guaranteed to be close to optimal.

2.1 Formal description of the problems

In this section we give a formal description of the problems that we will solve using our general framework. We model the network as a complete graph $G = (V, E)$. We use $n = |V|$ to denote the number of nodes in G. Network links are represented via a weight function $\mathrm{b} : V \times V \to \mathbb{R}_0^+$ that describes the link capacities between node-pairs. Usually we assume that the network is undirected, i.e., we assume $\mathrm{b}(u, v) = \mathrm{b}(v, u)$ for any two nodes $u, v \in V$.

The common goal for all our problems is to minimize the congestion which is defined as follows. For a given problem solution, let the *(absolute) load* of an edge be the amount of data transmitted by the corresponding network link. Let the *relative load* of an edge be its load divided by its bandwidth. We define the *congestion* of the problem solution to be the maximum relative load of a network link.

We focus our analysis on the following three problems.

Virtual circuit routing. The first and most basic problem that we consider is the virtual circuit routing problem. In this problem we are given routing requests $(s_1, t_1), (s_2, t_2), \ldots$, where each request consists of a source node s_i and a target node t_i. The task is to select a routing path in G for each request that connects the source to the target. For simplicity we assume a uniform cost model in which each routing request is weighted equally, i.e., each request induces load 1 on each edge of its routing path. The goal is to minimize the congestion.

Multicast routing. The second problem is the multicast routing problem which is a generalization of virtual circuit routing. In the multicast routing problem we are given a collection of M multicast groups, each with a source node $s_i \in V$, $1 \leq i \leq M$. (Usually the source supplies some information that has to be transmitted to all members of the corresponding group)

Further, we are given subscription requests that consist of pairs (i, v) with the meaning that node $v \in V$ has to be connected to the *i-th* multicast group (i.e., node v wants to get the information supplied by s_i). For such a request we say that node v *subscribes to* multicast group i.

For transmitting data from the source s_i to the subscribing nodes of a multicast group, a multicast routing algorithm has to select a Steiner tree that spans the source and all subscribing nodes. Similar to the virtual circuit routing problem this increases the load on each edge of the selected tree by 1. The goal is to minimize the congestion.

An important characteristic of our problem definition is that we do not require that all subscribers to a specific multicast group appear at once, but subscriptions to different multicast groups may interleave, arbitrarily. Hence, a routing algorithm has to build up the Steiner tree incrementally, i.e., it has to connect a subscribing node to its multicast group without knowing all requests directed to the group.

Data management. The third problem is a data management problem that was introduced in [MMVW97]. In this problem we are given a set X of shared data objects, and nodes of the network may issue read or write requests to these objects. The task is to serve all read and write requests in such a way that the congestion in the network is minimized.

In general, a data management strategy has to decide where to create copies of shared objects and when to delete or migrate copies to other nodes. Further, it has to assign read requests to copies, i.e., it has to decide for every request from which copy the request is served, and along which path in the network the corresponding data is transmitted. More precisely, a data management strategy has to do the following.

- In case of a read request for some object $x \in X$ issued at a node $v \in V$, the data management strategy has to select a path from v to a node $u \in V$ holding a copy of x, along which the content of x is sent to v. This increases the load on any edge of the path by 1.
- In case of a write request for object $x \in X$ issued at node $v \in V$, the data management strategy has to select a Steiner tree that contains v and all nodes

holding a copy of x. All copies of x can then be updated along this Steiner tree. This increases the load on every edge of the Steiner tree by 1.

- We focus on dynamic scenarios, where the distribution of copies is allowed to change during the running time of an application. For this purpose a data management strategy may initiate migrations.

  During a migration for object $x \in X$ a data management strategy chooses a tree T in G that contains at least one copy of x. Then it changes the distribution of copies for x within the node set of T (e.g., the strategy may delete all copies, move a copy to every node of T, etc.). However, the strategy has to ensure that always at least one copy of x remains in the network. A migration increases the load on every edge of the selected tree by 1.

- A data management strategy may combine the service of a read or write request for object $x \in X$ with a migration for x. This means when serving, e.g., a write request it may change the distribution of copies for x within the node set of the selected update tree without additional communication. Similarly, when serving a read request it can change the distribution of copies along the traversed path between the reading node v and the serving node u.

Note that the above procedure of performing write accesses and migrations mirrors the fact that we consider writes to be object modifications rather than overwrites. This means that write accesses cannot be ignored, even in the case of immediately consecutive writes. Further, new copies (for the case of dynamic scenarios where placement of copies can change) cannot be created from scratch but the value to be written has to be merged with the current content of the global object.

For the initial configuration we define that each object $x \in X$ has a home $h(x)$ that holds the initial copy of x.

Competitive analysis

We analyze our problems in the framework of competitive analysis as introduced in [ST85]. Our strategies have to work online, i.e., they receive the requests one by one, in form of a requests sequence. An online strategy has to serve the requests *ad hoc*, i.e., a request σ_i has to be served before the request σ_{i+1} is presented to the strategy. An online algorithm is said to be *c-competitive* if for all requests sequences $\sigma = \sigma_1, \sigma_2, \ldots$

$$C_{\text{onl}}(\sigma) \leq c \cdot C_{\text{opt}}(\sigma) + a \ , \tag{2.1}$$

where $C_{\text{onl}}(\sigma)$ and $C_{\text{opt}}(\sigma)$ denote the congestion obtained by the online algorithm and by an optimal offline algorithm, respectively, for request sequence σ. Randomized algorithms must fulfill this bound with high probability, i.e., they must create congestion at most $\alpha \cdot (c \cdot C_{\text{opt}}(\sigma) + a)$ with probability at least $1 - 1/n^{\alpha}$. The value c is also called the *competitive ratio* of the online algorithm.

The term a is a constant that does not depend on the request sequence σ, but may depend on parameters of the network. Thus, the additive term a can dominate the expression $c \cdot C_{\text{opt}}(\sigma) + a$ for a long time until the request sequence σ reaches the required length. For the special case that Equation (2.1) holds with $a = 0$, i.e., the congestion of the online algorithm and the optimal congestion always differ by a factor of at most c, the online algorithm is called *strictly c-competitive*.

The above description is the standard definition of competitive analysis as introduced in [ST85]. We require that our online algorithms fulfill another property that is very important for implementations in a distributed environment. Our online algorithms have to work in a distributed fashion. This means that the individual nodes do not have information about the global state of the system, but only have local knowledge that is sent to them via explicit messages, which of course increase the communication load.

This requirement makes the development of online routing and data management algorithms a challenging task. However, we have to cope with the following difficulties.

- For a solution of the virtual circuit routing problem that works in a distributed environment the routing path chosen for a request may only depend on previous routing requests that *are known* to the source of the request (we assume that the source node fixes the whole routing path).

  However, we investigate algorithms that are even more restricted. For an *oblivious* algorithm the route chosen for a request may not depend on any other request, which means that routing decisions are completely independent from the traffic in the network. Such algorithms can be implemented very easily and efficiently in a distributed setting. We present oblivious routing algorithms that for any routing problem obtain close to optimum congestion.

- In our distributed setting dynamic data management strategies have to deal with the data tracking problem. In case of a read or write access the strategy must be able to locate one or all copies in the network by using only local information stored at the nodes. In [MMVW97] it is shown that this problem can be solved in a very elegant fashion on tree networks.

  We will benefit from this tree solution by transforming it into a solution on general graphs. This example shows that our approach of simulating tree strategies on general topologies is very attractive, since it allows to profit from previous results on tree networks. The development of a distributed data tracking scheme on arbitrary networks from scratch would have been very difficult.

The notion of a request sequence that specifies the order in which requests are given to a strategy is slightly restrictive since such a model serializes the requests, i.e., it does not allow parallel and concurrent requests. However, for virtual circuit and multicast routing, our strategies also work for concurrent requests. (For virtual circuit routing this is directly clear, since we develop oblivious algorithms.)

For the data management problem we show in Section 2.3.5 how to transform our results to scenarios in which a certain concurrency among requests is allowed.

2.2 Related work and new results

In this section we give a brief overview on previous work and describe our new results on the static and dynamic versions of the virtual circuit routing problem, the multicast routing problem and the data management problem. Note that data management is a generalization of multicast routing[1] and that this, in turn, is a generalization of virtual circuit routing. Therefore, results for data management also hold for multicast and virtual circuit routing.

2.2.1 Previous work

Virtual circuit routing. In the *offline setting* of the virtual circuit routing problem all routing requests are given in advance, which results in a static optimization problem, a so called *concurrent multicommodity flow problem* (CMCF-problem). Raghavan and Thompson [RT87] show how to obtain a routing algorithm from a fractional solution to the CMCF-problem, via randomized rounding. This gives a centralized offline algorithm that well approximates the minimum possible congestion. Awerbuch and Leighton [AL94] give a distributed algorithm for the (fractional) CMCF-problem and, hence, obtain a distributed offline solution for the routing problem.

In the *online setting* routing requests arrive during the running time and have to be served "ad-hoc" by an online algorithm. Therefore, a routing path chosen for a request σ_i may not depend on a future request σ_j, $j > i$, since this request is not yet known when σ_i is served. Leonardi [Leo98] gives an excellent survey on online network routing that also includes models that do not aim to minimize the congestion but to maximize throughput, i.e., the number of messages that can be routed without exceeding the capacity of the network.

Aspnes et al. [AAF$^+$97] design an $O(\log n)$-competitive online routing algorithm for general directed networks. The algorithm assigns a length to every edge

[1]To see this choose for the i-th multicast group a shared object x with home $h(x) = s_i$. Add an initial write request from node s_i to x, and transform every subscription request (i, v) to the i-th group into a read request for x issued at node v. The data management strategy has to select a Steiner tree that connects all nodes of the multicast group in order to deliver the content of x to all readers.

e of G that is exponential in the current load of e. When a routing request occurs it is routed along the shortest path between source and destination, w.r.t. the current length function. Unfortunately, this approach requires centralized control and serializes the routing requests, i.e., requests cannot be served concurrently. Awerbuch and Azar show how to choose the routing path in a distributed and concurrent way by repeatedly scanning the network. However, their algorithm requires shared variables on the edges of the network and, therefore, is hard to implement.

Apart from the upper bound, the paper by Aspnes et al. [AAF$^+$97] also gives a lower bound of $\Omega(\log n)$ on the competitive ratio of randomized online algorithms on certain directed networks. Subsequently, it has been shown independently by Bartal and Leonardi [BL97], and by Maggs et al. [MMVW97] that a lower bound of $\Omega(\log n)$ on the competitive ratio already holds for the 2-dimensional mesh.

In all the above routing algorithms the paths chosen for routing requests depend on other requests, which makes these algorithms *adaptive*. *Oblivious routing* is a totally different approach. For an oblivious algorithm the route chosen for a request may not depend on any other request, which means that routing decisions are completely independent from the current traffic in the network. In fact, for an oblivious algorithm a routing path may only depend on the source node, the target node, and on some random bits for the case of randomized routing algorithms. Therefore, such an algorithm can be implemented very easily via routing tables that store at each node v, a probability distribution over paths for each possible destination node.

Because of their simple implementations, much effort has been made to design oblivious routing algorithms for specific network topologies. Valiant and Brebner [VB81] were the first to perform a worst case theoretical analysis for oblivious routing on the hypercube. They design a randomized packet routing algorithm that routes any permutation in $O(\log n)$ steps. These results give an $O(\log n)$-competitive virtual circuit routing algorithm, as well. The techniques developed in this paper, in particular the technique of routing to random intermediate destinations, were later used for efficient permutation routing in many other networks.

Scheideler [Sch98], e.g., uses the *routing number* of a network G, which is defined as the maximum, taken over all permutations, of the minimum amount of time to route the permutation in G using a best possible strategy. He shows that for any constant degree network of size n with routing number R, there is an oblivious algorithm that routes any permutation in time $O(R + \log^{1+\epsilon} n)$ with high probability. This result is worst case optimal, but for certain routing problems (such as sending packets to direct neighbors) this bound may be very weak, as it guarantees no competitive ratio.

In [MMVW97] and [Vöc98] the authors develop data management strategies that aim to minimize the congestion in a large variety of regular network topologies, as, e.g., meshes of arbitrary dimension, hypercubic networks, or Caley networks. Since data management is a generalization of routing, these results give oblivious routing algorithms with polylogarithmic competitive ratio for the specified topologies.

Despite of these successes in the design of oblivious algorithms, it was a common belief that the simple structure of oblivious algorithms would lead to a bad performance on many networks. This assumption was partly supported by the fact that *deterministic* oblivious algorithms perform very poorly, on most topologies. Borodin and Hopcroft [BH85] have shown that in any network with constant degree, there is a permutation routing problem that has congestion $\Omega(\sqrt{n})$. This means that, e.g., on a butterfly network, deterministic oblivious routing algorithms have a competitive ratio of $\Omega(\sqrt{n}/\log n)$, as any permutation routing problem can be solved with congestion $O(\log n)$. Further lower bounds on oblivious routing in the hypercube can be found in [ALMN91] and [KKT90].

Multicast routing. In the offline version of the multicast routing problem it is not directly possible to use randomized rounding of an optimal fractional solution to obtain a good approximation. This difference to virtual circuit routing is due to the fact that a collection of fractional Steiner trees (the output of the fractional solution for multicast routing), cannot be decomposed into a convex combination of trees, whereas a collection of flows (fractional solution for virtual circuit routing) can be decomposed into a convex combination of paths (cf. [AMO93]). Carr and Vempala [CV02] show how to obtain a convex combination of trees for which the congestion is only a factor of 2 away from the congestion of an optimum fractional solution. Applying randomized rounding they get an integral solution with congestion at most $2 \cdot C_{\text{opt}} + O(\log n)$.

In the online version the approach that uses an exponential length function and returns a minimum spanning tree w.r.t. this function for each request, gives a competitive ratio of $O(\log n)$, exactly as in the virtual circuit routing problem (see [AAF$^+$97]). For the case of interleaving subscription patterns, Awerbuch and Azar [AA95] design an online algorithm with competitive ratio $O(\log n \cdot \log d)$, where d is the maximum size of a multicast group.

Data management. The problem of distributing and accessing shared objects in networks has been investigated in various different forms. We first give an overview over the most important theoretical models and the results obtained in these models. Then we present the previous work concerning the static and dynamic data management problem as defined in Section 2.1.

The first work concerning static data management deals with heuristics for the static file allocation problem, in which files have to be assigned to nodes of a network, such that a cost function that, e.g., models the communication or storage cost is minimized. Dowdy and Foster [DF82] give a survey about several different models, and describe these models using mixed integer programs with different cost functions and constraints. Analogously to our static model, all models use the standard write policy, i.e., a write request has to update all copies of an object and a read request can be satisfied by any one copy. However, none of the described models allows that copies are updated via a multicast tree.

Wolfson and Milo [WM91] consider static data management in a model that allows multicast trees for copy updates and that aims at minimizing the total communication load. They develop algorithms that calculate optimal placements for cycles, trees, and complete networks.

Bartal et al. [BFR95] introduce a dynamic variant of the file allocation problem that is analyzed in an online framework. The main objective in this model is to minimize the total communication load and not the congestion as in our model. They obtain a centralized file allocation strategy that achieves optimal competitive ratio $O(\log n)$, and a distributed version that achieves competitive ratio $O(\log^4 n)$. Both algorithms are randomized. Subsequently, Awerbuch et al. [ABF93] have shown that randomization is not crucial. They design deterministic algorithms that achieve the same bounds as their randomized counterparts. In [ABF98] the distributed algorithm is also adapted to a model in which the memory capacity at the network nodes is limited.

The data management problem with the goal to minimize the congestion was investigated in [MMVW97]. Therein, the authors first design static and dynamic algorithms for tree networks. They obtain an algorithm that calculates an optimal solution for the static data placement problem in linear time, and an algorithm for the dynamic problem that achieves competitive ratio of 3. The latter result is optimal due to a lower bound of Bartal [BFR95].

The main result of [MMVW97] is a bisimulation technique that makes it possible to transfer tree solutions for the congestion based data management problem to other networks. For applying this technique the authors need a hierarchical decomposition of the network for which a data management strategy has to be designed. It is shown in [MMVW97] how to obtain such a decomposition for meshes and for networks that fulfill a certain clustering property. For a d-dimensional mesh, e.g., the authors obtain a static data management strategy with approximation ratio $O(d \cdot \log n)$, and a dynamic strategy that is $O(d \cdot \log n)$-competitive. In [Wes00] it is shown that the approach works for a variety of quite regular networks as, e.g., Caley networks and hypercubic networks, as well.

It is conjectured in [Vöc98] that the bisimulation approach can be generalized to arbitrary networks by designing a suitable decomposition scheme for general topologies. One consequence of our work is that this conjecture is true for general, undirected graphs but that it does not hold for directed graphs.

There exist several further extension of the congestion based data management problem. In [MVW99] the model is extended to scenarios where reads and writes only access a small part of a global object. In this model migration messages that always affect a whole object, cause more communication overhead than read or write messages. In [MVW00] and [Wes00] the model is further extended to incorporate limited memory capacity at the network nodes.

2.2.2 Our results

We generalize the bisimulation technique introduced in [MMVW97] to general networks. This approach depends on a hierarchical decomposition of the target network (i.e., the network for which a routing or data management algorithm has to be designed). We first specify the characteristics that a hierarchical decomposition has to fulfill in order to allow for good routing and data management strategies. Then we show that a hierarchical decomposition with these characteristics exists for any undirected network. Moreover, we show that a slightly weaker decomposition can be constructed in polynomial time.

Based on the decomposition we then develop algorithms for virtual circuit routing, multicast routing, and data management. For the online versions of these problems, we obtain a competitive ratio of $O(\log^3 n)$, with respect to the congestion of the network links. All algorithms can be implemented efficiently in a distributed environment.

For the routing problems these results also give polynomial time algorithms that only create congestion $O(\log^3 n)\cdot C_{\mathrm{opt}}+O(\log n)$ for the static problem versions, when all routing requests are known in advance.[2] At first glance, these results may seem quite unsatisfactory, since, e.g., for virtual circuit routing or multicast routing the already mentioned results of [RT87] and [CV02] obtain far better bounds on the congestion. However, the real advantage of our strategies is that they are extremely fast, once the hierarchical decomposition of the network has been computed. For the virtual circuit routing problem, for example, the preprocessing of the hierarchical decomposition gives a unit flow between every pair of nodes. A routing problem is then solved by scaling each unit flow by the demand between the corresponding node pair. Obviously this can be performed for all node pairs in time $O(|E| \cdot |D|)$, where $|E|$ denotes the number of links, and $|D|$ is the number of node pairs with nonzero demand. This is much faster than solving a fractional multicommodity flow problem as in [RT87].

For meshes, the uniform bound of $O(\log^3 n)$ on the competitive ratio is too large. We show how to obtain an upper bound of $O(\log n)$ on the competitive ratio on meshes of arbitrary dimension. This is done by showing that the properties of a straightforward decomposition of the mesh, are better than the properties

[2]However, these results do not give an approximation algorithm. For this one needs to relate C_{opt} to $O(\log n)$ in some way (note that the optimum congestion can be much lower than 1).

guaranteed by our general decomposition technique. Note that the bound of $O(\log n)$ for d-dimensional meshes improves the bound given in [MMVW97] by a factor of d.

For the virtual circuit routing problem we also present strategies that do not depend on a hierarchical decomposition of the target network, but on linear programming. We show that for any given network we can compute the optimal oblivious routing scheme in polynomial time.

2.2.3 Further work

The results presented in this thesis have been improved and extended in various ways by several researchers. Harrelson et al. [HHR03] present an improved partitioning algorithm that guarantees a competitive ratio of $O(\log^2 n \log\log n)$ instead of $O(\log^3 n)$ for all our applications.

Applegate and Cohen [AC03] show that the optimal oblivious routing scheme can be computed by an LP with polynomial number of variables and constraints. This gives an enormous improvement on the running time compared to our solution that uses the Ellipsoid method.

Moreover, Maggs et al. [MMP+03] have shown that our hierarchical decomposition can be used to speed up iterative solutions to linear equations.

2.3 Hierarchy-based algorithms

In this section we present data management and routing algorithms that aim to minimize the congestion and that are based on a hierarchical decomposition of the target network. Algorithms that do not depend on a hierarchical network decomposition but on linear programming are presented in Section 2.4.

2.3.1 Preliminaries

We model the network as a complete weighted undirected graph G with node set V and edge set $E = V \times V$. We use n to denote the cardinality of V, i.e., $|V| = n$. Network links are represented via a weight function $\mathrm{b} : V \times V \to \mathbb{R}_0^+$ that for a pair of nodes describes the link-capacity between these nodes. If $\mathrm{b}(u, v) = 0$ for two nodes u and v, then there is no link between these nodes in the physical network. Note that the graph G is undirected which means that we assume $\mathrm{b}(u, v) = \mathrm{b}(v, u)$ for any two nodes $u, v \in V$. Furthermore, we assume that the weight function b is normalized, i.e., the minimum nonzero capacity of a link is 1. We denote the maximum capacity of a network link with $\mathrm{b}_{\max}$.

We define a function $\mathrm{cap} : 2^V \times 2^V \to \mathbb{R}_0^+$ which for two subsets $X, Y \subseteq V$ describes the total link-capacity that is available between nodes of X and nodes of Y. It is defined as follows:

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

For a set $X \subseteq V$ we denote the total capacity of edges leaving set X in G with $\mathrm{out}(X) = \mathrm{cap}(X, \overline{X})$, where $\overline{X} := V \setminus X$.

2.3.2 The bisimulation technique

In the following we sketch the general concept for solving problems that aim to minimize the congestion in the communication network of a parallel or distributed system. The framework describes how to transform a problem solution for tree networks into a problem solution for general networks with arbitrary edge capacities. It is based on a bisimulation between the physical network $G = (V, E)$ and a virtual tree network $T = (V_\mathrm{t}, E_\mathrm{t})$ that is suitably constructed out of G. The framework consists of the following three steps.

In a first step it is shown that the tree T can simulate the network G in the following way. Suppose that we are given a problem instance I on G. We translate I into a problem instance I_t on T by mapping nodes of G to nodes of T via a certain, carefully chosen mapping $\pi_1 : V \to V_\mathrm{t}$. Now, we use a solution for I to derive a solution for I_t, as follows. Whenever the original solution sends a message between two nodes $u, v \in V$, a corresponding message is sent between nodes $\pi_1(u), \pi_1(v) \in V_\mathrm{t}$. It can be shown that the congestion produced in T by this simulation scheme is at most as large as the congestion of the original solution to I in G.

In a second simulation step we show that there is a probabilistic simulation of T on the network G such that for every edge of G, the expected load is only a small factor f larger than the congestion on the tree. More formally, there is a (randomized) mapping $\pi_2 : V_\mathrm{t} \to V$ that transforms a problem instance that can be solved on the tree with congestion C_t, into a problem instance on G that can be solved such that $\forall e \in E$: $\mathrm{E}[\mathrm{L}(e)] \leq f \cdot C_\mathrm{t} + K$, where $\mathrm{E}[\mathrm{L}(e)]$ denotes the expected load of an edge e. The term K denotes a constant that does not depend on parameters of the problem instance, as, e.g., the number of requests, but may depend on the network G.

For this simulation a node $v \in V$ may have to simulate several nodes of V_t, since usually the virtual tree network contains more nodes than the physical network G. In addition to the mapping $\pi_2 : V_\mathrm{t} \to V$ we describe for this simulation step how the routing between host nodes in G that simulate adjacent tree nodes is performed. Note that such a routing definition is not needed for the first simulation since there is a unique routing path between any two nodes in T. The exact description of the routing and the definitions of π_1 and π_2 will be given in

Section 2.3.4. For the purpose of this section it is only important that $\pi_2(\pi_1(v)) = v$ holds for each $v \in V$.

The above simulations, i.e., the first two steps of our framework, are completely independent of the problem that has to be solved. In a final *problem-specific* step we use standard techniques from probability theory in order to show that the result from the second simulation step holds with high probability, i.e., $\forall e \in E : \ \mathrm{L}(e) \leq f' \cdot C_{\mathrm{opt}}(I_{\mathrm{t}}) + K'$, w.h.p. for some constants f' and K'. Note that this implies that the congestion in G due to simulation of the tree is at most $f' \cdot C_{\mathrm{opt}}(I_{\mathrm{t}}) + K'$, with high probability.

How do these results help us in solving problems that aim to minimize the congestion in the network G? A condition for our framework is that the respective problem can be solved efficiently for trees. Thus, suppose that there is a tree strategy that for a problem instance I_{t} on T, generates a solution with congestion at most $f_{\mathrm{t}} \cdot C_{\mathrm{opt}}(I_{\mathrm{t}}) + K_{\mathrm{t}}$, where $C_{\mathrm{opt}}(I_{\mathrm{t}})$ denotes the optimal congestion that can be achieved for instance I_{t}, and f_{t} and K_{t} denote parameters that only depend on the network G.

We get a strategy for G as follows. Let I denote a problem instance on G. First, we translate this problem instance into the corresponding instance I_{t} on T using the mapping π_1. Let $C_{\mathrm{opt}}(I)$ and $C_{\mathrm{opt}}(I_{\mathrm{t}})$ denote the optimal congestion for instances I and I_{t}, respectively. Further, let $C_{\mathrm{alg}}(I_{\mathrm{t}})$ denote the congestion produced by the tree strategy when solving problem instance I_{t}. The first simulation result and the properties of the tree strategy give

$$C_{\mathrm{opt}}(I) \geq C_{\mathrm{opt}}(I_{\mathrm{t}}) \geq \frac{1}{f_{\mathrm{t}}} \cdot C_{\mathrm{alg}}(I_{\mathrm{t}}) - K_{\mathrm{t}} \ .$$

Now, we can map the problem instance I_{t} and its solution back to the network G using the mapping π_2. Since $\pi_2(\pi_1(v)) = v$ for every $v \in V$ we get a solution to the original instance I on G, as desired. For the congestion $C_{\mathrm{alg}}(I)$ of this solution we have

$$C_{\mathrm{alg}}(I) \leq f' \cdot C_{\mathrm{alg}}(I_{\mathrm{t}}) + K' \leq f' \cdot f_{\mathrm{t}} \cdot C_{\mathrm{opt}}(I) + K' + K_{\mathrm{t}} \ \text{, w.h.p.}$$

Hence, we get some bound on $C_{\mathrm{alg}}(I)$ in terms of $C_{\mathrm{opt}}(I)$.

What are the main challenges for applying this bisimulation approach? The most important and most difficult part is to choose the virtual tree network T in such a way that the simulations work properly. Fortunately, this part does not depend on the concrete problem that has to be solved. It has to be performed only once for a network G, and then this solution can be used for any problem that aims to minimize the congestion in G. This basic simulation result is developed in sections 2.3.3, 2.3.4, and 2.3.6.

The second task is to transfer the simulation result on the expected load of the edges of G into a result on the congestion. As stated before, this part uses standard techniques from probability theory, as, e.g., Chernoff bounds, but the analysis has to be performed for each problem individually. In Section 2.3.5 this task is solved for a variety of problems as, e.g., data management, virtual circuit routing and multicast routing.

2.3.3 The virtual tree network

The virtual tree network that is used for the bisimulation technique described in the previous section, corresponds to a *hierarchical decomposition* of the physical network G. Formally, a hierarchical decomposition $\mathcal{H}$ of the graph G is a set system over the universe V that fulfills the following properties.

- $\mathcal{H}$ is *laminar*, i.e., for two subsets $X, Y \in \mathcal{H}$ either $X \setminus Y$, $Y \setminus X$ or $X \cap Y$ is empty.
- $\mathcal{H}$ contains V and all sets $\{v\}$, $v \in V$.

Such a hierarchical decomposition describes a partitioning process in which the graph G is partitioned into smaller and smaller subgraphs until the remaining subgraphs contain only a single node of G. There is a natural tree associated to this decomposition, in which each node represents a subgraph and the children of a node represent the subgraphs that result from partitioning the parent. Essentially our virtual tree network used for the simulations will be such a natural tree, corresponding to a specific hierarchical decomposition $\mathcal{H}$. However, for technical reasons we add the following extra nodes. For each edge (u, v) of the tree we add an intermediate node x and replace edge (u, v) by edges (u, x) and (x, v). The tree $T_{\mathcal{H}}$ obtained by this process is called the *decomposition tree* corresponding to the hierarchical decomposition $\mathcal{H}$.

In order to introduce all terms and definitions properly we define the decomposition tree $T_{\mathcal{H}} = (V_{\mathrm{t}}, E_{\mathrm{t}})$ more formally. The node set $V_{\mathrm{t}} = V_{\mathrm{t}}^N \uplus V_{\mathrm{t}}^I$ of the tree consists of a set V_{t}^N of *natural nodes* and a set V_{t}^I of *intermediate nodes*. For each set $H \neq V$ from the laminar system $\mathcal{H}$ the tree contains two nodes; a natural node v_{n} and an intermediate node v_{i}. We call H the set or cluster corresponding to v_{n} and v_{i}. Further, v_{n} and v_{i} are called the *natural node* and *intermediate node*, respectively, corresponding to cluster H. For $H = V$ the tree contains only a natural node.

In the following the cluster corresponding to a node $v \in V_{\mathrm{t}}$ will be denoted with H_v. A natural node v_{n} and an intermediate node v_{i} in $T_{\mathcal{H}}$ are connected if either $H_{v_{\mathrm{n}}} = H_{v_{\mathrm{i}}}$, or if $H_{v_{\mathrm{i}}} \subsetneq H_{v_{\mathrm{n}}}$ and if there is no $H \in \mathcal{H}$ such that $H_{v_{\mathrm{i}}} \subsetneq H \subsetneq H_{v_{\mathrm{n}}}$. Note that by this definition $T_{\mathcal{H}}$ is indeed a tree, since $\mathcal{H}$ is a laminar system. We assume $T_{\mathcal{H}}$ to be rooted at the node corresponding to cluster V. By this definition the root and the leaves of $T_{\mathcal{H}}$ are natural nodes and the leaves correspond to

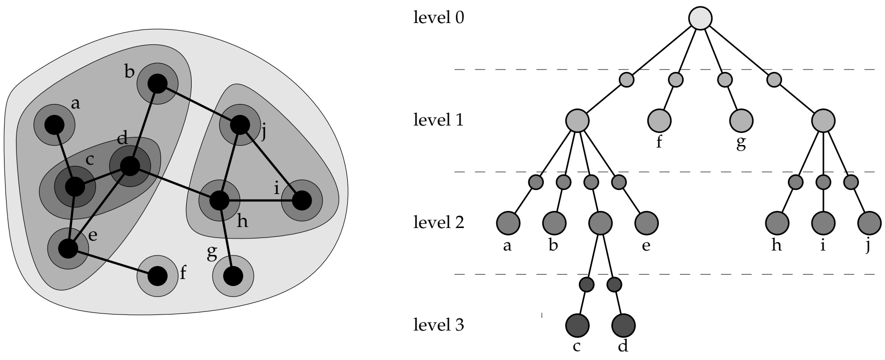

Figure 2.1: A hierarchical decomposition of a graph and the associated decomposition tree. Large circles in the right figure represent natural nodes while small circles represent intermediate nodes.

sets $\{v\}, v \in V$, i.e., there is a one-to-one relation between nodes of G and leaf nodes of $T_{\mathcal{H}}$.

It remains to define the bandwidths for the edges of the decomposition tree. An edge $(v_\mathrm{n}, v_\mathrm{i}) \in E_\mathrm{t}$ is assigned a bandwidth of $\mathrm{b_t}(v_\mathrm{n}, v_\mathrm{i}) := \mathrm{out}(H_{v_\mathrm{i}})$, i.e., the bandwidth of all edges leaving the cluster corresponding to the intermediate node of the edge.

We define levels for nodes and edges in $T_{\mathcal{H}}$, as follows. The level of a node v_t of $T_{\mathcal{H}}$ is defined as the number of natural nodes on the path from v_t to the root, not counting v_t. The level of an edge $(v_\mathrm{n}, v_\mathrm{i}) \in E_\mathrm{t}$ is defined as the level of the natural node v_n of the edge. Further, we define the level of a cluster H of the laminar system as the level of a corresponding node in $T_{\mathcal{H}}$. (Note that both nodes corresponding to H are on the same level.) Finally, we say that an edge e of G is *cut* on level $\ell \geq 1$ if both endpoints of e are contained in the same level $\ell - 1$ cluster but in different level ℓ clusters. We use level(e) for an edge $e \in E$ to denote the level on which e is cut. Figure 2.1 gives an example of a hierarchical decomposition and the corresponding decomposition tree.

So far, we have described the general structure of the virtual tree network, namely that it is a decomposition tree for some hierarchical partitioning $\mathcal{H}$. In Section 2.3.4 we will show that every decomposition tree can simulate the physical network G, i.e., the first simulation step works regardless of how we choose $\mathcal{H}$. Now, we define a parameter for a hierarchical partitioning that somehow measures how well the second simulation step works, i.e., how well the corresponding decomposition tree can be simulated by G. Then, in Section 2.3.6, we will show how to construct a partitioning for which this parameter has a good value.

In order to specify our parameter, we first introduce a weight function $\mathrm{w}_\ell : 2^V \to \mathbb{R}_0^+$ for each level $\ell \in \{0, \ldots, \mathrm{height}(T_{\mathcal{H}})\}$ of the decomposition tree, as follows:

$$\mathrm{w}_\ell(X) := \sum_{\substack{e \in X \times V \\ \mathrm{level}(e) \leq \ell}} c(e) \enspace .$$

Informally speaking, the weight function $\mathrm{w}_\ell(X)$ counts for a subset X, the capacity of all edges that are adjacent to nodes in X and are cut before, or at level ℓ in the hierarchical decomposition. The following properties of w_ℓ will be used intensively throughout this chapter. 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 H_{v_t} we have $\mathrm{w}_\ell(H_{v_\mathrm{t}}) = \mathrm{out}(H_{v_\mathrm{t}})$. Finally, $\mathrm{w}_{\ell-1}(X) \leq \mathrm{w}_\ell(X)$ holds for any $\ell \in \{1, \ldots, \mathrm{height}(T_{\mathcal{H}})\}$.

We define a fractional concurrent multicommodity flow problem (CMCF-problem) for each cluster of the hierarchical decomposition, as follows. Let H_{v_t} denote a level ℓ cluster of $\mathcal{H}$. The CMCF-problem for H_{v_t} has a commodity $d_{u,v}$ for each (ordered) pair $u, v \in H_{v_\mathrm{t}}$. The source for commodity $d_{u,v}$ is u, its sink is v and its demand is

$$\mathrm{dem}(u, v) := \frac{\mathrm{w}_{\ell+1}(u) \cdot \mathrm{w}_{\ell+1}(v)}{\mathrm{w}_{\ell+1}(H_{v_\mathrm{t}})} \enspace . \tag{2.2}$$

Let c_{v_t} denote the minimum possible edge congestion that can be achieved for a solution that routes all these demands using only edges inside H_{v_t}. Note that we may route fractionally and, hence, the value of c_{v_t} can be computed by solving a linear program. We define $c_{\mathcal{H}} := \max_{v_\mathrm{t} \in V_\mathrm{t}} c_{v_\mathrm{t}}$ as the maximum edge congestion that is needed for a CMCF-problem of a cluster of the hierarchical decomposition $\mathcal{H}$.

This parameter $c_{\mathcal{H}}$ will turn out to be very important for the second simulation step. Let $\delta_{\mathcal{H}}$ denote the factor between the expected load on edges in G and the congestion in the tree for this simulation. In the following section it is shown that there is a simulation such that $\delta_{\mathcal{H}} = O(c_{\mathcal{H}} \cdot \mathrm{height}(T_{\mathcal{H}}))$.

2.3.4 Simulation results

In this section we show how the bisimulation between G and $T_{\mathcal{H}}$ works.

Simulating G on $T_{\mathcal{H}}$

The simulation of the network G on a decomposition tree $T_{\mathcal{H}}$ works as follows. Suppose that we are given a problem instance I on G that can be solved with congestion C. We get a problem instance on $T_{\mathcal{H}}$ by mapping each node $v \in V$ to the leaf node in $T_{\mathcal{H}}$ that corresponds to cluster $\{v\}$. Now, for every message that is sent between nodes u and v in the solution to I on G, we send a message along the unique shortest path connecting the respective counterparts in $T_{\mathcal{H}}$.

The following theorem states that this simulation does not increase the congestion.

Theorem 2.1 *Let I denote a problem instance on G that can be solved with congestion C and let I_t denote the corresponding instance on $T_{\mathcal{H}}$. Then the instance I_t can be solved with congestion $C_t \leq C$.*

Proof. We show that the above simulation produces congestion at most C. Let $e_t = (v_n, v_i)$ denote an edge of $T_{\mathcal{H}}$ that connects a natural node v_n to an intermediate node v_i and that has relative load C_t, due to the simulation. Then, the absolute load of e_t is $C_t \cdot b_t(e_t)$.

Now, consider the congestion produced by the corresponding solution to I on G. Any message that crosses edge e_t in $T_{\mathcal{H}}$ has either to leave or to enter the cluster H_{v_i}. The total bandwidth of all edges leaving cluster H_{v_i} is $\text{out}(H_{v_i}) = b_t(e_t)$. Hence, one of those edges must have relative load $C_t \cdot b_t(e_t)/\text{out}(H_{v_t}) = C_t$. Therefore, $C \geq C_t$. ■

Simulating $T_{\mathcal{H}}$ on G

In this section, we show that the decomposition tree $T_{\mathcal{H}}$ can be simulated on the network G such that for every edge of G the expected relative load is small.

We first try to develop an intuition for the decomposition tree and the role of the hierarchical partitioning $\mathcal{H}$. The clusters in $\mathcal{H}$ can be viewed as *potential bottlenecks* in the network G. Of course, it may or may not happen that a cluster $H \in \mathcal{H}$ becomes a bottleneck for a specific problem instance. For example, if we deal with the problem of virtual circuit routing, and the problem instance consists of sending messages to direct neighbors, there is no bottleneck at all, and therefore no set in $\mathcal{H}$ is a bottleneck.

But we will show that, *if* there is a bottleneck in the system, then *at least one* of the sets in $\mathcal{H}$ is an approximate bottleneck, where the quality of the approximation depends on the value of $c_{\mathcal{H}}$.

Since the clusters in $\mathcal{H}$ form bottlenecks in G the simulation tries to avoid creating too much load on edges that leave or enter subsets of $\mathcal{H}$. This leads to the following basic rules for the simulation.

- Only send a message across a cut $(H_{v_i}, \overline{H}_{v_i})$ if the tree solution sends a corresponding message to intermediate node v_i.
- Distribute the messages that leave a cluster $H \in \mathcal{H}$ evenly among all edges that connect H to the rest of the graph.

Note for the first rule that if the tree solution sends a message to v_{i}, then this causes load on an edge incident to v_{i} in the tree. Such an edge has bandwidth $\mathrm{out}(H_{v_{\mathrm{i}}})$. Therefore it is possible to amortize the load created in G by sending a message over the cut $(H_{v_{\mathrm{i}}}, \overline{H}_{v_{\mathrm{i}}})$, which has capacity $\mathrm{out}(H_{v_{\mathrm{i}}})$, against the load created in $T_{\mathcal{H}}$.

Now, we give a formal description of the simulation. Each node of the decomposition tree is simulated by a random node of the corresponding cluster. The exact probability distribution depends on the type of the node and on its level in the tree. The probability that a node $u \in H_{v_{\mathrm{i}}}$ is chosen for simulating an *intermediate* tree node v_{i} on level ℓ is

$$\frac{\mathrm{w}_\ell(u)}{\mathrm{w}_\ell(H_{v_{\mathrm{i}}})} \ .$$

The probability that a *natural* level ℓ node v_{n} is simulated by $u \in H_{v_{\mathrm{n}}}$ is

$$\frac{\mathrm{w}_{\ell+1}(u)}{\mathrm{w}_{\ell+1}(H_{v_{\mathrm{n}}})} \ .$$

The intuition behind these values is as follows. For an intermediate node v_{i} the probability distribution is according to the weight of edges that leave or enter cluster $H_{v_{\mathrm{i}}}$. (Recall that $\mathrm{w}_\ell(H_{v_{\mathrm{i}}})$ counts the capacity of all edges that leave $H_{v_{\mathrm{i}}}$.) This somehow reflects the first rule. If the tree solution sends a message to v_{i}, then we can afford to send a message to the border nodes of $H_{v_{\mathrm{i}}}$, i.e., nodes of $H_{v_{\mathrm{i}}}$ that have incident edges that leave the cluster.

For a natural node v_{n} the probability distribution is with respect to edges that leave or enter subclusters of $H_{v_{\mathrm{n}}}$. This is reasonable because a message that is sent to v_{n} probably comes from a child node in the tree. Since these child nodes are embedded with respect to the weight of edges leaving the respective cluster, the distribution for v_{n} ensures that child node and father node are embedded "close to each other".

In order to complete the description of the second simulation step we have to specify how the routing between nodes in G that simulate adjacent tree nodes is performed. Let $e_{\mathrm{t}} = (v_{\mathrm{i}}, v_{\mathrm{n}})$ denote a tree edge in $T_{\mathcal{H}}$, where v_{i} is the intermediate node and v_{n} is the natural node of the edge. Further, let u_{i} and u_{n} denote the node in G that simulates v_{i} and v_{n}, respectively. Note that for the clusters corresponding to v_{i} and v_{n} we have $H_{v_{\mathrm{i}}} \subseteq H_{v_{\mathrm{n}}}$ and therefore, $u_{\mathrm{i}}, u_{\mathrm{n}} \in H_{v_{\mathrm{n}}}$.

We choose a routing path between u_{i} and u_{n} according to the solution of the CMCF-problem on cluster $H_{v_{\mathrm{n}}}$. For this we decompose the flow for commodity $d_{u_{\mathrm{i}},u_{\mathrm{n}}}$ into a convex combination of paths between u_{i} and u_{n}. (see [AMO93] for the idea of flow decomposition). This means we compute a path system $\mathcal{P}_{u_{\mathrm{i}},u_{\mathrm{n}}}$ that contains paths between u_{i} and u_{n} and assign a *weight* $\mathrm{weight}(p_j)$ to every

$p_j \in \mathcal{P}_{u_\mathrm{i},u_\mathrm{n}}$. Let $F_{u_\mathrm{i},u_\mathrm{n}}(e)$ denote the flow that passes edge e for commodity $d_{u_\mathrm{i},u_\mathrm{n}}$ in the solution of the CMCF-problem for H_{v_n}. The flow decomposition ensures that

$$\sum_{j\,:\,e\in p_j} \frac{\mathrm{weight}(p_j)}{\mathrm{weight}(\mathcal{P}_{u_\mathrm{i},u_\mathrm{n}})} = F_{u_\mathrm{i},u_\mathrm{n}}(e),$$

where weight($\mathcal{P}_{u_\mathrm{i},u_\mathrm{n}}$) denotes the sum of the weights of all paths in $\mathcal{P}_{u_\mathrm{i},u_\mathrm{n}}$.

The routing paths are chosen as follows. Whenever a message is sent along edge e_t in the tree we randomly select a path $p \in \mathcal{P}_{u_\mathrm{i},u_\mathrm{n}}$ according to its weight (i.e., $\Pr[p \text{ is chosen}] = \mathrm{weight}(p)/\mathrm{weight}(\mathcal{P}_{u_\mathrm{i},u_\mathrm{n}})$), and then we send a corresponding message in G between u_i and u_n along this path.

The following theorem captures the main result of our bisimulation framework. It shows that the expected load of an edge due to the above simulation of $T_\mathcal{H}$ on the network G is only a small factor larger than the congestion on the tree network.

Theorem 2.2 *Let C_t denote the congestion for the tree solution to problem instance I_t on $T_\mathcal{H}$. The expectation of the relative load $L(e)$ of an edge e of G due to the simulation is bounded by*

$$\mathrm{E}[L(e)] \le \delta_\mathcal{H} \cdot C_\mathrm{t} \enspace ,$$

where $\delta_\mathcal{H} := 2 \cdot \mathrm{height}(T_\mathcal{H}) \cdot c_\mathcal{H}$.

Proof. Let $L_\ell(e)$ denote the load of an edge e due to the simulation of level ℓ edges of $T_\mathcal{H}$ and let $L_\ell^{\mathrm{abs}}(e)$ denote the corresponding absolute load. We show that $\mathrm{E}[L_\ell(e)] \le 2c_\mathcal{H} C_\mathrm{t}$, which yields the lemma.

Fix a level ℓ and an edge $e = (x, y) \in E$. We analyze the expected load that is created on e due to the simulation of level ℓ edges of $T_\mathcal{H}$. A level ℓ edge of the tree connects a natural node on level ℓ to an intermediate node either on level ℓ or on level $\ell - 1$. The simulation of a level ℓ edge $(v_\mathrm{i}, v_\mathrm{n})$ only induces load on e if both endpoints x and y of e lie in the cluster H_{v_n} that corresponds to the natural node v_n of the edge. This holds because the routing paths between the nodes simulating v_i and v_n do not leave the cluster H_{v_n}, since in the definition of the CMCF-problem for H_{v_n} the flow is restricted to use only edges inside H_{v_n}.

Now, let v_n denote a natural level ℓ node such that $x, y \in H_{v_\mathrm{n}}$. (If no such node exists $\mathrm{E}[L_\ell(e)] = 0$.) Further, let V_i denote the set of intermediate nodes that are adjacent to v_n. We say that a tree edge $(v_\mathrm{i}, v_\mathrm{n}), v_\mathrm{i} \in V_\mathrm{i}$ is mapped to a pair $(u_1, u_2) \in H_{v_\mathrm{n}} \times H_{v_\mathrm{n}}$ iff v_i is simulated by u_1 and v_n is simulated by u_2. The simulation of edge $(v_\mathrm{i}, v_\mathrm{n})$ creates load on e if $(v_\mathrm{i}, v_\mathrm{n})$ is mapped to a pair (u_1, u_2) *and* e is contained in the randomly selected path between u_1 and u_2. In this case at most $C_\mathrm{t} \cdot \mathrm{b_t}(v_\mathrm{i}, v_\mathrm{n})$ messages are routed along e for simulating $(v_\mathrm{i}, v_\mathrm{n})$, since the relative load of a tree edge is at most C_t. Hence, we get the following bound for

$\mathrm{E}[L_\ell^{\mathrm{abs}}(e)]$.

$$\mathrm{E}[L_\ell^{\mathrm{abs}}(e)] \le \sum_{(u_1,u_2)} \sum_{v_\mathrm{i} \in V_\mathrm{i}} \Pr[(v_\mathrm{i}, v_\mathrm{n}) \text{ mapped to } (u_1, u_2)] \cdot \Pr[u_1\text{-}u_2 \text{ path contains } e] \cdot C_\mathrm{t}\, \mathrm{b_t}(v_\mathrm{i}, v_\mathrm{n}) \tag{2.3}$$

Let p denote the parent node of v_n in the tree, and let v_1 denote the child node such that $u_1 \in H_{v_1}$. Since the node of G that simulates a node v_i is always from the cluster H_{v_i} the probability $\Pr[(v_\mathrm{i}, v_\mathrm{n}) \text{ is mapped to } (u_1, u_2)]$ is zero if $v_\mathrm{i} \notin \{v_1, p\}$. From the definition of the decomposition tree and the simulation it follows that

$$\begin{aligned}\Pr[&(v_1, v_\mathrm{n}) \text{ is mapped to } (u_1, u_2)] \cdot \mathrm{b_t}(v_1, v_\mathrm{n}) \\ &= \frac{\mathrm{w}_{\ell+1}(u_1)}{\mathrm{w}_{\ell+1}(H_{v_1})} \frac{\mathrm{w}_{\ell+1}(u_2)}{\mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})} \cdot \mathrm{w}_{\ell+1}(H_{v_1}) \\ &= \frac{\mathrm{w}_{\ell+1}(u_1) \cdot \mathrm{w}_{\ell+1}(u_2)}{\mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})} \ ,\end{aligned}$$

and

$$\begin{aligned}\Pr[&(p, v_\mathrm{n}) \text{ is mapped to } (u_1, u_2)] \cdot \mathrm{b_t}(p, v_\mathrm{n}) \\ &= \frac{\mathrm{w}_{\ell}(u_1)}{\mathrm{w}_{\ell}(H_p)} \frac{\mathrm{w}_{\ell+1}(u_2)}{\mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})} \cdot \mathrm{w}_{\ell}(H_p) \\ &\le \frac{\mathrm{w}_{\ell+1}(u_1) \cdot \mathrm{w}_{\ell+1}(u_2)}{\mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})} \ .\end{aligned}$$

Plugging these results into Equation (2.3) gives

$$\mathrm{E}[L_\ell^{\mathrm{abs}}(e)] \le \sum_{(u_1,u_2)} \Pr[u_1\text{-}u_2 \text{ path contains } e] \cdot 2C_\mathrm{t} \cdot \frac{\mathrm{w}_{\ell+1}(u_1) \cdot \mathrm{w}_{\ell+1}(u_2)}{\mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})} \ .$$

Since the routing path between u_1 and u_2 is chosen according to the solution of the CMCF-problem for H_{v_n} we know that

$$\Pr[u_1\text{-}u_2 \text{ path contains } e] \le \frac{F_{u_1,u_2}(e)}{\mathrm{dem}(u_1, u_2)} \ ,$$

where $F_{u_1,u_2}(e)$ is the amount of flow of commodity d_{u_1,u_2} that passes e, and $\mathrm{dem}(u_1, u_2)$ is the total value of flow between u_1 and u_2 as defined in the CMCF-problem. This demand is defined as $\mathrm{dem}(u_1, u_2) = \frac{\mathrm{w}_{\ell+1}(u_1)\cdot \mathrm{w}_{\ell+1}(u_2)}{\mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})}$ (see Equation (2.2)). Hence, we get

$$\mathrm{E}[L_\ell^{\mathrm{abs}}(e)] \le 2C_\mathrm{t} \cdot \sum_{(u_1,u_2)} F_{u_1,u_2}(e) \le 2C_\mathrm{t} \cdot c_{\mathcal{H}} \cdot \mathrm{b}(e) \ ,$$

where the last step follows from the definition of $c_{\mathcal{H}}$, because $\sum_{(u_1,u_2)} F_{u_1,u_2}(e)$ is the absolute load on edge e in the solution to the CMCF-problem. Clearly, the above inequality gives $\mathrm{E}[L_\ell(e)] = 2\,c_{\mathcal{H}} \cdot \mathrm{C_t}$ for the relative load $L_\ell(e)$ of edge e. ■

2.3.5 Applications

So far, we have shown that $T_{\mathcal{H}}$ 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 adapted such that the load of any edge does not deviate too much from its expectation. This cannot be done in a problem-independent manner but the analysis has to be performed for each problem separately. In the following we show results for virtual circuit routing, multicast routing and data management. For some of these problems we further differentiate into static and dynamic settings. For every problem that we consider, we show how to obtain an efficient tree solution and how to modify the simulation such that it gives a result on the congestion.

For analyzing the latter part we use standard tail inequalities from probability theory that are known as Chernoff-Hoeffding bounds. We use the following form which is due to Hoeffding [Hoe63].

Lemma 2.3 ([Hoe63]) *Let $X_1, \ldots, X_n$ be independent random variables that take values in the range $[0, W]$ for some $W > 0$. Let $X = \sum_{i=1}^n X_i$, $\mu \geq E[X]$. Then for $\delta \geq 1$,*

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

Corollary 2.4 $\Pr[X \geq 2\mu + kW] \leq e^{-k/3}$.

Proof. We choose $\delta = 1 + \frac{W}{\mu}k$ in the above lemma. This gives $\Pr[X \geq 2\mu + kW] \leq e^{-(1+Wk/\mu)\mu/(3W)} \leq e^{-k/3}$, as desired. ■

Virtual circuit routing

In the virtual circuit routing problem the task is to select routing paths between source-target pairs in such a way that the link congestion in the network is minimized.

The tree solution for this problem is straightforward. Simply, route a request (s, t) between source node s and target node t along the unique shortest s-t path in the tree. Obviously, this strategy minimizes the congestion in the tree network. Note that in the tree instance of the routing problem there are only routing requests between leaf-nodes of the tree.

The important observation for adapting the simulation technique is, that in the routing problem the internal tree nodes do not carry any information. 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 random variable describing the load on edge $e \in E$ due to the routing of the *i-th* request σ_i. Since we use a new random embedding for each request the variables $L_i(e)$ are independent. Further, we restrict the routing algorithm to use only simple paths. (If according to the simulation scheme the routing algorithm would have to use a path that contains some nodes twice, the algorithm computes a corresponding simple path and uses this path, instead. This does not increase the expected load on any edge.) Then, the load $L(e) = \sum_i L_i(e)$ of edge e is a sum of independent random variables in the range $[0, 1/\mathrm{b}(e)]$ with expectation at most $\delta_{\mathcal{H}} \cdot \mathrm{C_t}$.

We can apply Corollary 2.4 with $\mu = \delta_{\mathcal{H}} \cdot \mathrm{C_t}$ and $W = 1/\mathrm{b}(e)$ to this sum. This gives that the load of edge e is smaller than $2\delta_{\mathcal{H}} \cdot \mathrm{C_t} + \alpha \log n/\mathrm{b}(e)$ with probability at least $1 - n^{-\alpha/3}$. Since this holds for every edge e we can apply a union bound which gives the following bound for the congestion C_{alg} of our routing algorithm.

$$C_{\mathrm{alg}} \leq 2\delta_{\mathcal{H}} \cdot C_{\mathrm{opt}} + \alpha \log n \quad \text{with probability at least } 1 - n^{-\Omega(\alpha)} \ . \qquad (2.4)$$

Here we utilized $\mathrm{C_t} \leq C_{\mathrm{opt}}$, which holds due to the first simulation, and $\mathrm{b}(e) \geq 1$, which holds due to the normalization of edge-capacities in G. We get the following lemma.

Lemma 2.5 *Given a graph G and an associated decomposition tree $T_{\mathcal{H}}$ there exists an oblivious routing algorithm that is $O(\delta_{\mathcal{H}})$-competitive with respect to the congestion.*

Note that the routing algorithm chooses each path independently from all other paths. Hence it is indeed oblivious, as stated in the lemma.

By utilizing the results of Section 2.3.6 on $\delta_{\mathcal{H}}$, the above lemma shows *the existence* of an $O(\log^3 n)$-competitive, oblivious routing algorithm for general networks. In Section 2.4 we show that such an algorithm can be constructed in polynomial time.

In [Räc02] it was erroneously claimed that the routing algorithm is strictly $O(\delta_{\mathcal{H}})$-competitive. This does not hold in general but depends on the value of the maximum bandwidth $\mathrm{b_{max}}$ of an edge of G. This can be seen as follows. Under the condition that $\alpha \geq 2$ and $\delta_{\mathcal{H}} \geq \log n$ (follows from Section 2.3.6) we can transform Equation (2.4) into

$$C_{\mathrm{alg}} \leq \alpha\, \delta_{\mathcal{H}} \cdot C_{\mathrm{opt}} \cdot \mathrm{b_{max}} + \alpha \cdot \delta_{\mathcal{H}} \quad \text{with probability at least } 1 - n^{-\Omega(\alpha)} \ .$$

Now, we can utilize that $C_{\mathrm{opt}} \cdot \mathrm{b_{max}} \geq 1$ and we get that

$$C_{\mathrm{alg}} \leq \delta_{\mathcal{H}} \cdot (\mathrm{b_{max}} + 1) \cdot C_{\mathrm{opt}} \ \text{, w.h.p.} \ ,$$

which means that the algorithm is strictly $O(\delta_{\mathcal{H}} \cdot \mathrm{b}_{\max})$-competitive. In Section 2.5 it is shown that, in general, the competitive ratio of a strictly competitive routing algorithm depends on $\mathrm{b}_{\max}$.

Note that in the above discussion we did not specify if the routing requests are given in advance or are given to the algorithm in form of a request sequence that has to be processed in an online fashion. This was not necessary since our routing algorithm is oblivious and, hence, works for online as well as for offline scenarios.

There are further interesting application of the hierarchical decomposition and the decomposition tree for static routing problems. In the fractional concurrent multicommodity flow we are given demands between pairs of vertices and the task is to establish a flow of the specified value between each pair such that the link congestion is minimized. This problem can be solved or approximated in polynomial time using, e.g., linear programming techniques (see [AMO93, GK98]).

Our algorithm solves this problem as follows. It first precomputes a flow of value 1 between every pair of nodes. This flow acts as a routing rule that describes how the demand between the corresponding node pair has to be routed. Then, when the algorithm receives the demands of the CMCF-problem it outputs a solution in time $O(|E| \cdot |D|)$, where $|E|$ is the number of edges in G with nonzero capacity, and $|D|$ is the number of demands. The congestion of this solution is only a polylogarithmic factor away from the congestion of an optimal solution. Hence, the technique makes it possible to speed up approximate solutions to CMCF-problems, if several instances have to be solved for the same network.

The sparsity of a cut in G for a routing problem is the ratio between the capacity of the cut and the total demand of all source-target pairs that are separated by the cut. A *sparsest cut* is a cut with minimum sparsity. The decomposition tree $T_{\mathcal{H}}$ of a network G can be used to find an approximate sparsest cut for a given routing problem. The algorithm works as follows. Compute the sparsity of every cluster in the hierarchical decomposition $\mathcal{H}$ and output the cluster with minimum sparsity. The following lemma shows that the returned cut is an approximate sparsest cut with approximation ratio $\delta_{\mathcal{H}}$.[3]

Lemma 2.6 *Let $\phi_{\mathcal{H}}$ denote the minimum sparsity of a cluster in $\mathcal{H}$, and let $\phi_{\min}$ denote the sparsity of a sparsest cut in G. Then*

$$\phi_{\mathcal{H}} \leq \delta_{\mathcal{H}} \cdot \phi_{\min} \quad .$$

Proof. Let H denote a cluster in $\mathcal{H}$, and let v_{n} and v_{i} denote the natural and intermediate node, respectively, corresponding to H. The sparsity of H is the capacity of edges that connect H to the rest of the graph, divided by the demand

[3]Intuitively, this fact means that the hierarchical decomposition contains all potential bottlenecks of a network. Whenever there is a bottleneck (i.e., a cut with low sparsity), one of the clusters in $\mathcal{H}$ is a bottleneck, as well.

of all messages that leave or enter H. This is exactly the inverse of the relative load of the edge $(v_{\mathrm{n}}, v_{\mathrm{i}})$ in the tree solution, because every message that leaves or enters H traverses this edge, and the capacity of the edge is $\mathrm{out}(H)$. This means that finding the cut with minimum sparsity in $\mathcal{H}$ is the same as finding the edge with maximum relative load in $T_{\mathcal{H}}$. This edge determines the congestion C_{t} of the tree solution, and therefore $\phi_{\mathcal{H}} = 1/C_{\mathrm{t}}$.

The congestion that is needed for solving the routing problem in G is at least $1/\phi_{\min}$. In Theorem 2.2 it is shown that it is possible to solve the routing problem with expected load $\mathrm{E}[L(e)] \leq \delta_{\mathcal{H}} \cdot C_{\mathrm{t}}$ on any edge $e \in E$. If we route fractionally we can achieve this bound. Therefore

$$\delta_{\mathcal{H}} \cdot C_{\mathrm{t}} \geq 1/\phi_{\min} \quad .$$

Now, $\phi_{\mathcal{H}} = 1/C_{\mathrm{t}}$ yields $\phi_{\mathcal{H}} \leq \delta_{\mathcal{H}} \cdot \phi_{\min}$, as desired. ■

The algorithm for finding the sparsest cluster in $\mathcal{H}$ can be implemented in time $O(|R| \cdot \mathrm{height}(T_{\mathcal{H}}) + n)$, where $|R|$ denotes the number of routing requests for the routing problem. This holds because each routing request increases the load on at most $2\,\mathrm{height}(T_{\mathcal{H}})$ tree edges, and the maximum loaded tree edge can be found in time $O(n)$, since there are only $O(n)$ edges.

Multicast routing

In the multicast routing problem we are given a collection of M multicast groups, each with a different source s_i, $1 \leq i \leq M$. Further we are given subscription requests that consist of pairs (i, v) with the meaning that node $v \in V$ has to be connected to the *i-th* multicast group. For such a request we say that node v *subscribes to* multicast group i. The task is to select for each group a Steiner tree that spans the source and all subscribing nodes in such a way that the congestion is minimized.

On the tree, the multicast routing problem is solved as follows. At any time the nodes that already have subscribed to a multicast group are connected via the minimum Steiner tree connecting these nodes to the source. When a tree node v issues a subscription request for the group, it sends a message in the tree towards the source of the group. At some point this message reaches a node u of the current Steiner tree. Augmenting the Steiner tree with the path from v to u gives the minimum Steiner tree that contains the source, all previous subscribers, and the new subscriber v. These steps can be performed totally distributed, since they only require local information stored at the visited tree nodes. The solution achieves optimal load on every edge of the tree and, hence, also optimal congestion. Further, this solution works for concurrent subscription requests, as well.

Now, we show how to adapt the second simulation in order to get a result on the congestion. The only information that is needed at internal tree nodes in the tree solution is whether the node already belongs to the Steiner tree of the respective multicast group. Therefore we can use a different embedding of the decomposition tree for each multicast group.

Let $L_i(e)$ denote a random variable describing the load on edge $e \in E$ due to the simulation of the *i-th* Steiner tree in $T_{\mathcal{H}}$ on G. Note that in G the subgraph that connects the subscribers of a multicast group to the source, needs not to be a tree because of the simulation. This is similar to the virtual circuit routing problem, where a path selected according to the simulation needs not to be a simple path. Therefore, we change the simulation and allow messages to take shortcuts. This means, if a message for group i that is sent to a host for tree node v_t finds that the just visited node v on its path already belongs to the tree for the *i-th* group, it does not continue on its way to v_t, but it simply connects to v forming a new legal tree for the *i-th* multicast group. This adaption of the simulation does not increase the load on any edge. Now, the load $L(e) = \sum_i L_i(e)$ of an edge is a sum of independent random variables in the range $[0, 1/\mathrm{b}(e)]$, and we can apply Corollary 2.4 exactly as for the case of virtual circuit routing. This gives the following lemma.

Lemma 2.7 *Given a graph G and an associated decomposition tree $T_{\mathcal{H}}$ there exists a distributed multicast routing algorithm that is $O(\delta_{\mathcal{H}})$-competitive with respect to the congestion. The algorithm also works for concurrent multicast requests.*

For the case of ordinary routing problems, i.e., exactly one source and one target for each request, we have shown in the previous section how to obtain very fast approximation algorithms for the static, fractional problem version when the decomposition tree $T_{\mathcal{H}}$ is known. The same works for multicast routing, too.[4] Applying our simulation technique gives an $O(\delta_{\mathcal{H}})$-approximation algorithm with running time $O(|R| \cdot |E|)$, where $|R|$ denotes the number of multicast routing requests.

Interestingly, there is a related cut problem, as well. Let for a cut $(S, \bar{S})$ and a given multicast routing problem, $\mathrm{dem}(S, \bar{S})$ denote the number of routing requests such that both sides of the cut contain at least the source or a subscribing node of the request. Define the (*multicast*)-*sparsity* of the cut $(S, \bar{S})$ as $\mathrm{cap}(S, \bar{S})/\mathrm{dem}(S, \bar{S})$. This is the straightforward extension of the sparsity of a cut to the multicast setting. The following lemma does not only show that the hierarchical decomposition $\mathcal{H}$ always contains a cluster with approximately minimum multicast sparsity, but it also relates the minimum multicast sparsity to the congestion of an optimal solution of the routing problem. Such relationships, i.e., generalizations of the max-flow min-cut theorem, are called approximate max-flow min-cut relationships, and play an important role in the design of approximation algorithms (see, e.g., [LR99]). It is well known that an approximate max-flow min-cut relation holds between the congestion for concurrent multicommodity flow problems and

[4]Fractional for the case of multicast routing means that a multicast routing request can be served by a fractional Steiner tree connecting the source to all subscribers.

sparsest cuts (see [LR99, KPR93, LLR95, AR98]). To our knowledge this is the first result that shows a polylogarithmic max-flow min-cut relation for multicast routing problems.

Lemma 2.8 *Let $\phi_{\mathcal{H}}$ denote the minimum multicast sparsity of a cluster in $\mathcal{H}$, and let $\phi_{\min}$ denote the multicast sparsity of a sparsest cut in G. Then*

$$\phi_{\min} \leq \phi_{\mathcal{H}} = 1/C_{\mathrm{t}} \leq \delta_{\mathcal{H}} \cdot 1/C_{\mathrm{opt}}(G) \leq \delta_{\mathcal{H}} \cdot \phi_{\min} \ ,$$

where C_{t} and $C_{\mathrm{opt}}(G)$ denote the optimal congestion for the routing problem on the tree network and on the physical network, respectively.

Proof. The proof is similar to the proof of Lemma 2.6. We only need to show $\phi_{\mathcal{H}} = 1/C_{\mathrm{t}}$, as the third inequality follows from $C_{\mathrm{opt}}(G) \leq \delta_{\mathcal{H}} \cdot C_{\mathrm{t}}$ (follows from Theorem 2.2), and the remaining inequalities are obvious.

Let H denote a cluster in $\mathcal{H}$, and let v_{n} and v_{i} denote the natural and intermediate node, respectively, corresponding to H. The multicast sparsity of H is the capacity of edges leaving H, divided by the number of all requests for which both H and $V \setminus H$ contain at least one node that has to be connected in the multicast. Exactly these requests create load on $(v_{\mathrm{n}}, v_{\mathrm{i}})$ in the tree solution of the routing problem. Therefore, the multicast sparsity of H equals the inverse of the relative load of the tree edge $(v_{\mathrm{n}}, v_{\mathrm{i}})$. This gives $\phi_{\mathcal{H}} = 1/C_{\mathrm{t}}$, as desired. ■

Data management

For the dynamic data management an efficient tree solution was presented in [MMVW97]. This solution is totally distributed and achieves a competitive ratio of 3.

The task of adapting the simulation technique such that it gives a result on the congestion is considerably more difficult for the data management problem than for the virtual circuit routing problem or the multicast routing problem. The reason is that in the solution to the data management problem the internal nodes of the decomposition tree store information, namely copies of shared data objects and/or pointers to other copies in the network. Therefore, it is not possible to use a new embedding of the whole decomposition tree for every request but one has to carefully define when to change the embedding of a tree node because the stored information has to be transferred to the new host. These *migration messages* increase the congestion in the network and have to be taken into account for the analysis.

We adapt the simulation technique, as follows. First of all we use a different embedding of the decomposition tree for each shared object. This is possible because the dynamic tree strategy handles each object independently from other objects.[5] Furthermore, we change the embedding of a tree node for object x, whenever too many *access messages* for object x, i.e., messages that simulate messages of the tree strategy, traverse the node. We call such a modification of the embedding a *remapping*. It is performed as follows.

For every object x, and every node $v_t \in V_t$ we introduce a counter $\tau(x, v_t)$. Initially the counter is set to zero and it is increased whenever an access message for object x starts at node v_t, arrives at v_t, or traverses v_t. The host node of v_t for object x is changed whenever the counter reaches the value D, i.e., the maximum degree of a node in the decomposition tree $T_{\mathcal{H}}$. In this case the old host node first selects a new host according to the probability distribution for node v_t. Then it sends a *migration message* to the new host that contains all relevant information for the corresponding object, as, e.g., links to other copies, or the actual data associated with the global object. Finally, in order to ensure an efficient search for data in the network, the old host establishes a pointer to the new host. Note that after several remappings, this process may create a sequence of pointers that connects several outdated hosts and leads to the current host node.

For our analysis we assume that a migration message increases the load on every edge of the traversed path by one. This assumption is justified in our uniform model, since the transmitted information mainly consists of the data content of a single global object. Therefore a migration message costs as much as a read or write message to a shared object. Note that this assumption of uniform cost for a migration message is only reasonable because we use a different embedding for every object, and therefore a migration message does not need to contain much information.

The search operation for the current position of a host node works as follows. If an access message arrives at a host node that has been remapped it follows the sequence of pointers to reach the new host. After that, it informs all outdated host nodes on the traversed path, and the origin node of the access message about the current position of the host. Note that the access message itself may cause another remapping. For this case, we define that already the new position of the host node, i.e., the position after the remapping, is reported back to the origin.

We call messages along the pointer sequence from an old host to a newer host *search messages*. We analyze the expected load generated by migration and search messages.

First, consider an intermediate node v_i on level ℓ of $T_{\mathcal{H}}$. The expected load generated by migration and search messages for v_i can be estimated as follows. Since v_i has only two incident edges both with capacity $\mathrm{out}(H_{v_i})$, the total number of access messages that reach v_i is bounded by $2 \cdot C_t \cdot \mathrm{out}(H_{v_i})$, where C_t denotes the congestion of the tree solution. The same bound holds for the number of

[5]If a request for object x would at some internal tree node require information stored for object x, and information stored for object y, it would be problematic to use different embeddings for x and y.

migration messages for v_i because each migration message is caused by some access message that traverses v_i.

The neighbor of v_i that has sent the latest access message always knows about the current position of the host. Only the other neighbor may initiate a search operation to find the new host. However, for each migration message at most one search message is sent. Therefore, the total number of search messages for v_i is bounded by $2 \cdot C_\mathrm{t} \cdot \mathrm{out}(H_{v_\mathrm{i}})$, too.

We choose the routing paths for search and migration messages for v_i as follows. Let v_n denote the child node of v_i in the tree. Due to the definition of the decomposition tree, v_n is a level ℓ node and it holds that $H_{v_\mathrm{i}} = H_{v_\mathrm{n}}$.

For a search/migration message between an old host $h \in H_{v_\mathrm{i}}$ and a new host $h' \in H_{v_\mathrm{i}}$, we choose a random intermediate node x from H_{v_i} such that $\Pr[x \text{ is chosen}] = \mathrm{w}_{\ell+1}(x)/\mathrm{w}_{\ell+1}(H_{v_\mathrm{i}})$. Then we route the message from h to x and then from x to h' using the routing paths of the CMCF-problem for cluster H_{v_i}.

Since x is chosen with the same probability distribution as the host node for v_n, we have that the expected load created by a search/migration message on some edge e is exactly twice the expected load created by an access message that traverses edge $(v_\mathrm{n}, v_\mathrm{i})$ in the tree.

In Section 2.3.4 it was shown that the expected load generated by a single access message along this edge is only $c_{\mathcal{H}}/\mathrm{out}(H_{v_\mathrm{i}})$. Consequently, the expected load created for search/migration messages is at most $8\,c_{\mathcal{H}}\,C_\mathrm{t}$, as there are only $4 \cdot C_\mathrm{t} \cdot \mathrm{out}(H_{v_\mathrm{i}})$ messages and each creates only twice the load of an access message.

Now, consider a natural node v_n on level ℓ of $T_{\mathcal{H}}$. Let a denote the number of access messages that traverse v_n during the running time of the parallel application. These messages induce at most a/D migration messages. Therefore, at most a/D *new* host nodes for v_n are generated. (The total number of host nodes for v_n can be much larger, namely as large as $|X| + a/D$, where $|X|$ denotes the number of global objects.) Each new host node h receives at most $D-1$ search messages, because after $D-1$ messages every node that simulates a neighbor of v_n in $T_{\mathcal{H}}$ knows about the migration to h. (The node that sent the access message that caused the migration to h is informed directly and will never send a search message.) Hence, the number of search/migration messages sent for node v_n is at most a. The expected number of messages sent between a pair $(x, y) \in H_{v_\mathrm{n}} \times H_{v_\mathrm{n}}$ is

$$\frac{\mathrm{w}_{\ell+1}(x)}{\mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})} \cdot \frac{\mathrm{w}_{\ell+1}(y)}{\mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})} \cdot a \ ,$$

because the function $\mathrm{w}_{\ell+1}(\cdot)/\mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})$ describes the probability that a node is the source or the destination of a search/migration message.

The total capacity of tree edges that are incident to v_n is $\mathrm{w}_{\ell+1}(H_{v_\mathrm{n}}) + \mathrm{w}_\ell(H_{v_\mathrm{n}}) \leq 2 \cdot \mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})$. Therefore, $a \leq 2\,C_\mathrm{t} \cdot \mathrm{w}_{\ell+1}(H_{v_\mathrm{n}})$, since a larger number of messages directed to v_n in the tree could not be handled with congestion C_t. This gives that the expected number of messages between x and y is at most $2\,C_t \cdot \mathrm{w}_{\ell+1}(x) \cdot \mathrm{w}_{\ell+1}(y) \,/\, \mathrm{w}_{\ell+1}(H_{v_n})$, which is $2C_t$ times the demand between x and y in the CMCF-problem for cluster H_{v_n}. Therefore, if we route the search/migration messages according to the solution of the CMCF-problem, the expected load of an edge is only $2\,c_{\mathcal{H}}\,C_t \leq 16\,c_{\mathcal{H}}\,C_t$.

We have shown that for each node v_t the search/migration messages for v_t only induce an expected load of at most $16\,c_{\mathcal{H}}\,C_t$ on an edge of H_{v_t}. On all other edges the induced load is 0. Hence, the expected load $L(e)$ of an edge e for all types of messages is only

$$L(e) \leq 18\,c_{\mathcal{H}}\,C_t\,\mathrm{height}(T_{\mathcal{H}}) = 9\,\delta_{\mathcal{H}} \cdot C_t \ ,$$

i.e., the remapping procedure only increases the bound of Theorem 2.2 by a constant factor.

Now, we derive a bound for the load of an edge that holds with high probability. For this we have to analyze the dependencies between messages sent in the adapted simulation. All messages are sent between host nodes that simulate adjacent tree nodes of $T_{\mathcal{H}}$, and the node of G that simulates a given tree node is changed from time to time. Therefore, a message is of the form: "connect the j-th host node of v_i and the k-th host node of v_n". Messages only depend on one another if they share an endpoint, i.e., they use the same embedding of some tree node. The remapping guarantees that for some fixed embedding a host node only receives $2D$ messages, namely at most D access messages and at most D search messages. Therefore, a fixed message m can only depend on at most $4D$ other messages ($2D$ messages for each endpoint). We divide the messages into $4D + 1$ classes such that the messages in any given class are independent. Let $L_i(e)$ denote the load created on edge e due to messages from class i, and let $\mu_i(e) := \mathrm{E}[L_i(e)]$ denote the expected value.

Each $L_i(e)$ is a sum of independent random variables with weight at most $1/b(e)$, because we can assume that each message is routed along a simple path. We can apply Corollary 2.4 with $\mu = \mu_i(e)$ and $W = 1/\mathrm{b}(e)$. This gives

$$L_i(e) \leq 2\mu_i(e) + \alpha \log n \quad \text{with probability at least } 1 - n^{-\Omega(\alpha)} \ .$$

As this holds for every i we get that $L(e) = \sum_i L_i(e) \leq 2\sum_i \mu_i(e) + (4D+1) \cdot \alpha \log n \leq 8\,c_{\mathcal{H}}\,\mathrm{height}(T_{\mathcal{H}})\,C_{\mathrm{opt}} + O(D \cdot \log n)$, with high probability. Since this holds for any edge e we get the following result on the congestion.

Lemma 2.9 *Given a graph G and an associated decomposition tree $T_{\mathcal{H}}$ there exists a dynamic data management algorithm that is $O(\delta_{\mathcal{H}})$-competitive with respect to the congestion.*

Concurrent requests. Now, we show how to generalize our results to scenarios where requests do not appear one by one, but have to be served concurrently. However, we do not allow arbitrary concurrent requests but we restrict the requests to form a data-race free application, i.e., a write request to an object is not allowed to overlap with other write requests to the same object, and there is some order among the requests to the same object such that for each read or write request there is a unique most recent write. This allows arbitrary concurrent accesses to different object and concurrent read accesses to the same object.

The reason for the above definition is that it allows a simple notion of *consistency*. A data management strategy is called *consistent* if it ensures that a read request directed to an object always returns the value of the most recent write access. Note that for application that are not data race free, this definition of consistency is not valid, as there may be no "most recent" write.

Of course, there exist relaxed notions of consistency for application that are not data race free. However, it might be very difficult to obtain online algorithms with an acceptable competitive ratio in such a scenario.

To see this suppose that a large number of concurrent write requests are issued in the network without an intermediate read request. An optimal strategy that has global knowledge could serve all write requests by communicating along the branches of *one* Steiner tree that connects all writing nodes and all copies. An online algorithm cannot do this because in a distributed setting it does not "know" that there is no read request, and that it thus can delay the execution of write requests for quite a while. Somehow a write request of a distributed online algorithm has to fulfill some postconditions in order to guarantee the consistency of the algorithm regardless of future requests.

We have to describe how an online strategy handles concurrent requests that are issued according to the restrictions of data race free programs, without increasing the congestion by too much. For concurrent read requests we can simply refer to the work of Maggs et al. (see [MMVW97]). Their tree strategy can handle concurrent read requests, and hence, our strategies can do so, too, since we only simulate the tree strategy in different networks.

However, it may also happen that migration messages or search messages overlap with access messages or other search messages. For example consider a search message that is sent from an old host node h_{old} to a newer host node specified by the respective pointer in the sequence. While this message is in transit other search messages or access messages may arrive at the outdated host h_{old}. In this case these messages do not initiate a second search operation but wait until the acknowledgment for the ongoing search operation returns.

This guarantees that not too many search messages are sent along the same link in the network. In general *all* messages are acknowledged in order to ensure that at most two messages for the same object are in transit between two (possibly outdated) host nodes. This mechanism can be viewed as a serialization of requests at the visited host nodes. This means if during the execution of two requests, these requests "meet" at some host node, the latter is blocked until the first has finished its operation. This method is very general and can be applied to nearly any distributed online algorithm to obtain a concurrent solution. (Note, however, that for non-distributed online algorithms as, e.g., presented in [AAF$^+$97] it is quite difficult to obtain a corresponding concurrent solution.)

We do not have to change our analysis because the only additional communication are the acknowledgments. Obviously, these messages do not influence our asymptotic bounds since each of them corresponds to a message that we take into account.

2.3.6 Constructing the hierarchical decomposition

In this section we show how to construct for any graph $G = (V, E)$ a hierarchical decomposition $\mathcal{H}$, such that the height of the corresponding decomposition tree $T_{\mathcal{H}}$ is $O(\log n)$ and in each cluster the CMCF-problem can be solved with congestion at most $c_{\mathcal{H}} = \Omega(1/\log^3 n)$. This gives a factor of $\delta_{\mathcal{H}} = O(\log^4 n)$ between the congestion in $T_{\mathcal{H}}$ and the expected load of an edge in G for the simulation (see Theorem 2.2). Furthermore, we show that the construction can be done in polynomial time with respect to $\mathrm{b}_{\max}$ and the number of nodes in the graph G.

Before we formally describe our construction algorithm we give some preliminaries on cuts and CMCF-problems. Consider any set of nodes $X \subseteq V$ and a concurrent multicommodity flow problem on X. A cut in the subgraph induced by X is a partition of X into two subsets A and $B = X \setminus A$. The sparsity of a cut (A, B) is defined as $\mathrm{cap}(A, B)/\mathrm{dem}(A, B)$, where $\mathrm{dem}(A, B)$ is the demand of the CMCF-problem that is separated by the cut, i.e., the sum over all demands of commodities for which source and destination lie in different parts of the cut.

The inverse of the sparsity of a cut in X places a lower bound on the congestion required for routing all commodities with their specified demands.[6] Therefore, it is an important problem to determine for a given routing problem a cut with minimum sparsity, i.e., a *sparsest cut*, as this is somehow the worst bottleneck in the network. There exist several approximation algorithms for this problem. We define for a given algorithm A that η_A denotes the maximum possible ratio between the sparsity of a cut computed by A and $1/C_{\mathrm{opt}}$, where C_{opt} denotes the optimum congestion that can be obtained for the CMCF-problem. Note that since the sparsity of the sparsest cut is an upper bound on $1/C_{\mathrm{opt}}$, this means, in particular, that the cut algorithm has an approximation ratio of η_A. For general graphs there exist approximation algorithms with $\eta_A = O(\log n)$ and for planar graphs there are algorithms with $\eta_A = O(1)$ (see [AR98] and [KPR93]).

[6]Note that this only holds since we assume that the flow in the solution to the CMCF-problem may not leave X.

Recapitulating, we can compute a cut (A, B) with

$$1/C_{\text{opt}} \leq \frac{\text{cap}(A, B)}{\text{dem}(A, B)} \leq \eta_A \cdot 1/C_{\text{opt}}$$

in polynomial time.

For the remainder of the section we define a parameter λ as $\lambda := 64\,\eta_A \log n$ and we choose $c_{\mathcal{H}} := 24\,\eta_A\,\lambda$. We say that a cluster H fulfills the *CMCF-property* if the solution to the CMCF-problem in H has congestion at most $c_{\mathcal{H}}$. The goal is to find a hierarchical decomposition such that every cluster fulfills the CMCF-property and height$(T_{\mathcal{H}})$ is logarithmic. Note that according to the above values $c_{\mathcal{H}}$ is $O(\log n)$ for planar networks and $O(\log^3 n)$ for general networks if the best known approximation algorithms are used for fixing η_A. Hence, if we manage to obtain a decomposition in which every cluster fulfills the CMCF-property we get a decomposition $\mathcal{H}$ with $\delta_{\mathcal{H}} = O(\log^4 n)$.

The idea of our construction algorithm is to use a recursive approach. We design an algorithm for partitioning a single cluster and apply the algorithm in a recursive manner to G, i.e., we first use the algorithm to partition V and then call it recursively for the returned subclusters of V. Such an approach is possible because the CMCF-problem for a cluster H only depends on how H is partitioned into subclusters, but *not* on the further decomposition of these subclusters. This holds because the partition of H into subcluster already completely defines the function $\mathrm{w}_{\ell+1}(\cdot)$ for subsets of H and, hence, it defines the CMCF-problem for H, as well.

Therefore if we design an algorithm that on input of a cluster H partitions H such that the CMCF-problem on H can be solved with congestion at most $c_{\mathcal{H}}$, and each subcluster contains only a constant fraction of H's nodes, we are done. Unfortunately, this is not possible because there exist input clusters that cannot be partitioned such that the CMCF-problem can be solved with low congestion. Figure 2.2 shows an example of such a cluster.

In order to overcome these difficulties we introduce an additional precondition that an input cluster given to the algorithm has to fulfill. Then we present an algorithm that partitions a cluster H fulfilling the precondition into subclusters H_i such that the CMCF-property holds for H, and every H_i fulfills the precondition. This algorithm can then be applied recursively. The precondition is as follows.

Definition 2.10 (Precondition) *A level ℓ cluster H fulfills the precondition if for all sets U, such that $|U| \leq \frac{3}{4}|H|$ the following condition holds:*

$$\lambda \cdot \text{cap}(U, H \setminus U) \geq \mathrm{w}_\ell(U) \ .$$

Observe that the bad cluster given in Figure 2.2 does not fulfill this precondition. In the proof of the following lemma it is shown that the precondition "eliminates" all input clusters that cannot be partitioned properly.

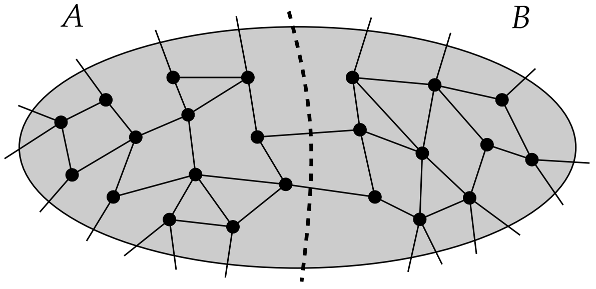

Figure 2.2: An input cluster $H = A \cup B$ that cannot be partitioned properly. Regardless of how the partitioning is done, the CMCF-problem for the cluster will send at least a flow of $\text{cap}(A, V \setminus H)/2$ or $\text{cap}(B, V \setminus H)/2$ over the indicated cut. In either case, the cut has not enough capacity to support this flow.

Lemma 2.11 (Main Lemma) *Let H be a level ℓ cluster that contains at least two vertices and fulfills the precondition. It is possible to partition H into disjoint subclusters H_i with the following characteristics:*

1. *H fulfills the CMCF-property.*
2. *For each subcluster H_i we have $|H_i| \leq \frac{2}{3} \cdot |H|$.*
3. *Each subcluster H_i fulfills the precondition.*

Moreover this partitioning can be computed in polynomial time with respect to $|H|$ and $\mathrm{b}_{\max}$, where $\mathrm{b}_{\max}$ denotes the maximum capacity of a network link.

Now[7], we argue that the described algorithm yields the construction of the hierarchical decomposition $\mathcal{H}$. This can be seen as follows. First we apply the algorithm to the set V which is the only cluster on level 0 of the decomposition tree. V fulfills the precondition, because $\mathrm{w}_0(V) = 0$. The algorithm returns a partitioning of V that defines the function $w_1(\cdot)$ and yields the level 1 clusters which fulfill the precondition. We apply the algorithm recursively to all these clusters until we get singleton sets $\{v\}, v \in V$. By Property 1 our algorithm ensures that for each cluster the corresponding CMCF-problem can be solved with low congestion. Further, the height of the decomposition tree $T_{\mathcal{H}}$ is logarithmic because of Property 2 of the lemma. The total number of clusters in the hierarchy $\mathcal{H}$ is at most $O(n)$. Therefore the total construction time is also polynomial with respect to $\mathrm{b}_{\max}$ and n.

Before proving Lemma 2.11 we introduce an important subroutine used in the partitioning algorithm. During the algorithm it may happen that some of the current clusters do not fulfill the precondition. Then a subroutine called

[7]The proof of the main lemma can be found on page 43.

```
AssurePrecondition ($R$)
  $\mathcal{P}_R := \{R\}$
  do
    for each $R_i \in \mathcal{P}_R$ do
      compute $(A, B)$ - an approximate sparsest cut for $\mathcal{G}(R_i)$
      $\psi$ := sparsity of the cut $(A, B)$
      if $\psi \leq \frac{4\sigma}{\lambda}$ then
        $\mathcal{P}_R := \mathcal{P}_R \setminus \{R_i\}$
        $\mathcal{P}_R := \mathcal{P}_R \cup \{A, B\}$
  until we made no changes to $\mathcal{P}_R$ in this iteration
  return $\mathcal{P}_R$
```

Figure 2.3: The algorithm AssurePrecondition

AssurePrecondition() splits such a cluster into smaller parts such that each part fulfills the precondition.[8] For maintaining some global property of the partitioning algorithm the capacity of edges connecting the created parts must not be too large. This is formalized in the following lemma.

Lemma 2.12 *It is possible to partition any set $R \subseteq V$ into disjoint sets R_i, such that each R_i fulfills the precondition and $\sum_i \mathrm{out}(R_i) \leq 2\mathrm{out}(R)$. Moreover, this partitioning can be computed in polynomial time with respect to $|R|$.*

Proof. The algorithm AssurePrecondition() is described in Figure 2.3. It works as follows. We start with a partition that contains only R. In each iteration we consider each set R_i of the current partitioning $\mathcal{P}_R$. We define a concurrent multicommodity flow problem $\mathcal{G}$ with demands $\mathrm{dem}(u, v) = \mathrm{w}_\ell(u)/|R_i|$ for each ordered pair $u, v \in R_i$. Then we compute (A, B) – an approximate sparsest cut of R_i. Let ϕ denote the sparsity of this cut, i.e., $\phi = \mathrm{cap}(A, B)/\left(\frac{|B|}{|R_i|} \cdot \mathrm{w}_\ell(A) + \frac{|A|}{|R_i|} \cdot \mathrm{w}_\ell(B)\right)$. If $\phi \leq 4\eta_A/\lambda$, then R_i is replaced by A and B in the current partitioning $\mathcal{P}_R$. We proceed until the sparsity of the computed approximate cut for each R_i is greater than $4\eta_A/\lambda$. For simpler notation we denote the term $4\eta_A/\lambda$ with Λ in the following.

The algorithm runs in polynomial time because the number of iterations is bounded by $|R|$ and each iteration runs in polynomial time.

[8]Note that this is always possible because clusters that contain only a single node fulfill the precondition.

First we prove that after this algorithm has finished, each set R_i from the partitioning of R fulfills the precondition. Assume for contradiction that there exists a set R_i and $U \subseteq R_i$ such that $|U| \le \frac{3}{4}|R_i|$ and $\lambda \cdot \mathrm{cap}(U, R_i \setminus U) < w_\ell(U)$. Let $\phi_{\min}$ denote the sparsity of a sparsest cut for $\mathcal{G}(R_i)$. We derive a bound on $\phi_{\min}$ and thus also on the sparsity of the approximate sparsest cut ϕ computed by the algorithm. It holds that

$$\begin{aligned}\lambda \cdot \mathrm{cap}(U, R_i \setminus U) &< w_\ell(U)\\ &\le 4\frac{|R_i \setminus U|}{|R_i|} \cdot w_\ell(U)\\ &\le 4\left(\frac{|R_i \setminus U|}{|R_i|} w_\ell(U) + \frac{|U|}{|R_i|} w_\ell(R_i \setminus U)\right) ,\end{aligned}$$

where we utilized that $|U| \le \frac{3}{4}|R_i|$ for the second step. This gives that the sparsity of the cut $(U, R \setminus U)$ and, hence, also the value of the sparsest cut is at most $4/\lambda$. Since ϕ is the value of an approximate sparsest cut we get

$$\phi \le \eta_A \cdot \phi_{\min} \le \eta_A \frac{4}{\lambda} = \Lambda ,$$

which is a contradiction, because in this case the algorithm would have divided R_i. Consequently, each set $R_i \in \mathcal{P}_R$ fulfills the precondition.

To prove that $\sum_i \mathrm{out}(R_i) \le 2\mathrm{out}(R)$ we consider a directed weighted graph H with node set V_H whose vertices correspond to edges of G leaving a partition R_i in the final partitioning $\mathcal{P}_R$. For simpler notation, let $R_H \subseteq V_H$ denote the set of nodes of H that represent edges which have exactly one endpoint in R, i.e., edges that contribute to $\mathrm{out}(R)$.

The edges of H will model the fact that newly introduced capacity is amortized against already existing capacity. In the following, we define the edges of H more precisely. Consider a step of the algorithm in which a set R_i is divided into sets A and B. Such a step increases the capacity of edges that leave partitions of $\mathcal{P}_R$ by the capacity of edges between A and B, i.e., $2\mathrm{cap}(A, B)$. (Each edge is counted twice since it leaves two partitions of $\mathcal{P}_R$.) For each such edge we introduce a new vertex in H. We want to derive a bound on the total capacity that is added to H. Therefore we amortize the newly created capacity $2\mathrm{cap}(A, B)$ against the capacity $\mathrm{out}(R_i)$.

Let $E_A = A \times \overline{R_i}$ and $E_B = B \times \overline{R_i}$ denote the set of edges that have one endpoint outside R_i and the other in A and B, respectively. ($E_A \cup E_B$ contains all edges that leave the set R_i.) In order do describe our amortization scheme we introduce the following notion. We say that the edges from $E_A \cup E_B$ *pay* for the new edges from $A \times B$. We define for each pair of edges $e \in A \times B$ and $e' \in E_A \cup E_B$ a price $\mathrm{pay}(e', e)$

that describes the amount that is paid by edge e' for edge e. We require that for each edge $e \in A \times B$ it holds that

$$\sum_{e' \in E_A \cup E_B} \mathrm{pay}(e', e) \geq 2\mathrm{b}(e) \ ,$$

i.e., we pay enough for edge e.

We model this payment in the graph H via a directed edge from $v_{e'}$ to v_e that is weighted with $\mathrm{pay}(e', e)$. Then the above requirement simply states that for each node $v_e \in V_H \setminus R_H$ (a node that is added to H during the running time of AssurePrecondition), the weight of incoming edges must be at least as large as two times the weight of v_e, i.e., $\mathrm{b}(e)$.

The exact definition of the function $\mathrm{pay}(\cdot, \cdot)$ is as follows. For an edge $e_a \in E_A$ we define

$$\mathrm{pay}(e_a, e) := 2\Lambda \cdot \frac{\mathrm{b}(e)}{\mathrm{cap}(A, B)} \cdot \frac{|B|}{|R_i|} \cdot \mathrm{b}(e_a),$$

and for an edge $e_b \in E_B$ we define

$$\mathrm{pay}(e_b, e) := 2\Lambda \cdot \frac{\mathrm{b}(e)}{\mathrm{cap}(A, B)} \cdot \frac{|A|}{|R_i|} \cdot \mathrm{b}(e_b) \ .$$

In order to simplify our notation we extend the function $\mathrm{pay}(\cdot, \cdot)$ to vertices of H, i.e., for two vertices $v_e, v_{e'} \in V_H$ that correspond to edge $e, e' \in V \times V$ we define $\mathrm{pay}(v_{e'}, v_e) = \mathrm{pay}(e', e)$ which describes the weight of edge $(v_{e'}, v_e) \in E_H$.

The following claim shows that by the above definition we pay enough for new edges.

Claim 2.13 $\forall v_e \in V_H \setminus R_H : \ \sum_{v \in V_H} \mathrm{pay}(v, v_e) \geq 2\mathrm{b}(e)$.

Proof. Let e denote an edge that is created when partitioning a set R_i into A and B. We can estimate the payment for incoming edges of v_e by

$$\begin{aligned}
\sum_{v \in V_H} \mathrm{pay}(v, v_e) &= \sum_{e' \in E_A \cup E_B} \mathrm{pay}(v_{e'}, v_e) \\
&= \sum_{e_a \in E_A} \mathrm{pay}(v_{e_a}, v_e) + \sum_{e_b \in E_B} \mathrm{pay}(v_{e_b}, v_e) \\
&= \sum_{e_a \in E_A} 2\Lambda \cdot \frac{|B|}{|R_i|} \cdot \frac{\mathrm{b}(e)}{\mathrm{cap}(A, B)} \cdot \mathrm{b}(e_a) + \sum_{e_b \in E_B} 2\Lambda \cdot \frac{|A|}{|R_i|} \cdot \frac{\mathrm{b}(e)}{\mathrm{cap}(A, B)} \cdot \mathrm{b}(e_b) \\
&= 2\Lambda \cdot \frac{\mathrm{b}(e)}{\mathrm{cap}(A, B)} \cdot \left(\frac{|B|}{|R_i|} \cdot \mathrm{w}_\ell(A) + \frac{|A|}{|R_i|} \cdot \mathrm{w}_\ell(B) \right) \\
&\geq 2\mathrm{b}(e) \ ,
\end{aligned}$$

where the last step follows from the fact that (A, B) is a cut with sparsity at most Λ, which implies that $\Lambda \cdot \left(\frac{|B|}{|R_i|} \cdot \mathrm{w}_\ell(A) + \frac{|A|}{|R_i|} \cdot \mathrm{w}_\ell(B)\right) \geq \mathrm{cap}(A, B)$. ■

The following claim relates the total weight of edges leaving node $v_e \in V_H$ to $\mathrm{b}(e)$.

Claim 2.14 *The total payment of an edge e during the whole algorithm is at most* $\mathrm{b}(e)/2$*, i.e.,*

$$\sum_{v \in V_H} \mathrm{pay}(v_e, v) \leq \mathrm{b}(e)/2 \ .$$

Proof. Let $e = (v, u)$. We consider sequences of different sets in which v and u lie during the run of the algorithm. We denote these sequences of sets as $V_0, V_1, \dots, V_l$ and $U_0, U_1, \dots, U_k$ for v and u, respectively. For those sequences, let $n_i = |V_i|$ and $m_j = |U_j|$.

Each edge (v_e, v) of H corresponds to cutting some V_i into V_{i+1} and $V_i \setminus V_{i+1}$ or U_j into U_{j+1} and $U_j \setminus U_{j+1}$. First we consider the case in which V_i is cut into $A := V_{i+1}$ and $B := V_i \setminus V_{i+1}$. We can estimate the total weight of edges between v_e and nodes of H that represent edges from $A \times B$ as follows:

$$\begin{aligned}
\sum_{e' \in A \times B} \mathrm{pay}(v_e, v_{e'}) &= \sum_{e' \in A \times B} 2\Lambda \cdot \frac{|B|}{|V_i|} \cdot \frac{\mathrm{b}(e')}{\mathrm{cap}(A, B)} \cdot \mathrm{b}(e) \\
&= 2\Lambda \cdot \mathrm{b}(e) \cdot \frac{|B|}{|V_i|} \cdot \frac{\sum_{e' \in A \times B} \mathrm{b}(e')}{\mathrm{cap}(A, B)} \\
&= 2\Lambda \cdot \mathrm{b}(e) \cdot \frac{|B|}{|V_i|} \\
&= 2\Lambda \cdot \mathrm{b}(e) \cdot \frac{n_i - n_{i+1}}{n_i} \ .
\end{aligned}$$

An analogous bound can be proved for the case when we divide U_j into U_{j+1} and $U_j \setminus U_{j+1}$, namely $\sum_{e' \in A \times B} \mathrm{pay}(v_e, v_{e'}) = 2\Lambda \cdot \mathrm{b}(e) \cdot \frac{m_j - m_{j+1}}{m_j}$.

Proposition 2.15 *For any sequence of natural numbers* $n = n_0 > \ldots > n_k \geq 1$ *it holds that* $\frac{n_0 - n_1}{n_0} + \frac{n_1 - n_2}{n_1} + \cdots + \frac{n_{k-1} - n_k}{n_{k-1}} \leq 2 \log n$.

With this proposition we can estimate the total weight outgoing from v_e as

$$\begin{aligned}
\sum_{v \in V_H} \mathrm{pay}(v_e, v) &= 2\Lambda \cdot \mathrm{b}(e) \cdot \left[\left(\tfrac{n_0 - n_1}{n_0} + \cdots + \tfrac{n_{k-1} - n_k}{n_{k-1}}\right) + \left(\tfrac{m_0 - m_1}{m_0} + \cdots + \tfrac{m_{l-1} - m_l}{m_{l-1}}\right)\right] \\
&\leq 2\Lambda \cdot \mathrm{b}(e) \cdot (2 \log n + 2 \log n) \\
&\leq 8 \cdot \frac{4\eta_A}{\lambda} \cdot \log n \cdot \mathrm{b}(e) \\
&\leq \mathrm{b}(e)/2 \ .
\end{aligned}$$

This completes the proof of Claim 2.14. ■

Now, we can use the above claims to show that $\sum_i \mathrm{out}(R_i) \leq 2\mathrm{out}(R)$. This is done by summing the weight of all incident edges for each node $v_e \in H$, where incoming edges are counted positively and outgoing edges are counted negatively. Recall that $R_H \subseteq H$ denotes the set of nodes of H that represent edges leaving set R in the graph G. We get

$$\begin{aligned} 0 &= \sum_{v_e \in V_H} \left(\sum_{v \in V_H} \mathrm{pay}(v, v_e) - \sum_{v \in V_H} \mathrm{pay}(v_e, v) \right) \\ &= \sum_{v_e \in V_H \setminus R_H} \left(\sum_{v \in V_H} \mathrm{pay}(v, v_e) - \sum_{v \in V_H} \mathrm{pay}(v_e, v) \right) - \sum_{v_e \in R_H} \sum_{v \in V_H} \mathrm{pay}(v_e, v) \\ &\geq \sum_{v_e \in V_H \setminus R_H} \left(2\mathrm{b}(e) - \frac{1}{2}\mathrm{b}(e) \right) - \sum_{v_e \in R_H} \frac{1}{2}\mathrm{b}(e) \ . \end{aligned}$$

This gives $\mathrm{out}(R) \geq 3 \sum_{v_e \in V_H \setminus R_H} \mathrm{b}(e)$. Altogether we get

$$\begin{aligned} 2\,\mathrm{out}(R) &\geq \mathrm{out}(R) + 3 \sum_{v_e \in V_H \setminus R_H} \mathrm{b}(e) \\ &\geq \mathrm{out}(R) + 2 \sum_{v_e \in V_H \setminus R_H} \mathrm{b}(e) \\ &= \sum_i \mathrm{out}(R_i) \ , \end{aligned}$$

as desired. ■

Now, we prove the main lemma.

Lemma 2.11 (Main Lemma) *Let H be a level ℓ cluster that contains at least two vertices and fulfills the precondition. It is possible to partition H into disjoint subclusters H_i with the following characteristics:*

1. *H fulfills the CMCF-property.*
2. *For each subcluster H_i we have $|H_i| \leq \frac{2}{3} \cdot |H|$.*
3. *Each subcluster H_i fulfills the precondition.*

Moreover this partitioning can be computed in polynomial time with respect to $|H|$ and $\mathrm{b}_{\max}$, where $\mathrm{b}_{\max}$ denotes the maximum capacity of a network link.

```
PARTITION ($S$)
    $\mathcal{P}_H := \{\{v\} \mid v \in H\}$
    while $H$ does not fulfill the throughput property
    do
        compute $(A, B)$ - an approx. sparsest cut of $H$ with $|A| \leq |B|$
        $U^* :=$ round($A$)
        for each $H_i \subseteq U^*$ do
            $\mathcal{P}_H := \mathcal{P}_H \setminus H_i$
            $\mathcal{P}_H := \mathcal{P}_H \cup$ ASSUREPRECONDITION($U^*$)
        end
    end
    return $\mathcal{P}_H$
```

Figure 2.4: The algorithm PARTITION

Proof. The algorithm PARTITION is shown in Figure 2.4 and works as follows. It always maintains a partitioning $\mathcal{P}_H$ that fulfills requirements 2 and 3, i.e., each subcluster H_i contains at most $\frac{2}{3}|H|$ nodes and fulfills the precondition. The initial partitioning consists of subclusters containing only one node. Clearly this partitioning fulfills both requirements.

Now, in a kind of local search approach the algorithm successively changes the partitioning until requirement 1 is fulfilled. If this happens the algorithm has found the desired partitioning and can terminate. We show that if the algorithm has not yet found a suitable partitioning it can change the current partitioning such that a global work function decreases by a certain value. Since this work function is bounded from below, we can deduce that the algorithm terminates in a polynomial number of iterations and outputs a good partitioning.

The details are as follows. In each iteration the algorithm checks whether the CMCF-problem that corresponds to the current partitioning can be solved with congestion $c_{\mathcal{H}}$. If this is the case, the algorithm terminates, since all the requirements are fulfilled and a good partitioning is found.

Otherwise the algorithm tries to find a collection of subclusters H_i of the current partitioning that has a certain property, namely that $\text{out}(\biguplus_{i \in I} H_i) \ll \text{w}_{\ell+1}(\biguplus_{i \in I} H_i)$, where I is the index set of the collection of these subclusters.

Then the algorithm merges theses subclusters together to form a new single subcluster, i.e., it removes all the subclusters that belong to the collection and adds a new single subcluster $U^* := \biguplus_{i \in I} H_i$ that simply contains all nodes from the collection. Then in a final step the algorithm partitions U^* with AssurePrecondition in order to ensure that every subcluster of the partitioning fulfills the precondition.

A replacement of all H_i by clusters that result from AssurePrecondition(U^*) is called a *local improvement* of the algorithm. A key result for the construction is that we show that in each local improvement step the total capacity of edges that connect different subclusters decreases at least by a constant. Since this capacity is clearly bounded from below by 0, the algorithm will terminate after at most $|H|^2 \cdot \mathrm{b}_{\max}$ iterations, i.e., in polynomial time. Furthermore, since we show that the algorithm always makes an improvement step if H does not fulfill the CMCF-property, we can conclude that after termination the CMCF-problem on H can be solved with congestion at most $c_{\mathcal{H}}$.

Now we describe how the algorithm finds a collection of subclusters such that $\mathrm{out}(\biguplus_{i\in I} H_i) \ll \mathrm{w}_{\ell+1}(\biguplus_{i\in I} H_i)$ if H does not fulfill the CMCF-property. First we compute an approximate sparsest cut (A, B) corresponding to the CMCF-problem in H. Without loss of generality we can assume that $|A| \leq |B|$. For this cut we have

$$\frac{\mathrm{cap}(A,B)}{\mathrm{dem}(A,B)} \leq \eta_A \cdot 1/C_{\mathrm{opt}} \leq \eta_A \cdot 1/c_{\mathcal{H}} = 1/(24\lambda) \ ,$$

because the optimal congestion C_{opt} for solving the CMCF-problem is larger than $c_{\mathcal{H}}$. The demand $\mathrm{dem}(A, B)$ can be estimated as

$$\begin{aligned}
\mathrm{dem}(A,B) &= \sum_{u\in A, v\in B} \mathrm{dem}(u,v) + \sum_{u\in B, v\in A} \mathrm{dem}(u,v) \\
&= \sum_{u\in A, v\in B} \frac{\mathrm{w}_{\ell+1}(u)\cdot w_{\ell+1}(v)}{w_{\ell+1}(H)} + \sum_{u\in B, v\in A} \frac{\mathrm{w}_{\ell+1}(u)\cdot w_{\ell+1}(v)}{w_{\ell+1}(H)} \\
&= 2 \cdot \frac{\mathrm{w}_{\ell+1}(A)\cdot w_{\ell+1}(B)}{w_{\ell+1}(H)} \\
&\leq 2 \cdot \mathrm{w}_{\ell+1}(A) \ .
\end{aligned}$$

Combining the sparsity of cut (A, B) with the above inequality we get

$$\frac{\mathrm{cap}(A,B)}{\mathrm{w}_{\ell+1}(A)} \leq 2 \cdot \frac{\mathrm{cap}(A,B)}{\mathrm{dem}(A,B)} \leq \frac{1}{12\lambda} \ . \tag{2.5}$$

We cannot directly use the set A to improve the current partitioning of H, because it does not have to consist of whole subclusters H_i. Therefore we define a set U^*, that is a rounding of the set A using the current partitioning, i.e., U^* is a union of disjoint subclusters H_i. More precisely, let $A_i := A \cap H_i$ and $B_i := B \cap H_i$.

We partition all indices of subclusters into sets I_L and I_S. If $|A_i| \geq \frac{3}{4} \cdot |H_i|$ then we say that H_i has *large intersection* with A and we define $i \in I_L$. Otherwise the index i belongs to I_S. U^* is a union of all subclusters H_i that have large intersection with A, i.e., $U^* := \biguplus_{i \in I_L} H_i$. This definition and the fact that $|A| \leq \frac{1}{2}|H|$ ensure that $|U^*| \leq \frac{2}{3}|H|$. The following technical claim is proved in the appendix on page 97.

Claim 2.16 $\dfrac{\mathrm{out}(U^*)}{w_{\ell+1}(U^*)} \leq 4\lambda \cdot \dfrac{\mathrm{cap}(A,B)}{w_{\ell+1}(A)}$.

Using this claim and Equation (2.5) we get $\mathrm{w}_{\ell+1}(U^*) \geq 3\,\mathrm{out}(U^*)$. We are now able to prove that the algorithm Partition terminates.

Lemma 2.17 *The algorithm* Partition *terminates and its running time is polynomially bounded with respect to* $\mathrm{b_{max}}$ *and* $|H|$.

Proof. Let $W(H) := \mathrm{w}_{\ell+1}(H) - \mathrm{out}(H)$ denote the total capacity of edges that connect different subclusters in H. For proving the lemma it suffices to show that in each iteration of the Partition algorithm, $W(H)$ decreases by at least 1.

In each round we remove all the subclusters contained in U^*. Therefore, $W(H)$ decreases by $\mathrm{w}_{\ell+1}(U^*)$, i.e., by at least $3\,\mathrm{out}(U^*)$. After that we add clusters R_i returned by AssurePrecondition(U^*) and $W(H)$ increases by $\sum_i \mathrm{w}_{\ell+1}(R_i)$. Using Lemma 2.12 we obtain

$$\sum_i w_{\ell+1}(R_i) = \sum_i \mathrm{out}(R_i) \leq 2\,\mathrm{out}(U^*) \ .$$

Thus in each iteration $W(H)$ decreases by at least $\mathrm{out}(U^*)$. Since the capacity function $\mathrm{b}(\cdot)$ is normalized, $\mathrm{out}(U^*) \geq 1$.

At the beginning of Partition, $W(H)$ is the sum of bandwidths of all edges that have both endpoints in the cluster H. Hence, initially $W(H) \leq |H|^2 \cdot \mathrm{b_{max}}$. Since $W(H)$ decreases in each iteration by at least 1 and is bounded from below by 0, Partition must terminate after at most $|H|^2 \cdot \mathrm{b_{max}}$ iterations. This yields the lemma. ■

This finishes the proof of the main lemma. ■

Theorem 2.18 *There is a polynomial time algorithm that constructs a hierarchical decomposition $\mathcal{H}$ that guarantees a factor of $\delta_{\mathcal{H}} = O(\log^2 n \cdot \eta_A^2)$ between the congestion in $T_{\mathcal{H}}$ and the expected load of an edge in G.*

Proof. Lemma 2.11 shows that a hierarchical decomposition with $\mathrm{height}(T_{\mathcal{H}}) = O(\log n)$ and $c_{\mathcal{H}} = O(\eta_A^2 \cdot \log n)$ can be constructed in polynomial time. Since $\delta_{\mathcal{H}} = O(c_{\mathcal{H}} \cdot \mathrm{height}(T_{\mathcal{H}}))$ the theorem follows. ■

Remark 2.19 *If in the algorithm* AssurePrecondition *the subroutine for approximating a sparsest cut is replaced by an exact algorithm, the hierarchical decomposition has a factor of* $\delta_{\mathcal{H}} = O(\log^2 n \cdot \eta_A)$.

Proof. If a cut in the AssurePrecondition algorithm is computed optimally the precondition holds with $\lambda = O(\log n)$ for every cluster. Therefore, we could compute a partitioning for which the congestion of the CMCF-problem for a cluster is at most $c_{\mathcal{H}} = 24 \cdot \eta_A \cdot \lambda = O(\eta_A \cdot \log n)$. Since the height of the decomposition tree is logarithmic, we obtain a factor $\delta_{\mathcal{H}} = O(\log^2 n \cdot \eta_A)$. ■

Note that this version of the construction algorithm is not polynomial since computing a sparsest cut is NP-hard. But this modification of our construction algorithm proves, e.g., the existence of an oblivious routing scheme with competitive ratio $O(\log^3 n)$, for general networks. In Section 2.4 we show how to compute the optimal oblivious routing scheme for any network. However, the latter result *does not* show any bound on the competitive ratio for this optimal scheme. Therefore, an existence proof of an $O(\log^3 n)$ competitive algorithm is important because such an algorithm can be constructed in polynomial time with completely different techniques that do not give a bound on the competitive ratio.[9]

A simple partitioning algorithm for meshes

For specific network topologies it is not necessary to apply the complex and time consuming partitioning algorithm presented in the previous section. For meshes, e.g., there is the following much simpler approach. Consider an N-ary d-cube, i.e., a d-dimensional mesh with sidelength N in each dimension, where N is a power of two, i.e., $N := 2^k$, and that has uniform edge capacities. The partitioning divides such a cube into 2^d subcubes of sidelength $N/2 = 2^{k-1}$ by bisecting the cube in each dimension. Since this partition gives $N/2$-ary subcubes it can be applied recursively for $\log N$ steps. Therefore, the height of the corresponding decomposition tree is $\log N = \log(\sqrt[d]{n}) = 1/d \cdot \log n$. We show that this simple scheme achieves a good bound on the congestion needed for solving the CMCF-problem in a cluster of the resulting hierarchical partitioning $\mathcal{H}$.

Lemma 2.20 *For a d-dimensional cube* $M(d,N)$ *with sidelength* $N = 2^k$ *and* $n = N^d$ *nodes, there is a hierarchical partitioning such that* $\text{height}(T_{\mathcal{H}}) = 1/d \cdot \log n$ *and* $c_{\mathcal{H}} = O(d)$. *This gives a factor of* $\delta_{\mathcal{H}} = O(\log n)$ *between the congestion in* $T_{\mathcal{H}}$ *and the expected load on edges of* $M(d,N)$ *for the simulation.*

[9]Note, however, that this result has been improved by Harrelson et al. [HHR03]. They show how to construct a hierarchical decomposition with factor $\delta_{\mathcal{H}} = O(\eta_A \log n \log\log n)$ in polynomial time.

Proof. We only have to show that $c_{\mathcal{H}} = O(d)$ since the bound on height$(T_{\mathcal{H}})$ directly follows from the partitioning scheme described above. We use known results from routing on meshes.

Theorem 2.21 *On a cube of sidelength N, an all-to-all routing problem can be solved with congestion $O(N^{d+1})$.*

Proof. Baumslag and Annexstein [BA91] have shown how to route any permutation on $M(d, N)$ in $O(d \cdot N)$ steps with congestion $O(N)$. Since an all-to-all routing problem can be decomposed into $N^d - 1$ permutation routing problems, the theorem follows. ∎

For showing that every CMCF-problem can be solved with congestion $O(d)$ we translate the CMCF-problem into an all-to-all routing problem so that we can apply the above theorem.

The CMCF-problem for a cluster $H \in \mathcal{H}$ defines that every node that has m edges leaving a subcluster of the partitioning of H has to distribute m units of flow among the border nodes, i.e., among nodes that have incident edges leaving a subcluster. (Note that we only need to consider the number of edges, since the edge capacities are uniform for mesh networks.) An analogous notion of this CMCF-problem is to view it as if the flow originates at the border edges (edges leaving a subcluster), and that each border edge has to distribute one unit of flow among all other border edges.

We first distribute this flow, among all nodes of the network. For this, each edge distributes its flow evenly among all nodes in the corresponding linear array. This can be done with constant congestion because a single divide step cuts each linear array at most once. Therefore a linear array contains at most four border edges.

A node receives at most flow of value $4d/N$, as it belongs to d linear arrays, and each array contributes at most flow $4/N$. Note that each node gets exactly the same amount of flow, since all arrays along the same dimension have the same number of border edges, because of the symmetry of the partitioning.

Now, each node distributes its flow among all other nodes. This can be done using an all-to-all routing in which each node sends a fraction of $1/N^d$ of its flow to every other node. According to Theorem 2.21 we need only congestion $O(N^{d+1}) \cdot 4d/N \cdot 1/N^d = O(4d)$ for this step.

In the last step each array distributes its flow evenly among its border edges. Thereby, each edge receives the right amount of flow. Obviously, this step can be performed with constant congestion.

Altogether we have shown that we can route the CMCF-problem with congestion $O(d)$. This yields the lemma. ∎

Note that the above theorem gives an $O(\log n)$-competitive algorithm for virtual circuit routing and data management on meshes of arbitrary dimension. This improves the result of [MMVW97], where a competitive ratio of $O(d \cdot \log n)$ was shown.

2.4 Optimal oblivious routing

In the previous sections we have presented a basic framework for minimizing congestion in distributed systems, and we have shown that this framework gives, e.g., an oblivious virtual circuit routing algorithm with polylogarithmic competitive ratio for general, undirected graphs. Since this framework provides a uniform bound on all graphs it may not generate the best routing scheme for a network but only one that obtains the polylogarithmic bound. A better oblivious routing scheme may potentially exist, i.e., one that guarantees a smaller competitive ratio w.r.t. congestion.

In this section we show how to compute the best possible oblivious routing scheme for any given network in polynomial time. The techniques used to derive this result are based on linear programming and differ completely from techniques used in the previous section. In particular, the result in this section does not provide general bounds on the competitive ratio of oblivious routing. It is only shown that it is possible to compute the optimum bound for a given network in polynomial time.

2.4.1 Preliminaries

We use the same notation for the network as in Section 2.3. This means the network is modeled as a complete graph $G = (V, E)$ with $n = |V|$ nodes, and the function $\mathrm{b} : E \to \mathbb{R}_0^+$ describes link capacities. Note, however, that in this section we deal with directed graphs, this means the bandwidth function b is not symmetric as it was in the previous section.

Our results can be transferred to undirected networks, as well, because there is the following, well known reduction from undirected to directed graphs. Replace each undirected edge $e = (u, v)$, of capacity $\mathrm{b}(e)$, with the directed gadget u, x, y, v which consists of five directed edges: four directed edges $e_1 = (u, x), e_2 = (v, x), e_3 = (y, u), e_4 = (y, v)$, with capacity ∞, and one directed edge $e_5 = (x, y)$ with capacity $\mathrm{b}(e)$. The gadget is illustrated in Figure 2.5. The transformation preserves the property that a multicommodity flow is feasible on the undirected graph if and only if it is feasible on the directed graph. Therefore, all results that we develop for directed graphs in this section hold for undirected graphs, as well.

An oblivious routing scheme specifies for every source target pair s, t a unit flow from s to t that defines how the demand between s and t is routed. Formally

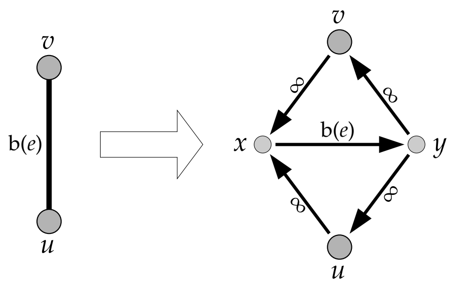

Figure 2.5: The reduction from undirected to directed graphs

a *routing scheme* or *a routing* f is defined by the following linear constraints

$$\begin{array}{lrcl} \forall e \in E \ \ \forall s,t \in V & f_{st}(e) & \geq & 0 \\ \forall v \in V \ \forall s,t \in V \setminus \{v\} & \sum_{e \in \text{OUT}(v)} f_{st}(e) - \sum_{e \in \text{IN}(v)} f_{st}(e) & = & 0 \\ \forall s,t \in V & \sum_{e \in \text{OUT}(s)} f_{st}(e) - \sum_{e \in \text{IN}(s)} f_{st}(e) & = & 1 \end{array} \tag{2.6}$$

where $\textsc{out}(v)$ and $\textsc{in}(v)$ denote the set of edges leaving and entering v, respectively. Note that each set of variables $f_{st}(e)$, $e \in E$ defines a unit, single-commodity flow from s to t.

A *demand matrix* $\boldsymbol{D}$ is an $n \times n$ nonnegative matrix, in which the element D_{st} defines the amount of flow that has to be sent from s to t. We can route this demand using a routing scheme $\boldsymbol{f}$ by simply scaling each unit flow f_{st} of the routing scheme by a factor of D_{st}. The *load* created on edge e when routing the demand matrix $\boldsymbol{D}$ with routing scheme $\boldsymbol{f}$ is

$$\textsc{load}(e, \boldsymbol{f}, \boldsymbol{D}) = \frac{1}{\mathrm{b}(e)} \cdot \sum_{s,t} D_{st} \cdot f_{st}(e) \ \ ,$$

i.e., the total flow traversing e divided by the bandwidth of e. The *congestion* $C(\boldsymbol{f}, \boldsymbol{D})$ for routing $\boldsymbol{D}$ using $\boldsymbol{f}$ is is the maximum load of a network link, i.e., $C(\boldsymbol{f}, \boldsymbol{D}) = \max_{e \in E} \textsc{load}(e, \boldsymbol{f}, \boldsymbol{D})$.

For a demand matrix $\boldsymbol{D}$ we denote the optimal congestion that can be achieved when routing these demands, with $C_{\text{opt}}(\boldsymbol{D})$. The competitive ratio of a routing scheme $\boldsymbol{f}$ is the maximum possible ratio between the congestion produced by the routing scheme and the congestion produced by an optimal scheme. Formally,

$$\textsc{competitive-ratio}(\boldsymbol{f}) = \sup_{\boldsymbol{D}} \frac{C(\boldsymbol{f}, \boldsymbol{D})}{C_{\text{opt}}(\boldsymbol{D})} \ \ .$$

We are interested in obtaining an *optimal oblivious routing scheme* for a network G, i.e., a scheme f that obtains the best possible competitive ratio, that is

$$f = \arg\min_{g} \textsc{competitive-ratio}(g) \ .$$

2.4.2 LP formulation

The main theorem of this section is as follows.

Theorem 2.22 *There is a polynomial time algorithm that for any input network G (directed or undirected) outputs a routing f such that the competitive ratio of f is best possible, i.e., $f = \arg\min_g \textsc{competitive-ratio}(g)$.*

The running time of our algorithm will be polynomial in the number of nodes n, and in $\textsc{rep}(\mathrm{b}(\cdot))$, the size of the bit representation of the edge capacities. If the input network is undirected we apply the transformation illustrated in Figure 2.5.

We first observe that the problem of computing an optimal oblivious routing can be formulated as the following LP that has $|E| \cdot |V|^2 + 1$ variables, but an infinite (continuous) number of constraints.

$$\begin{array}{ll} \text{minimize} & z \\ \text{subject to} & f \text{ is a routing} \\ & \forall e \in E \, \forall \boldsymbol{D} \ \ \textsc{load}(e, f, \boldsymbol{D}) \leq z \cdot C_{\text{opt}}(\boldsymbol{D}) \end{array} \tag{2.7}$$

The variables in this LP are the routing variables $f_{st}(e)$ and the minimization parameter z. The constraints of this LP are the *routing constraints* (Equation (2.6)) which specify that the variables $f_{st}(e)$ constitute a routing scheme, and for every demand matrix $\boldsymbol{D}$, and every edge $e \in E$, a constraint that ensures that routing f produces at most load $z \cdot C_{\text{opt}}(\boldsymbol{D})$ on e. The latter type of constraint will be called *congestion constraint* in the following. Note that the demand matrices $\boldsymbol{D}$ and the respective optimal congestion values $C_{\text{opt}}(\boldsymbol{D})$ are constants in this LP.

Our solution essentially solves LP 2.7 using the Ellipsoid method with a separation oracle (see [GLS88]). The problem with this approach is that our LP contains constraints that have coefficients with non-polynomial bit representation. This affects the running time of the Ellipsoid method in two ways. First, the bound on the number of iterations depends on the size of the initial ellipsoid and the smallest "volume" of a feasible set. If the bit representation of constraints is not polynomial, the minimum volume of a feasible set may be too small. Second, the time needed for a single iteration is non-polynomial if the separation oracle returns a "violated constraint" (demand matrix) with non-polynomial size.

Therefore, we first show that the constraints in our LP can be pruned such that only constraints with short bit representation remain. All other constraints are redundant. For this, observe that if we scale a demand matrix $\boldsymbol{D}$, the ratio

$$\frac{\text{LOAD}(e, f, \boldsymbol{D})}{C_{\text{opt}}(\boldsymbol{D})}$$

remains fixed. Hence, it suffices to use only the congestion constraints of LP 2.7 where the demand matrix can be routed with minimum congestion equal to 1, as all other demand matrices are scaled versions of such a matrix. We denote the set of demand matrices $\boldsymbol{D}$ that can be routed with congestion at most 1 by

$$H_1 = \{\boldsymbol{D} \mid C_{\text{opt}}(\boldsymbol{D}) \leq 1\} \ .$$

We now consider the LP, where the congestion constraints only contain constraints for demand matrices from H_1.

$$\begin{array}{ll} \text{minimize} & z \\ \text{subject to} & f \text{ is a routing} \\ & \forall e \in E\, \forall \boldsymbol{D} \in H_1 \;\; \text{LOAD}(e, f, \boldsymbol{D}) \leq z \end{array} \qquad (2.8)$$

This LP is equivalent to LP 2.7.[10]

We now show that H_1 constitutes a polyhedron on $|V|^2$-dimensional space, defined by a polynomial (in $|V|$) number of inequalities, with coefficients in the set $\{\text{b}(e) \mid e \in E\} \cup \{+1, -1\}$. Then, we argue that we can further trim the congestion constraints to demand matrices that are vertices of H_1. This gives the desired result that the LP only contains constraints with small bit representation.

The polyhedron H_1.

The polyhedron H_1 is the projection of the following on the variables $\boldsymbol{D}$.

1. The conservation constraints that guarantee that there is a flow $\boldsymbol{g}$ shipping D_{st} from s to t, for every pair s, t.

$$\begin{array}{lrcl} \forall s, t \in V & D_{st} & \geq & 0 \\ \forall e \in E \quad \forall s, t \in V & g_{st}(e) & \geq & 0 \\ \forall v \in V \quad \forall s, t \in V \setminus \{v\} & \sum_{e \in \text{OUT}(v)} g_{st}(e) - \sum_{e \in \text{IN}(v)} g_{st}(e) & = & 0 \\ \forall s, t \in V & \sum_{e \in \text{OUT}(s)} g_{st}(e) - \sum_{e \in \text{IN}(s)} g_{st}(e) & = & D_{st} \end{array}$$

[10]It would be sufficient to define $H_1^= = \{\boldsymbol{D} \mid C_{\text{opt}}(\boldsymbol{D}) = 1\}$, and to include only congestion constraints for demand matrices from $H_1^=$ in the LP. The definition of H_1 as $\{\boldsymbol{D} \mid C_{\text{opt}}(\boldsymbol{D}) \leq 1\}$ will simplify the following proof. Note that for demand matrices $\boldsymbol{D}$ with $C_{\text{opt}}(\boldsymbol{D}) < 1$ the congestion constraint in LP 2.8 has been relaxed, compared to the corresponding constrained in LP 2.7. However, this relaxation does not affect the set of feasible solutions of the system.

2. Constraints ensuring that the flow $\boldsymbol{g}$ has congestion at most 1.

$$\forall e \in E \quad \sum_{s,t} g_{st}(e) \quad \leq \quad \mathrm{b}(e)$$

It is not hard to see that the feasible solutions of this system are all demand matrices $\boldsymbol{D}$ such that the demands can be routed by some flow with congestion at most 1, i.e., $C_{\mathrm{opt}}(\boldsymbol{D}) \leq 1$. Hence, the above inequalities indeed characterize the set H_1.

The following lemma states that our LP only needs to contain congestion constraints for demand matrices that are vertices of H_1.

Lemma 2.23 *Let* $\mathrm{V}(H_1)$ *denote the set of vertices of* H_1*. All congestion constraints in LP 2.8 for demand matrices* $\boldsymbol{D} \notin \mathrm{V}(H_1)$ *are redundant.*

Proof. Suppose that there is a candidate routing $\boldsymbol{f}$ that is infeasible because of the congestion constraint for edge $e \in E$ and demand matrix $\boldsymbol{D}' \in H_1$, i.e.,

$$\textsc{load}(e, \boldsymbol{f}, \boldsymbol{D}') > z \ .$$

We show that there is a demand matrix $\boldsymbol{D}'' \in \mathrm{V}(H_1)$ from the vertex set of H_1 with $\textsc{load}(e, \boldsymbol{f}, \boldsymbol{D}'') > z$, i.e., the constraint for matrix $\boldsymbol{D}'$ is not required to certify the infeasibility of routing $\boldsymbol{f}$. Consider the following linear program.

$$\begin{array}{ll} \text{maximize} & \textsc{load}(e, \boldsymbol{f}, \boldsymbol{D}) \\ \text{subject to} & \boldsymbol{D} \in H_1 \end{array}$$

The routing $\boldsymbol{f}$ is a constant in this LP. It is well known that at least one of the maxima of a linear objective function over a polyhedron is obtained at a vertex of the polyhedron. We choose $\boldsymbol{D}''$ as a vertex maximum of the above linear program. Then we have

$$\textsc{load}(e, \boldsymbol{f}, \boldsymbol{D}'') \geq \textsc{load}(e, \boldsymbol{f}, \boldsymbol{D}') > z \ ,$$

as desired. This shows that we can trim all non-vertex constraints from LP 2.8, without changing the solution of the LP. ∎

Finally, we are left with the solution to

$$\begin{array}{ll} \text{minimize} & z \\ \text{subject to} & \boldsymbol{f} \text{ is a routing} \\ & \forall e \in E \, \forall \boldsymbol{D} \in \mathrm{V}(H_1) \ \ \textsc{load}(e, \boldsymbol{f}, \boldsymbol{D}) \leq z \end{array} \tag{2.9}$$

which is an LP with a polynomial number of variables, and constraints that have a bit representation polynomial in $(n, \textsc{rep}(\mathrm{b}(\cdot)))$. We can solve this LP in polynomial time using the following separation oracle.

Separation Oracle.

- **Input:** A network G, a capacity function $\mathrm{b}(\cdot)$, and an assignment to all variables of LP 2.9, i.e., variables $f_{st}(e)$ and minimization variable z.
- **Output:** Either, a confirmation that all constraints of LP 2.9 are fulfilled, or a "violated constraint", this means

  a) a violated routing constraint (Equation (2.6)), that shows that $\boldsymbol{f}$ does not constitute a routing scheme,

  or b) a violated congestion constraint, i.e., a demand matrix $\boldsymbol{D} \in \mathrm{V}(H_1)$ and an edge $e \in E$ such that

  $$\textsc{load}(e, \boldsymbol{f}, \boldsymbol{D}) > z \ .$$

- **Implementation:** The oracle algorithm first checks all routing constraints. These are only $O(E \cdot |V|^2)$ inequalities and can therefore be checked in polynomial time.

  The congestion constraints are checked as follows. For every edge $e \in E$ in turn the algorithm solves the following LP

  $$\begin{array}{ll} \text{maximize} & \textsc{load}(e, \boldsymbol{f}, \boldsymbol{D}) \\ \text{subject to} & \boldsymbol{D} \in H_1 \end{array}$$

  and returns a vertex solution $\boldsymbol{D_e} \in \mathrm{V}(H_1)$. If for any edge $\textsc{load}(e, \boldsymbol{f}, \boldsymbol{D_e}) > z$, the algorithm has found a violated constraint. Otherwise, for all edges and demand matrices from $\boldsymbol{D} \in \mathrm{V}(H_1)$, we have

  $$\textsc{load}(e, \boldsymbol{f}, \boldsymbol{D}) \leq \textsc{load}(e, \boldsymbol{f}, \boldsymbol{D_e}) \leq z \ ,$$

  which means that all congestion constraints in LP 2.9 are fulfilled. The running time of the oracle algorithm is polynomial, as the LP's have a polynomial number of variables, and a polynomial number of constraints with polynomial bit representation.

We can use this separation oracle for solving LP 2.9 in polynomial time with the Ellipsoid method. This gives the oblivious routing scheme that achieves the best possible competitive ratio w.r.t. congestion for a given network G.

The methodology presented in this section is not limited to optimize for an oblivious routing scheme that minimizes congestion but it can perform many different optimizations. One example is minimizing node congestion (this corresponds to router load on IP networks) which is the ratio of the total traffic traversing a node to its capacity. It is also possible to consider edge and node congestion simultaneously; to consider linear combinations of edges or nodes; to add additive factor to the congestion formula; or to limit the dilation.

Another interesting possibility is to limit the class of demand matrices in some way, as, e.g., to limit the sum of demands or require zero demand between certain pairs. The limiting factor in the selection of the optimization function and the restriction of demand matrices is the ability to express the problem and the separation oracle using linear constraints.

Note however that for these cost metrics we do not have a general bound on the competitive ratio of an oblivious routing scheme. Even for undirected networks the competitive ratio may be very large. But the results in this section make it possible to calculate the performance, i.e., the competitive ratio, of the best oblivious routing scheme for a given input network. Then a network administrator may decide whether this performance guarantee is acceptable for the intended application, or adaptive routing algorithms have to be used instead.

2.5 Conclusions

We have developed solutions to data management and routing problems that work completely distributed and create a low congestion on the network links of a distributed system. In particular we have presented a basic technique that transforms a congestion-efficient tree solution for a problem, into a solution on general undirected networks. These results directly lead to the following questions concerning the possibilities to extend the technique to new scenarios and to improve the results on the competitive ratios guaranteed by the framework.

- Can the bisimulation technique be extended to directed networks?
- How low can we make $\delta_{\mathcal{H}}$, the factor between the competitive ratio of a tree strategy and the competitive ratio of the corresponding strategy for G?
- Is it possible to incorporate more complex cost-measures into the framework such that not simply the congestion but some combination of congestion and total communication load is minimized?
- Can we obtain strictly competitive algorithms with polylogarithmic competitive ratios for general networks, i.e., networks where $\mathrm{b}_{\max}$ is arbitrarily large?

In the following we give some answers to these questions and discuss some open problems related to our work. We begin with the extension of the framework to directed graphs.

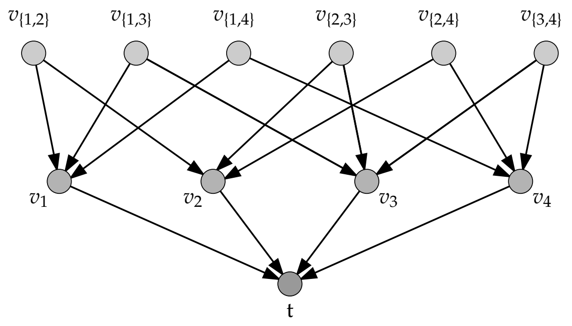

Figure 2.6: A directed network in which any oblivious routing scheme achieves a bad competitive ratio w.r.t. congestion.

Directed Networks. The question whether the framework can be extended to directed networks can be answered negatively. The following result from [ACF$^+$03] shows that in directed networks the best possible oblivious routing scheme may have a competitive ratio of $\Omega(\sqrt{n})$, where n denotes the number of nodes of the network. Since our framework implies an oblivious routing scheme with polylogarithmic competitive ratio, it cannot be extended to directed networks.

Lemma 2.24 *There exist directed networks such that the competitive ratio for an oblivious routing algorithm w.r.t. congestion is $\Omega(\sqrt{n})$.*

Proof. Consider the following directed network $G_k = (V, E)$. The node set V can be partitioned into three different sets V_1, V_2 and V_3 such that there are only directed edges from nodes in V_i to nodes in V_{i+1} for $i \in \{1, 2\}$. In the following the nodes in V_i are called nodes on level i. The set $V_3 = \{t\}$ contains only one node called the target. The set V_2 contains k nodes denoted with $v_1, \dots, v_k$. Finally, the set V_1 contains $k(k-1)/2$ denoted with $v_{\{i,j\}}$, $i, j \in \{1, \dots, k\}$.

The edge set E is defined as follows. Every node v_i from level 2 is connected to the target via a directed edge (v_i, t). Further, a node $v_{\{i,j\}}$ from the first level is connected to nodes v_i and v_j on the second level. All edges have unit capacity. Figure 2.6 illustrates the graph G_k for $k = 4$.

Now, fix an oblivious routing scheme for G_k. We show that this scheme has competitive ratio at least $(k-1)/2 = \Omega(\sqrt{n})$. The routing scheme defines for each node $v_{\{i,j\}}$ a unit flow to the target t that defines how routing requests between $v_{\{i,j\}}$ and t are routed. Adding all these flows, we get that there is a total flow of value $k(k-1)/2$ that reaches t. Hence, there must be a node v_ℓ on the second level that sends at least flow $(k-1)/2$ along its outgoing edge. This flow originates at nodes $v_{\{i,\ell\}}$, $i \in \{1, \dots, k\} \setminus \{\ell\}$.

Consider the following routing problem. Each node $v_{\{i,\ell\}}$, sends one unit of flow to the target. The oblivious routing scheme creates at least load $(k-1)/2$

on edge (v_ℓ, t). However, the routing problem can be solved with congestion 1 by simply routing the flow originating at $v_{\{i,\ell\}}$ over v_i to the target. Hence, the oblivious routing scheme has competitive ratio at least $(k-1)/2$. This proves the lemma. ■

The above lemma shows that the worst case competitive ratio of oblivious routing schemes differs substantially on directed and on undirected networks. This is interesting because for online routing algorithms such a substantial difference does not exist, as there are algorithms that achieve a competitive ratio of $O(\log n)$ for directed as well as for undirected networks (see [AAF+97]).

An explanation of this phenomenon may be that in undirected networks the limitations of obliviousness are not so critical because an oblivious algorithm can choose between many paths and can therefore achieve enough path diversification by using randomization. In directed networks this does not hold because the orientation of edges seriously restricts the number of different paths that can be selected between a source/target pair. Therefore the oblivious algorithm cannot achieve a good path diversification in directed networks and, hence, has a bad competitive ratio. In contrast to this, an online algorithm can cope with directed edges by using a traffic dependent path selection.

Strictly competitive routing. In Section 2.3.5 we presented a strictly competitive virtual circuit routing algorithm with competitive ratio $O(\delta_{\mathcal{H}} \cdot \mathrm{b}_{\max})$. This ratio depends on $\mathrm{b}_{\max}$, i.e., the ratio between the highest capacity and the lowest capacity of a link in the network. At first glance, this dependency seems unsatisfactory. However, the following lemma shows that, in general, the competitive ratio of a strictly competitive algorithm depends on $\mathrm{b}_{\max}$.

Lemma 2.25 *There exist undirected networks, in which any strictly competitive, oblivious virtual circuit routing algorithm has competitive ratio* $\Omega(\mathrm{b}_{\max})$.

Proof. Consider a network G as illustrated in Figure 2.7. There are two nodes a and b connected with a single link e of high capacity (capacity $\mathrm{b}_{\max}$) and with $\lceil \mathrm{b}_{\max}^2 \rceil$ links of low capacity (capacity 1). We show that on this graph any strictly competitive oblivious routing algorithm has at least competitive ratio $\Omega(\mathrm{b}_{\max})$. Note that our "bad example" has multi-edges, and that strictly speaking such a graph does not fit into our model as we model the network as an ordinary graph without multi-edges. However, the multi-edges are only used for simplifying the description of the proof. The proof also works for the graph G', obtained from G by adding an intermediate node to each edge of G, i.e., replacing edge e_i by edges (a, x_i), and (x_i, b).

The proof works as follows. Fix an oblivious routing scheme. This scheme defines a unit flow between a and b for routing the demands between these nodes.

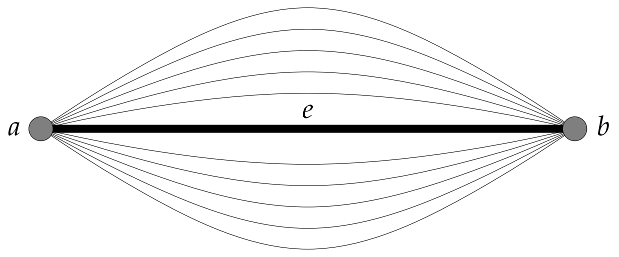

Figure 2.7: A bad network for strictly competitive routing algorithms.

We make a case distinction according to the fraction of this flow that is routed along *e*.

- For the case that more than half of the flow traverses the high capacity link *e*, a demand of $\lceil b_{\max}^2 \rceil$ messages between *a* and *b* creates expected load at least $b_{\max}/2$ on *e*. This gives an expected congestion of at least $b_{\max}$ for the oblivious routing scheme. Since the demand can be routed with congestion 1 by using only low capacity links, the competitive ratio of the scheme is at least $b_{\max}/2$.
- If, however, the flow along *e* is less than 1/2, a demand of 1 message between *a* and *b* is likely to create a bad congestion. With probability at least 1/2 the messages is routed along a low capacity link, thereby inducing a load of 1. Hence, the expected congestion of the oblivious routing scheme is at least 1/2. The optimum algorithm routes the message along the high capacity link which gives a congestion of $1/b_{\max}$. Therefore, the competitive ratio of the oblivious routing scheme is at least $b_{\max}/2$.

Both cases yield the lemma. ■

The above negative result is somehow specific to our way of modeling the network and the routing process. If, e.g., the oblivious algorithm could route fractionally, the load on any edge would be equal to the expected load in our randomized model. In this case Theorem 2.2 gives a factor of $O(\delta_{\mathcal{H}})$ between the congestion of the oblivious algorithm, and the congestion of an optimal algorithm and, hence, the oblivious algorithm is strictly $O(\delta_{\mathcal{H}})$-competitive.

Harrelson et al. use the following network model. They assume that all links in the network have unit capacity, and different bandwidth between nodes is modeled by multi-edges (i.e., a link with bandwidth *b* in the physical network is modeled via *b* links of unit bandwidth between the respective nodes in the graph). In this network model the routing algorithm of Lemma 2.5 is strictly competitive.

Different cost-measures. The question whether the bisimulation framework can be extended to further cost-measures is well motivated by the packet routing problem. The task in this problem is to schedule a set of packets from their source nodes to the corresponding target nodes as quickly as possible. It is assumed that the network is synchronized and that each network link can forward at most one packet in a time step.

This problem is often split into two subproblems, namely the *path selection problem*, in which one has to select routes for the packets, and the *scheduling problem*, in which the packets have to be scheduled along their predetermined routes, and conflicts between competing packets have to be resolved. The time needed for scheduling all packets it at least $\Omega(C + D)$, where C is the *congestion* of the chosen path system, i.e., the maximum number of paths that go through a single edge, and D is the *dilation*, i.e., the length of the longest path taken by a packet. There exists a lot of work (see, e.g., [LMR94, OR97]) that shows that one can in fact solve the scheduling problem in time close to this lower bound. This is even possible in a distributed scenario in which scheduling decisions are only based on local information.

In contrast to this, there is very little work about path selection strategies for general networks. Srinivasan and Teo show in [ST00] how to approximate an optimum path system, i.e., a path system that minimizes $C + D$, in a centralized manner. However, there are no distributed solutions to this problem for general networks.

If the path selection is done using our framework, it is only guaranteed that the congestion is nearly optimal, but there is no bound on the dilation. The following lemma shows that it is indeed impossible for an oblivious algorithm to get a good competitive ratio w.r.t. sum of congestion and dilation in general undirected networks.

Lemma 2.26 *There exist undirected networks in which no oblivious path selection algorithm can achieve a good competitive ratio w.r.t. $C + D$.*

Proof. Consider the network G_ℓ shown in Figure 2.8. There are two nodes a and b that are connected via many disjoint paths; one path with length 1, and ℓ paths with length ℓ. Altogether G_ℓ has $n = \ell(\ell - 1) + 2$ nodes, which gives $\ell = \Omega(\sqrt{n})$.

Now, consider an oblivious routing for G_ℓ. We show that such a scheme either creates a very large congestion or a very large dilation for some routing problems. The routing scheme defines a unit flow from a to b that defines how routing path between a and b are chosen. We distinguish two cases according to the fraction of this flow that traverses edge $e = (a, b)$.

- Suppose that more than half of the flow traverses e. This means that more than half of the routing paths between a and b contain e. Then the routing problem that sends ℓ^2 packets from a to b creates an expected load of at

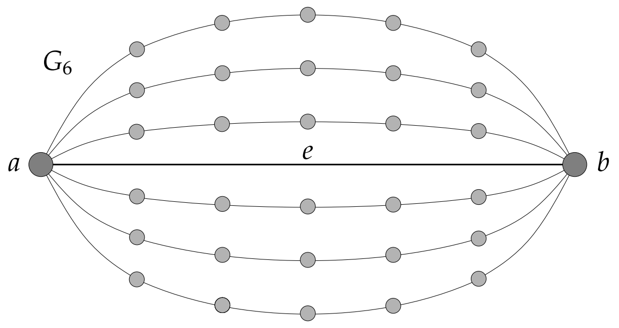

Figure 2.8: A network in which an oblivious routing scheme cannot achieve a good competitive ratio w.r.t. sum of congestion and dilation.

least $\ell^2/2$ on edge e. Hence, for the congestion C_{obl} and dilation D_{obl} of the oblivious routing scheme we get $C_{\text{obl}} + D_{\text{obl}} \geq \ell^2/2 = \Omega(n)$. However, routing the flow using only long paths gives a congestion of $\ell^2/\ell = \ell$ and a dilation of ℓ which means that an optimal routing algorithm would achieve $C + D \leq 2\ell = O(\sqrt{n})$. Hence, the competitive ratio of the oblivious routing scheme is at least $\Omega(\sqrt{n})$.

- Now, suppose that less than half of the flow traverses e. Then the expected length of a path between a and b that is chosen according to the oblivious routing scheme is at least $\ell/2 = \Omega(\sqrt{n})$. Therefore the routing problem that sends only one packet from a to b has an expected dilation of $\Omega(\sqrt{n})$. Clearly, this routing problem can be solved with congestion and dilation one, by sending the message along e. Hence, in this case the oblivious routing scheme has a competitive ratio of $\Omega(\sqrt{n})$, as well.

Combining both cases gives the lemma. ■

Although, this result shows that there is no good oblivious routing scheme with respect to sum of congestion and dilation, there exist other promising ways to incorporate the length of routing path into the cost-measure. One possibility to penalize long routing paths is to minimize the ℓ_p-norm of the link loads, i.e., $\sqrt[p]{\sum_{e \in E} L(e)^p}$, where $L(e)$ denotes the load of link e.

There are good oblivious routing schemes for $p = 1$, i.e., for the total communication load, and for $p = \infty$, i.e., for the congestion of the network links. It is an interesting task for further research to design efficient, oblivious algorithms that work for arbitrary p. However, it is not clear whether this is possible, or whether also for the ℓ_p-norm a counterexample like that in Figure 2.8 exists.

Improving the competitive ratio. The most challenging open problem concerning the bisimulation approach is the question whether the competitive ratio guaranteed by the framework can be improved. Clearly, $O(\log n)$ is a lower bound for general networks, since the lower bound of $O(\log n)$ for online routing on the 2-dimensional mesh (see [MMVW97] and [BL97]) also holds for oblivious routing, and hence also for the competitive ratio guaranteed by our framework. Currently the best upper bound is $O(\log^2 n \log\log n)$ due to Harrelson, Hildrum, and Rao [HHR03].

This bound seems too unnatural in order to believe that it is tight. However, conjecturing some value between $O(\log n)$ and $O(\log^2 n \log\log n)$ as the true bound would be pure speculation.

In the following we try to develop a vague intuition why it might be difficult to improve the bound beyond $O(\log^2 n)$ by using the technique of a hierarchical network decomposition.

There are strong indications that a hierarchical framework that guarantees a competitive ratio of c will lead to a c-approximation algorithm for minimum bisection. As the best known upper bound for this problem is $O(\log^2 n)$ for general networks (see [FK00]) this might justify our assumption that improving the competitive ratio to $o(\log^2 n)$ is difficult. Note, however, that this argumentation is very vague for the following reasons:

- We are not able to prove that a c-competitive hierarchical framework really gives a c-approximation algorithm for minimum bisection. This is only a conjecture and we supply some reasons why we believe that this conjecture is true.

- There is no hardness result known for minimum bisection that makes it impossible that, e.g., even a constant factor approximation for this problem exists.

Nevertheless, the following description reveals an interesting relation between our framework and the minimum bisection algorithm presented by Feige and Krauthgamar [FK00]. This approximation algorithm first computes a recursive network decomposition. Then this decomposition is used to find a minimum bisection in the graph G, i.e., a minimum capacity cut $(S, V \setminus S)$ in G such that both sides S and $V \setminus S$ contain $n/2$ nodes. We use the following notation from [FK00]. We refer to the two sides of a bisection as *white* W and *black* B. Further, we fix one optimal bisection and denote it with (W^*, B^*). A *labeling* of the decomposition tree assigns a color (either white or black) to each node of the tree.

Feige and Krauthgamar define a *charge* of a bisection with respect to a labeling such that the following holds

$$\begin{aligned} &\forall \text{bisections } (W,B),\ \text{labelings } L: && \mathrm{cap}(W,B) \le \mathrm{charge}_L(W,B) \\ &\forall \text{bisections } (W,B),\ \exists\, \text{labeling} L': && \mathrm{cap}(W,B) \ge 1/O(\log^2 n) \cdot \mathrm{charge}_{L'}(W,B)\ , \end{aligned} \tag{2.10}$$

where $\mathrm{charge}_L(W,B)$ denotes the charge of bisection (W,B) w.r.t. labeling L. Then they present an algorithm that computes the combination of a bisection (W',B') and a labeling L' that minimizes the charge. The above inequalities give

$$\begin{aligned} \mathrm{cap}(W',B') &\le \mathrm{charge}_{L'}(W',B') \\ &\le \mathrm{charge}_{L^*}(W^*,B^*) \\ &\le O(\log^2 n) \cdot \mathrm{cap}(W^*,B^*)\ , \end{aligned}$$

which means that (W',B') is an approximate minimum bisection with approximation ratio $O(\log^2 n)$. Informally speaking, Feige and Krauthgamar use the charge of a bisection to approximate its size. This approximation has the advantage that a bisection with minimum charge can be computed efficiently, while a bisection with minimum size is very difficult to compute.

In the following we show that to some extend this approach works for our hierarchical decomposition, as well. We show that we can define a charge for our decomposition in such a way that inequalities (2.10) are fulfilled.

Assume that we have a hierarchical decomposition $\mathcal{H}$ that guarantees a competitive ratio of $\delta_{\mathcal{H}}$. This means that for each edge e we have

$$\sum_{H \in \mathcal{H}:\, e \in H} c_H \le \delta_{\mathcal{H}}\ ,$$

where c_H denotes the maximum edge congestion for the solution of the CMCF-problem of cluster H. We define a charge for a bisection w.r.t. a labeling of the decomposition tree $T_{\mathcal{H}}$ as follows.

Definition 2.27 *The charge of a level ℓ cluster $H \in \mathcal{H}$ for bisection (W,B) w.r.t. labeling L is*

$$\mathrm{charge}_L^H(W,B) = \begin{cases} w_{\ell+1}(H \cap B) & L(H) = \text{"black"} \\ w_{\ell+1}(H \cap W) & L(H) = \text{"white"} \end{cases}$$

i.e., the weight of those nodes in H that are colored according to the label of H. The charge $\mathrm{charge}_L(W,B)$ *of the bisection (W,B) is defined as* $\sum_{H\in\mathcal{H}} \mathrm{charge}_L^H(W,B)$.

We show that by this definition the charge fulfills inequalities (2.10).

- First we show that the charge for a bisection (W, B) is always larger than its size, i.e., larger than $\text{cap}(W, B)$. Suppose an edge e contributes to $\text{cap}(W, B)$. This means that it connects a white and a black node in the bisection (W, B). Assume that e is cut before or at level $\ell + 1$ in the hierarchical decomposition, i.e., both endpoints belong to the same level ℓ cluster but to different level $\ell + 1$ clusters. Let H denote the level ℓ cluster that contains e.

  Recall that $\text{w}_{\ell+1}(v)$ counts the capacity of edges that are adjacent to v and are cut before or at level $\ell + 1$. Depending on the labeling, the charge of cluster H is either equal to the weight of white nodes, or equal to the weight of black nodes in the cluster. However, in both cases the capacity of e is counted, as it has a black and a white endpoint in the cluster. This shows that the capacity of every edge between W and B contributes to the charge of (W, B). Hence, $\text{charge}_L(W, B) \geq \text{cap}(W, B)$ holds for all labelings L.

- Now, we show that there is a labeling L' such that $\text{charge}_{L'}(W, B) \leq O(\delta_{\mathcal{H}}) \cdot \text{cap}(W, B)$. We call a labeling *$\alpha$-consistent* (compare [FK00]) for bisection (W, B) if the following holds: if the label of a cluster H is white then $\text{w}_{\ell+1}(H \cap W) \leq \alpha \cdot \text{w}_{\ell+1}(H)$, and if the label of H is black then $\text{w}_{\ell+1}(H \cap B) \leq \alpha \cdot \text{w}_{\ell+1}(H)$. The following lemma shows that we can choose L' as any α-consistent labeling for some constant $\alpha \in (0, 1)$.

Lemma 2.28 *The charge of a bisection (W, B) with respect to an α-consistent labeling is at most $\delta_{\mathcal{H}}/(1 - \alpha) \cdot \text{cap}(W, B) = O(\delta_{\mathcal{H}}) \cdot \text{cap}(W, B)$.*

Proof. Let L' denote an α-consistent labeling for (W, B). Fix a cluster H on some level ℓ. Without loss of generality assume that the label of H is white. Then the charge of cluster H is $\text{w}_{\ell+1}(W \cap H)$. We compare this charge to $\text{cap}(W \cap H, B \cap H)$, i.e., the capacity of edges between W and B that lie in H.

The CMCF-problem for H requires that a demand of $\text{w}_{\ell+1}(W \cap H)$ (generated by the white nodes in H) is distributed evenly among all nodes (according to the weight of these nodes). In particular this means that a $(1 - \alpha)$-fraction of this demand has to cross the cut $(W \cap H, B \cap H)$, since the nodes in $B \cap H$ have more than a $(1 - \alpha)$-fraction of the total weight of H, because the labeling L' is α-consistent. Since the CMCF-problem can be solved with congestion c_H, this cut must have capacity at least $1/c_H \cdot (1 - \alpha) \cdot \text{w}_{\ell+1}(W \cap H)$, which gives

$$\text{charge}_{L'}^{H}(W, B) \leq \frac{c_H}{1 - \alpha} \cdot \text{cap}(W \cap H, B \cap H).$$

When summing this inequality over all H we can utilize that an edge in the cut (W, B) is only counted for clusters it belongs to. We get

$$\begin{aligned}
\mathrm{charge}_{L'}(W,B) &= \sum_H \mathrm{charge}_{L'}^H(W,B) \\
&\le \sum_H \frac{c_H}{1-\alpha} \cdot \mathrm{cap}(W \cap H, B \cap H) \\
&= \sum_H \sum_{e \in (W \cap H \times B \cap H)} \frac{c_H}{1-\alpha} \cdot \mathrm{b}(e) \\
&= \sum_{e \in W \times B} \sum_{H : e \in H} \frac{c_H}{1-\alpha} \cdot \mathrm{b}(e) \\
&\le \sum_{e \in W \times B} \frac{\delta_{\mathcal{H}}}{1-\alpha} \mathrm{b}(e) = O(\delta_{\mathcal{H}}) \cdot \mathrm{cap}(W,B) \ ,
\end{aligned}$$

as desired. ■

The above results show that a bisection with minimum charge as defined in Definition 2.27 is an approximate minimum bisection with approximation ratio $O(\delta_{\mathcal{H}})$. However, in order to obtain an $O(\delta_{\mathcal{H}})$-approximation algorithm for minimum bisection, the combination of labeling and bisection that minimizes charge has to be computed in polynomial time. Our hierarchy *does not* allow a dynamic programming approach like that used in [FK00] because in contrast to the decomposition of Feige and Krauthgamar, our decomposition does not have constant degree.[11] Therefore, it seems very unlikely that it is possible to derive an approximation algorithm for minimum bisection by directly using our hierarchical decomposition.

The reason for this is that the oblivious routing problem makes much higher demands on the hierarchical decomposition than the bisection problem. These demands do not allow decompositions with low degree and, hence, the approximation algorithm for minimum bisection on these decompositions becomes difficult. However it is possible to obtain a decomposition well suited for the bisection problem by slightly changing our decomposition technique. We only change the definition of weight such that $\mathrm{w}_\ell(v)$ counts the capacity of all edges adjacent to v that are cut at level ℓ (instead of edges that are cut *before or at* level ℓ). The definition of the CMCF-problem for a cluster remains the same (only the demands now depend on the new definition of weight). The quality of a decomposition is measured as before by the height and the maximum congestion of a CMCF-problem.

[11] In fact there are networks for which our decomposition has very large degree. To see this recall that our clusters have to fulfill the precondition, which restricts the ratio between the capacity of edges connecting a subset to the rest of its cluster, and the capacity of edges connecting the subset to nodes outside of the cluster. When partitioning a complete graph this restriction prohibits clusters of small size strictly larger than 1 (e.g., size $\sqrt{n}$). Therefore some of the tree nodes must have a very high degree in a valid decomposition of the complete network.

For this new decomposition the definition of charge as in Definition 2.27 still fulfills inequalities (2.10). Hence, a bisection with minimum charge is an approximate minimum bisection. However, the new decomposition rules allow to construct a decomposition with constant degree, because during the partitioning process the precondition is not important any more.[12] For this decomposition we can apply the dynamic programming techniques from [FK00] and compute the bisection with minimum charge in polynomial time. This gives an approximation algorithm for minimum bisection.

Because of this relationship between the minimum bisection algorithm of Feige and Krauthgamar and our decomposition, we believe that an improvement of the hierarchical decomposition will lead to improved approximation guarantees for minimum bisection. However, when only focusing on the oblivious routing problem a hierarchical approach might not be needed. Perhaps it is possible to improve the bound on oblivious routing by a careful analysis of the linear programming formulation developed in Section 2.4. Moreover, Applegate and Cohen [AC03] have presented an improved version that has a polynomial number of variables and a polynomial number of constraints. We think that further research in this direction is a more promising approach for improving oblivious routing than the improvement of the hierarchical decomposition.

[12]We can partition *any* connected subcluster according to the new rules (not only clusters that fulfill the precondition). Therefore in each step a partitioning algorithm only needs to find a balanced, binary partition for which the CMCF-problem can be solved efficiently. This gives constant degree.

CHAPTER 3

Cost-efficient Data Management Strategies

In this section we investigate data management strategies for commercial networks, in which resources are not for free as in the model investigated in Chapter 2, but are subject to fee. Such a scenario arises, e.g., in the area of Grid computing, where resource may only be shared for a certain price, or even more apparent in the area of web publishing, as web space usually is not for free, in particular, if a certain quality-of-service is guaranteed.

The existence of monetary costs has crucial effects on a data management strategy. For example the approach used in Chapter 2, which aims at utilizing the available resources as best as possible, may lead to data management strategies that induce very high monetary cost. We, therefore, present data management strategies that try to minimize the commercial cost instead of the communication cost for a given request pattern.

Our model mirrors, e.g., the perspective of a content provider that offers information via pages in the WWW. For that purpose, the content provider has to rent or buy some amount of the resources bandwidth and memory. We assume that there is a fee per transmitted byte for each communication link and a fee per stored byte for each memory module in the network. Then, the total cost for the content provider is a function of the amount of bytes that are sent along communication links or stored in memory modules.

In this cost based model we will present static data management strategies. In the static scenario we are given read and write request frequencies for each node-object pair. A data management strategy has to calculate a placement of the objects to the memory modules, possibly with redundancy, such that the total cost is minimized.

3.1 The cost based model

The computer system is modeled by an undirected graph $G = (V, E)$ with node set V and edge set E such that the nodes represent the processors with their memory modules, and the edges represent the links. The *cost per stored data object* for the memory modules is described by a function $\mathrm{c_s} : V \longmapsto \mathbb{R}_0^+$, where $\mathbb{R}_0^+$ denotes the set of nonnegative real numbers, and the *cost per transmitted data object* for the links is described by a function $\mathrm{c_t} : E \longmapsto \mathbb{R}_0^+$. For simplicity, we assume that all data objects have uniform size. Thus, the functions $\mathrm{c_s}$ and $\mathrm{c_t}$ do not depend on the data objects. However, all our results hold also in a non-uniform model, since our algorithms place all objects independently from each other.

Let $\mathrm{c_t}(v, v')$ denote the *cost per transmitted data object* from a node v to a node v'. We define

$$\mathrm{c_t}(v, v') := \min_{\text{path } p \text{ from } v \text{ to } v'} \left\{ \sum_{\text{edge } e \text{ lies on } p} \mathrm{c_t}(e) \right\} .$$

Then, the function $\mathrm{c_t}$ defines a *metric space* over the node set V, since $\mathrm{c_t}$ is nonnegative, symmetric, and satisfies the triangle inequality. Therefore, $\mathrm{c_t}$ can also be seen as a distance function. To keep our algorithms and proofs simple and clear we often use this view of $\mathrm{c_t}$.

The *static data management problem* is defined as follows. We are given a *set X of shared data objects* and the *read and write request frequencies* for each node-object pair which are described by the functions $f_r : V \times X \longmapsto \mathbb{N}_0$ and $f_w : V \times X \longmapsto \mathbb{N}_0$, respectively. For each object $x \in X$, we have to determine a set of nodes holding copies of x. Then, it remains to specify how each request r for x will be served. The node issuing r is called the *home of request r* and is denoted with $h(r)$.

- In case of a read request r, the home $h(r)$ simply reads the nearest copy of x. The node holding this copy is called the *node serving request r* and is denoted with $s(r)$. Note that for a given set of nodes holding copies of x a read request is always served with optimal cost.

- In case of a write request r, an update is sent from $h(r)$ to all copies of x. Thus, we have to determine the edges along which this update is sent. This is modeled by a multi-set of edges E_{U_r} which is called the *update set of request r*. Edges in this update set can induce a multi-cast tree that branches at arbitrary nodes. Note that we allow edges to appear several times in an update set, however, this will never happen in an optimal update set. For technical reasons, the node holding the copy of x that is nearest to $h(r)$ is again called the *node serving request r* and is denoted with $s(r)$.

The goal in the *cost based model* is to calculate a placement of the objects to the memory modules such that the total cost is minimized. The *total cost* is defined as follows.

- A copy of object x on the node v increases the total cost by $\mathrm{c_s}(v)$.
- A read request r for object x increases the total cost by $\mathrm{c_t}(h(r), s(r))$.
- A write request r for object x increases the total cost by $\sum_{e \in E_{U_r}} E_{U_r}(e) \cdot \mathrm{c_t}(e)$, where $E_{U_r}(e)$ denotes the number of appearances of edge e in the multi-set E_{U_r}.

The model described above is slightly restrictive in the sense that it fixes the update policy to a certain range. In particular, it does not include strategies that allow only a fraction of the copies to be updated in case of a write, which, e.g., is implemented in strategies using the majority trick introduced in [UW87]. However, all strategies using such techniques add time stamps to the copies. This requires that there is some definition of uniform time among different nodes. Since it is not clear how to realize this in an asynchronous setting, we restrict ourselves to strategies that update or invalidate all copies in case of a write.

3.2 Related work and new results

Baev and Rajaraman [BR01] and Cidon, Kutten, and Soffer [CKS01] investigate static data management in a model that is similar to our model. The major difference is that they only consider read requests. Baev and Rajaraman [BR01] study static data management in arbitrary networks with memory capacity constraints. They present an algorithm that calculates a constant factor approximation in polynomial time. This algorithm is based on solving a linear programming relaxation of the problem and rounding the obtained solution. Cidon, Kutten, and Soffer [CKS01] study static data management in so-called directed trees. These networks are rooted trees, in which all edges are directed towards the root. This means that a node may only read a copy of a shared object that is placed on an ancestor in the tree. They present an algorithm that calculates an optimal solution in polynomial time, and can be executed in distributed fashion on the tree network in linear time.

Our static data management problem reduces to the uncapacitated metric facility location problem, if only read requests are considered. The MaxSNP-hardness of the uncapacitated metric facility location problem for arbitrary graphs [GK99] implies the same for the static data management problem on arbitrary networks. A large number of approximation algorithms are proposed for this facility location problem. Shmoys, Tardos, and Aardal [STA97] present the first constant factor approximation algorithm. Their algorithm is based on solving linear programming relaxations and rounding the obtained solutions. Korupolu, Plaxton, and Rajaraman [KPR98] analyze a simple local search heuristic and show that it achieves an approximation factor of $5 + \epsilon$, for any $\epsilon > 0$. The current best known approximation factor is 1.52 due to Mahdian, Ye, and Zhang [MYZ02]. Their algorithm uses the ideas of cost scaling, a greedy algorithm of Jain, Mahdian, and Saberi [JMS02], and a greedy augmentation procedure of Charikar, Guha, and Kuller [CG99, GK99].

Tamir [Tam96] presents an algorithm for the uncapacitated metric facility location problem on trees. This dynamic programming algorithm calculates an optimal solution in time $O(|V|^2)$ on a tree $T = (V, E)$. Subsequently but independently, Shah and Farach-Colton [SF02] have developed a dynamic programming algorithm for tree networks that is very similar to our data management strategy. However, they give a better analysis which results in an $O(|V| \log(|V|))$-algorithm for facility location on tree networks.

Our contribution

We introduce new deterministic algorithms for the static data management problem on trees and arbitrary networks. Our algorithms aim to minimize the total cost in our cost based model. Our main result, presented in Section 3.3, is a combinatorial algorithm that calculates a constant factor approximation for arbitrary graphs in polynomial time. Further, we present in Section 3.4 a dynamic programming algorithm for trees that calculates an optimal placement of all objects in X on a tree $T = (V, E)$ in time $O(|X| \cdot |V| \cdot \text{diam}(T) \cdot \log(\deg(T)))$, where $\text{diam}(T)$ denotes the *unweighted diameter* of T, i.e., the maximum number of edges on a simple path connecting two arbitrary nodes, and $\deg(T)$ denotes its *maximum node degree*.

3.3 The approximation algorithm for arbitrary networks

In this section, we present a combinatorial algorithm that calculates a constant factor approximation for the static data management problem on arbitrary graphs. Our algorithm places all objects independently from each other. Thus, fix an object x.

The focus in our algorithm lies on the calculation of a good placement for object x. Given such a placement write accesses are handled as follows. A node that issues a write request r for x, i.e., $h(r)$, first sends a message to the closest node holding a copy, i.e., $s(r)$. Then an update of all copies of x via a minimum spanning tree is initiated, i.e., $s(r)$ sends out one message that is transmitted along the branches of a minimum spanning tree connecting all nodes holding copies of

x. Thus, the update set E_{U_r} contains all edges on the shortest path from $h(r)$ to $s(r)$ and all edges of the minimum spanning tree which is used as an approximation to the minimum Steiner tree. Note that edges on the path between $h(r)$ and $s(r)$ can be contained twice in the multi-set E_{U_r}.

In order to show the quality of our solution we will compare it to an optimal solution that fulfills the following additional constraints.

1. A write request r for x first sends a message to $s(r)$ and then initiates the update of all copies of x via a multi-cast tree. All write requests for x use the same multi-cast tree T_x.

2. Each copy of x serves at least W requests, with $W = \sum_{v \in V} f_w(v, x)$ denoting the total number of write requests for x.

A placement fulfilling these constraints is called *restricted*.

The following lemma ensures that there exists a restricted placement that has close to optimal total cost.

Lemma 3.1 *Let OPT and* OPT_W *denote an optimal and optimal restricted placement, respectively. Then*

$$C_{\mathrm{OPT}_W} \leq 4 \cdot C_{\mathrm{OPT}} \ ,$$

where C_{OPT_W} *and* C_{OPT} *denote the total cost of* OPT_W *and* OPT, *respectively.*

Proof. Suppose the optimum placement OPT is given together with the optimal update sets for all requests. We will successively transform this placement into a restricted placement as follows. First, we replace each update set $E_{U_r}^{\mathrm{OPT}}$ of the optimum placement by the edge set of a minimum spanning tree (MST) connecting all copies plus the edge set of the path from $h(r)$ to $s(r)$. The following claim shows that the total cost of the placement is only doubled by this transformation.

Claim 3.2 *Let* OPT' *denote the new placement and* $E_{U_r}^{\mathrm{OPT}'}$ *its update set for request* r. *Then*

$$\sum_{e \in E_{U_r}^{\mathrm{OPT}'}} E_{U_r}^{\mathrm{OPT}'}(e) \cdot \mathrm{c_t}(e) \leq 2 \cdot \sum_{e \in E_{U_r}^{\mathrm{OPT}}} E_{U_r}^{\mathrm{OPT}}(e) \cdot \mathrm{c_t}(e) \ .$$

Proof. Let E_{ST} and E_{MST} denote the edge set of a minimum Steiner tree and a minimum spanning tree, respectively, connecting $h(r)$ with all nodes holding a copy of x, and let E_p denote the edge set of the path from $h(r)$ to $s(r)$. First, we show that $\sum_{e \in E_{\mathrm{MST}}} \mathrm{c_t}(e) \leq 2 \cdot \sum_{e \in E_{\mathrm{ST}}} \mathrm{c_t}(e) - \sum_{e \in E_p} \mathrm{c_t}(e)$. The following traversal of the Steiner tree visits each edge in E_p once and all other edges twice, i.e., in combination with the triangle inequality we get the above inequality. The traversal starts at $h(r)$. Then, it recursively visits, first, the neighbor nodes that are not incident to an edge in E_p, and, finally the single unvisited neighbor node that is incident to an edge in E_p. Thus, we can conclude

$$\begin{aligned}\sum_{e \in E_{U_r}^{\mathrm{OPT}'}} E_{U_r}^{\mathrm{OPT}'}(e) \cdot \mathrm{c_t}(e) &\leq \sum_{e \in E_{\mathrm{MST}}} \mathrm{c_t}(e) + \sum_{e \in E_{\mathrm{P}}} \mathrm{c_t}(e) \\ &\leq 2 \cdot \sum_{e \in E_{\mathrm{ST}}} \mathrm{c_t}(e) \\ &= 2 \cdot \sum_{e \in E_{U_r}^{\mathrm{OPT}}} E_{U_r}^{\mathrm{OPT}}(e) \cdot \mathrm{c_t}(e) \ ,\end{aligned}$$

which yields the claim. ■

In a second step the placement OPT′ is transformed into a restricted placement OPT_W as follows.

- As long as the set $\mathcal{C}_{<W}$ of copies that do not serve at least W requests is not empty, do the following:
  - Delete the copy $c \in \mathcal{C}_{<W}$ with maximum tree distance from the root node of the MST which is rooted at an arbitrary node. (The tree distance between nodes u and v is the length of the weighted unique path that connects u and v via edges of the MST.)
  - Each request previously assigned to c is reassigned to its nearest remaining copy in the network.

This algorithm terminates because the number of requests is larger than W and thus the last copy will not be deleted. Obviously, the resulting placement is *restricted* because each remaining copy serves at least W requests. It remains to show that the additional cost that incur by reassigning requests to other copies is small.

Each request that is reassigned from a copy on node v to another copy on node v' increases the total cost by $\mathrm{c_t}(h(r), v') - \mathrm{c_t}(h(r), v)$. Note that this holds for write requests as well.

Let v_f denote the father node of v according to the tree structure given by the MST. Each reassignment increases the cost by

$$\begin{aligned}\mathrm{c_t}(h(r), v') - \mathrm{c_t}(h(r), v) &\leq \mathrm{c_t}(h(r), v_f) - \mathrm{c_t}(h(r), v) \\ &\leq \mathrm{c_t}(v, v_f) \ .\end{aligned}$$

In the first step uses the fact that v' holds the copy closest to $h(r)$, and that v_f also holds a copy, because copies closer to the root node are deleted later. The second step holds because of the triangle inequality.

At most W requests are reassigned in a deletion step because otherwise the copy would not have to be deleted since it would have served more than W requests. The total cost increment caused by a deletion of a copy on node v is therefore at most $W \cdot \mathrm{c_t}(v, v_f)$. Summing this over all nodes holding a copy yields that the total cost increment is at most the cost expended by OPT′ for updating objects. Together with Claim 3.2 this yields the lemma. ■

In the remainder of this section, we only consider restricted placements. We split the total cost of such a placement into read, update, and storage cost which are defined as follows. For a given placement P, the *storage cost* C_P^{st} is defined as $C_P^{\mathrm{st}} := \sum_{v \in V \text{ holding a copy}} \mathrm{c_s}(v)$, the *update cost* C_P^{up} is defined as $C_P^{\mathrm{up}} := W \cdot \sum_{e \in E_{T_x}} \mathrm{c_t}(e)$, with E_{T_x} denoting the edge set of the multi-cast tree connecting all copies of x and $W = \sum_{v \in V} f_w(v, x)$ denoting the total number of write requests for x, and the *read cost* C_P^{rd} is defined as $C_P^{\mathrm{rd}} = \sum_{\text{request } r} \mathrm{c_t}(h(r), s(r))$. Furthermore, $C_P^{\mathrm{rd}}(\mathcal{S}) := \sum_{r \in \mathcal{S}} \mathrm{c_t}(h(r), s(r))$ denotes the read cost for a set $\mathcal{S}$ of requests.

Note that these definitions are only reasonable for restricted placements because the cost for a write request r (represented by the multi-set E_{U_r}) is partitioned into the cost for a multi-cast tree E_{T_x} and the cost for the path from $h(r)$ to $s(r)$. The latter cost is defined to belong to the read cost. By defining the read cost this way, we do not differentiate between read and write requests any more. The write requests are only represented by their total number W. Their exact location in the network does not influence the update cost. Obviously, for the *total cost* C_P of a restricted placement P holds $C_P = C_P^{\mathrm{rd}} + C_P^{\mathrm{up}} + C_P^{\mathrm{st}}$.

3.3.1 Proper placements

In this section, we define proper placements and prove some helpful properties of such placements. First, we introduce some notations and definitions. For each node v and $z \in \mathbb{N}$, let $\mathcal{R}_v^z$ denote the set of the z requests that are closest to v, with respect to the weighted distance $\mathrm{c_t}$. Further, let $d(v, z)$ denote the average weighted distance between v and the requests in $\mathcal{R}_v^z$, i.e., $d(v, z) := \frac{1}{z} \cdot \sum_{r \in \mathcal{R}_v^z} \mathrm{c_t}(h(r), v)$. For completeness, we define $d(v, 0) := 0$ and, for $z > \sum_{v \in V} f_r(v, x) + f_w(v, x)$, we define $d(v, z) := \infty$.

For each node v, we define the *write radius* $r_{\mathrm{w}}(v) := d(v, W)$, where W denotes the total number of write requests for x. Furthermore, we choose the *storage radius* $r_{\mathrm{s}}(v) \in \mathbb{R}_0^+$ and the *storage number* $z_{\mathrm{s}}(v) \in \mathbb{N}$ such that

$$\begin{aligned} (z_{\mathrm{s}}(v) - 1) \cdot r_{\mathrm{s}}(v) &\leq \mathrm{c_s}(v) \leq z_{\mathrm{s}}(v) \cdot r_{\mathrm{s}}(v) \quad \text{and} \\ d(v, z_{\mathrm{s}}(v) - 1) &\leq r_{\mathrm{s}}(v) \leq d(v, z_{\mathrm{s}}(v)) \ . \end{aligned}$$

This is done as follows. Obviously, $z_{\mathrm{s}}(v)$ can be chosen such that $(z_{\mathrm{s}}(v) - 1) \cdot d(v, z_{\mathrm{s}}(v) - 1) \leq \mathrm{c_s}(v) \leq z_{\mathrm{s}}(v) \cdot d(v, z_{\mathrm{s}}(v))$. Then $r_{\mathrm{s}}(v)$ is chosen from the interval

$[d(v, z_s(v) - 1), d(v, z_s(v))]$ such that the first inequality holds. If $z_s(v) = 1$, then there is the additional constraint $r_s(v) = d(v, z_s(v))$. Obviously, both inequalities also hold in this setting.

Informally speaking, the intuition of the definitions above is that the write radius and the storage radius of a node v give an indication of a suitable weighted distance from v to the nearest copy in a placement. This is formalized in the following. We call a placement *proper* if the copies are distributed according to the write and storage radii of the nodes as follows.

1. Every node v has a copy in weighted distance at most $k_1 \cdot \max\{r_w(v), r_s(v)\}$, where k_1 denotes a suitable constant.

2. Every pair of nodes u and v both holding a copy have at least weighted distance $2k_2 \cdot \max\{r_w(u), r_w(v)\}$, where k_2 denotes a suitable constant.

In the remaining part of this section, we show that every proper placement guarantees a constant approximation factor for the read and update cost. In Section 3.3.2, we will present an algorithm that calculates a proper placement with low storage cost. First, we derive a bound on the read cost of a proper placement.

Lemma 3.3 *For the read cost $C^{\mathrm{rd}}_{\mathrm{PRO}}$ of a proper placement* PRO *holds*

$$C^{\mathrm{rd}}_{\mathrm{PRO}} \leq (k_1 + 1) \cdot (C^{\mathrm{rd}}_{\mathrm{OPT}_W} + C^{\mathrm{st}}_{\mathrm{OPT}_W}) \enspace ,$$

where OPT_W *denotes an optimal restricted placement.*

Proof. In order to show the lemma, we compare the proper placement PRO to the optimal restricted placement OPT_W. Suppose a copy is placed on node v in OPT_W. Let $\mathcal{S}_v$ denote the set of requests served by this copy in OPT_W. We show that $C^{\mathrm{rd}}_{\mathrm{PRO}}(\mathcal{S}_v) \leq (k_1 + 1) \cdot (C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) + \mathrm{c_s}(v))$. Then the theorem follows immediately by summing over all request sets.

Let $r \in \mathcal{S}_v$ denote a request issued at node $h(r)$ and served at node $s(r)$ in PRO. The weighted distance between $h(r)$ and $s(r)$ can be estimated by

$$\begin{aligned} \mathrm{c_t}(h(r), s(r)) &\leq \mathrm{c_t}(h(r), v') \\ &\leq \mathrm{c_t}(h(r), v) + \mathrm{c_t}(v, v') \enspace , \end{aligned}$$

where v' denotes the node holding the copy nearest to v in PRO. The first step uses the fact that $s(r)$ is the copy closest to $h(r)$ in PRO and the second step holds due to the triangle inequality.

Then the read cost $C^{\mathrm{rd}}_{\mathrm{PRO}}(\mathcal{S}_v)$ of PRO can be bounded by

$$\begin{aligned} C^{\mathrm{rd}}_{\mathrm{PRO}}(\mathcal{S}_v) &= \sum_{r\in\mathcal{S}_v} \mathrm{c_t}(h(r), s(r)) \\ &\le \sum_{r\in\mathcal{S}_v} \mathrm{c_t}(h(r), v) + \sum_{r\in\mathcal{S}_v} \mathrm{c_t}(v, v') \\ &= C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v| \cdot \mathrm{c_t}(v, v') \\ &\le C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v| \cdot k_1 \cdot \max\{r_\mathrm{s}(v), r_\mathrm{w}(v)\} \ . \end{aligned}$$

Recall that $\mathrm{c_t}(v, v') \le k_1 \cdot \max\{r_\mathrm{s}(v), r_\mathrm{w}(v)\}$ due to the first property of a proper placement. Now, we distinguish two cases according to $\max\{r_\mathrm{s}(v), r_\mathrm{w}(v)\}$.

- Suppose $r_\mathrm{w}(v) = \max\{r_\mathrm{s}(v), r_\mathrm{w}(v)\}$.
  Obviously, $C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) \ge |\mathcal{S}_v| \cdot r_\mathrm{w}(v)$, since the copy on node v serves $|\mathcal{S}_v| \ge W$ requests in OPT_W. Then

$$\begin{aligned} C^{\mathrm{rd}}_{\mathrm{PRO}}(\mathcal{S}_v) &\le C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v| \cdot k_1 \cdot \max\{r_\mathrm{s}(v), r_\mathrm{w}(v)\} \\ &\le C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) + k_1 \cdot |\mathcal{S}_v| \cdot r_\mathrm{w}(v) \\ &\le (k_1 + 1) \cdot C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) \ . \end{aligned}$$

- Suppose $r_\mathrm{s}(v) = \max\{r_\mathrm{s}(v), r_\mathrm{w}(v)\}$.
  In this case, we distinguish two subcases according to the cardinality of $\mathcal{S}_v$.
  - Suppose $|\mathcal{S}_v| < z_\mathrm{s}(v)$.
    Then

$$\begin{aligned} C^{\mathrm{rd}}_{\mathrm{PRO}}(\mathcal{S}_v) &\le C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v| \cdot k_1 \cdot \max\{r_\mathrm{s}(v), r_\mathrm{w}(v)\} \\ &\le C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) + (z_\mathrm{s}(v) - 1) \cdot k_1 \cdot r_\mathrm{s}(v) \\ &\le C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) + k_1 \cdot \mathrm{c_s}(v) \ . \end{aligned}$$

    Recall that $(z_\mathrm{s}(v) - 1) \cdot r_\mathrm{s}(v) \le \mathrm{c_s}(v)$ by the definition of $z_\mathrm{s}(v)$ and $r_\mathrm{s}(v)$.
  - Suppose $|\mathcal{S}_v| \ge z_\mathrm{s}(v)$.
    Then

$$\begin{aligned} C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) &= |\mathcal{S}_v| \cdot d(v, |\mathcal{S}_v|) \\ &\ge |\mathcal{S}_v| \cdot d(v, z_\mathrm{s}(v)) \\ &\ge |\mathcal{S}_v| \cdot r_\mathrm{s}(v) \ . \end{aligned}$$

Thus, for the read cost of PRO holds

$$\begin{aligned} C^{\mathrm{rd}}_{\mathrm{PRO}}(\mathcal{S}_v) &\le C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v| \cdot k_1 \cdot \max\{r_{\mathrm{s}}(v), r_{\mathrm{w}}(v)\} \\ &\le C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) + |\mathcal{S}_v| \cdot k_1 \cdot r_{\mathrm{s}}(v) \\ &\le (k_1 + 1) \cdot C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) \ . \end{aligned}$$

Altogether this yields the lemma. ■

Finally, we derive a bound on the update cost of a proper placement.

Lemma 3.4 *For the update cost* $C^{\mathrm{up}}_{\mathrm{PRO}}$ *of a proper placement* PRO *holds*

$$C^{\mathrm{up}}_{\mathrm{PRO}} \le 2\left(\frac{k_2}{k_2 - 1} \cdot (C^{\mathrm{rd}}_{\mathrm{PRO}} + C^{\mathrm{rd}}_{\mathrm{OPT}_W}) + C^{\mathrm{up}}_{\mathrm{OPT}_W}\right) \ ,$$

where OPT_W *denotes an optimal restricted placement.*

Proof. First, we prove the following claim showing that every copy in the proper placement PRO serves at least a certain number of requests.

Claim 3.5 *Every copy in a proper placement serves at least* $(1 - \frac{1}{k_2}) \cdot W$ *requests.*

Proof. Suppose that a copy is placed on a node v in a proper placement. Each request r with $\mathrm{c_t}(h(r), v) \le k_2 \cdot r_{\mathrm{w}}(v)$ is served by the copy on v due to the second property of a proper placement. Now, the claim follows from a simple averaging argument.

Assume for contradiction that a copy on v serves less than $(1 - \frac{1}{k_2}) \cdot W$ requests. Then at least $W - (1 - \frac{1}{k_2}) \cdot W = \frac{W}{k_2}$ requests in $\mathcal{R}^W_v$ have a weighted distance to v larger than $k_2 \cdot r_{\mathrm{w}}(v)$. This yields

$$\begin{aligned} r_{\mathrm{w}}(v) &= \frac{1}{|\mathcal{R}^W_v|} \cdot \sum_{r \in \mathcal{R}^W_v} \mathrm{c_t}(h(r), v) \\ &> \frac{1}{W} \cdot \frac{W}{k_2} \cdot k_2 \cdot r_{\mathrm{w}}(v) \\ &= r_{\mathrm{w}}(v) \ , \end{aligned}$$

which is a contradiction. ■

Suppose that a copy is placed on a node v in PRO and that the copy nearest to v in OPT_W is placed on v'. For a request r that is served by v in PRO, let $s(r)$ denote the node serving r in OPT_W. Then

$$\begin{aligned} \mathrm{c_t}(v, v') &\le \mathrm{c_t}(v, s(r)) \\ &\le \mathrm{c_t}(v, h(r)) + \mathrm{c_t}(h(r), s(r)) \ . \end{aligned}$$

The first step follows since v' holds the copy closest to v in OPT_W and the second step holds due to the triangle inequality.

Let $\mathcal{S}_v$ denote the set of requests served by v in PRO. Summing the above inequality over all requests in $\mathcal{S}_v$ yields

$$\begin{aligned} |\mathcal{S}_v| \cdot \mathrm{c_t}(v, v') &= \sum_{r \in \mathcal{S}_v} \mathrm{c_t}(v, v') \\ &\le \sum_{r \in \mathcal{S}_v} \mathrm{c_t}(v, h(r)) + \sum_{r \in \mathcal{S}_v} \mathrm{c_t}(h(r), s(r)) \\ &= C^{\mathrm{rd}}_{\mathrm{PRO}}(\mathcal{S}_v) + C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v) \ . \end{aligned}$$

Now, we compare the update cost of PRO and OPT_W. The additional cost in PRO caused by the update messages for the copy on v is less than

$$\begin{aligned} W \cdot \mathrm{c_t}(v, v') &\le \frac{k_2}{k_2 - 1} \cdot |\mathcal{S}_v| \cdot \mathrm{c_t}(v, v') \\ &\le \frac{k_2}{k_2 - 1} \cdot (C^{\mathrm{rd}}_{\mathrm{PRO}}(\mathcal{S}_v) + C^{\mathrm{rd}}_{\mathrm{OPT}_W}(\mathcal{S}_v)) \ . \end{aligned}$$

Recall for the first step that $|\mathcal{S}_v| \ge (1 - \frac{1}{k_2}) \cdot W$ due to Claim 3.5.

Summing this inequality over all nodes holding a copy in PRO and taking the cost for updating all copies of OPT_W into account yields that there exists a Steiner tree for updating all copies of PRO with cost at most $\frac{k_2}{k_2-1} \cdot (C^{\mathrm{rd}}_{\mathrm{PRO}} + C^{\mathrm{rd}}_{\mathrm{OPT}_W}) + C^{\mathrm{up}}_{\mathrm{OPT}_W}$. Since we use a minimum spanning tree for updating, $C^{\mathrm{up}}_{\mathrm{PRO}} \le 2(\frac{k_2}{k_2-1} \cdot (C^{\mathrm{rd}}_{\mathrm{PRO}} + C^{\mathrm{rd}}_{\mathrm{OPT}_W}) + C^{\mathrm{up}}_{\mathrm{OPT}_W})$ which yields the lemma. ∎

3.3.2 The approximation algorithm

In this section, we present the algorithm that computes a proper placement with low storage cost. The algorithm consists of the following three phases.

1. An initial placement is calculated by an approximation algorithm for the facility location problem. The input is the *related facility location problem*, i.e., the same input as for our data management problem with the difference that all write requests become read requests. Hence, in this phase the update cost is neglected.

2. Additional copies are added to the initial placement. As long as there exists a node v whose nearest copy has a weighted distance to v larger than $5 \cdot r_{\mathrm{s}}(v)$, a new copy is stored on v.

3. Copies that violate the second property of a proper placement are deleted in the following way. All nodes holding a copy are scanned in ascending order according to their write radii. When node v is processed according to this order, a copy on node u is deleted if $c_t(u, v) \leq 4 \cdot r_w(u)$.

The following theorem shows that the above algorithm calculates a constant approximation. Algorithm and proof are not optimized to obtain a minimal approximation factor, but to keep them as simple and clear as possible.

Theorem 3.6 *The placement calculated by the above algorithm achieves a constant approximation factor for the static data management problem.*

Proof. First, we show that the algorithm computes a proper placement, i.e., we can conclude with lemmas 3.3 and 3.4 that the read and update cost of this placement are optimal up to a constant factor. Finally, we show that the storage cost of this placement is also optimal up to a constant factor.

Lemma 3.7 *The above algorithm calculates a placement fulfilling the following properties.*

- *Every node v has a copy in weighted distance at most $29 \cdot \max\{r_w(v), r_s(v)\}$ from v. This is the first property of a proper placement with $k_1 = 29$.*
- *Every pair of nodes u and v both holding a copy has at least weighted distance $4 \cdot \max\{r_w(u), r_w(v)\}$. This is the second property of a proper placement with $k_2 = 2$.*

Proof. First, we show that the second property holds. Let u and v denote two nodes both holding a copy. Assume for contradiction that $c_t(u, v) \leq 4 \cdot r_w(u)$. Obviously, the copy on u would have been deleted in the third phase of the algorithm, during the scan of node v.

In order to show the first property, we make the following observation. If a node u holds a copy after the second phase of the algorithm, there exists a node holding a copy in the final placement with weighted distance at most $4 \cdot r_w(u)$ to u. Assume that the copy on node u is deleted in phase 3 during the scan of node v that holds a copy. Thus $c_t(u, v) \leq 4 \cdot r_w(u)$. Now, assume that the copy on node v is deleted later in phase 3 during the scan of some node v' that holds a copy. Hence, $c_t(v, v') \leq 4 \cdot r_w(v) \leq 4 \cdot r_w(v')$. The last inequality holds since the nodes are considered in ascending order according to their write radii. But in this case the copy on node v' would have been already deleted during the scan of node v, since $c_t(v', v) \leq 4 \cdot r_w(v')$. This is a contradiction.

Now, we bound the weighted distance from a node u to its closest copy in the network. Let v and v' denote the nodes holding the closest copy after the second and third phase of the algorithm, respectively. Then

$$\begin{aligned} c_t(u, v') &\leq c_t(u, v) + c_t(v, v') \\ &\leq 5 \cdot r_s(u) + 4 \cdot r_w(v) \enspace , \end{aligned}$$

because of the above observation and the triangle inequality.

To get the desired result we have to relate $r_{\mathrm{w}}(v)$ to $r_{\mathrm{w}}(u)$. Obviously, $r_{\mathrm{w}}(v) \leq \mathrm{c_t}(v,u) + r_{\mathrm{w}}(u)$, since $r_{\mathrm{w}}(v)$ denotes the average weighted distance between v and the set of those W distinct requests that are closest to v. Thus,

$$\begin{aligned} \mathrm{c_t}(u,v') &\leq 5 \cdot r_{\mathrm{s}}(u) + 4 \cdot r_{\mathrm{w}}(v) \\ &\leq 25 \cdot r_{\mathrm{s}}(u) + 4 \cdot r_{\mathrm{w}}(u) \ . \end{aligned}$$

Hence, with $k_1 = 29$ and $k_2 = 2$ we get a proper placement. ■

It remains to derive a bound for the storage cost of the placement calculated by the above algorithm.

Lemma 3.8 *The storage cost of the placement calculated by the above algorithm is at most* $f \cdot (C^{\mathrm{st}}_{\mathrm{OPT}_W} + C^{\mathrm{rd}}_{\mathrm{OPT}_W})$*, where* f *denotes the approximation ratio of the facility location algorithm used in the first phase.*

Proof. Let C^{st}_i and C^{rd}_i denote the storage and read cost, respectively, of the placement after the ith phase of the algorithm. Further, let $C^{\mathrm{st}}_{\mathrm{FLP}}$ and $C^{\mathrm{rd}}_{\mathrm{FLP}}$ denote the optimum storage and read cost, respectively, of the related facility location problem. Then

$$\begin{aligned} C^{\mathrm{st}}_3 &\leq C^{\mathrm{st}}_2 \\ &\leq C^{\mathrm{st}}_2 + C^{\mathrm{rd}}_2 \\ &\leq C^{\mathrm{st}}_1 + C^{\mathrm{rd}}_1 \\ &\leq f \cdot (C^{\mathrm{st}}_{\mathrm{FLP}} + C^{\mathrm{rd}}_{\mathrm{FLP}}) \\ &\leq f \cdot (C^{\mathrm{st}}_{\mathrm{OPT}_W} + C^{\mathrm{rd}}_{\mathrm{OPT}_W}) \ . \end{aligned}$$

The first inequality holds, since during the third phase copies are only deleted. The third inequality will be shown in Claim 3.9. The last two inequalities are obvious.

Claim 3.9 *The sum of storage and read cost does not increase during the second phase of the algorithm, i.e.,* $C^{\mathrm{st}}_2 + C^{\mathrm{rd}}_2 \leq C^{\mathrm{st}}_1 + C^{\mathrm{rd}}_1$.

Proof. In order to show the claim we need the following observation.

Observation 3.10 *In weighted distance at most* $2 \cdot r_{\mathrm{s}}(v)$ *from a node* v*, there are at least* $\lceil z_{\mathrm{s}}(v)/2 \rceil$ *requests.*

Proof. We call a request r with $\mathrm{c_t}(h(r), v) \leq 2 \cdot r_{\mathrm{s}}(v)$ *close to* v. Assume for contradiction that there are less than $\lceil z_{\mathrm{s}}(v)/2 \rceil$ close requests. In this case at most $\lceil z_{\mathrm{s}}(v)/2 \rceil - 1$

requests are close to v. Therefore, at least $(z_s(v) - 1) - (\lceil z_s(v)/2 \rceil - 1) = \lfloor z_s(v)/2 \rfloor$ requests from the set $\mathcal{R}_v^{z_s(v)-1}$ are not close. Recall that $|\mathcal{R}_v^{z_s(v)-1}| = z_s(v) - 1$. Then

$$\begin{aligned} d(v, z_s(v) - 1) &> \frac{1}{z_s(v) - 1} \cdot \left(\left\lfloor \frac{z_s(v)}{2} \right\rfloor \cdot 2 \cdot r_s(v) \right) \\ &\geq r_s(v) \\ &\geq d(v, z_s(v) - 1) \ , \end{aligned}$$

which is a contradiction. ■

We will show that the cost always decreases whenever a copy is placed on a node v during the second phase. Before a new copy is added every copy has weighted distance larger than $5 \cdot r_s(v)$ from v. Hence, the $\lceil z_s(v)/2 \rceil$ closest requests to v have weighted distance larger than $3 \cdot r_s(v)$ to the nearest copy. Therefore the read cost for these requests decreases by at least

$$\begin{aligned} &\lceil z_s(v)/2 \rceil \cdot 3 \cdot r_s(v) - \lceil z_s(v)/2 \rceil \cdot d(v, \lceil z_s(v)/2 \rceil) \\ \geq \quad & \lceil z_s(v)/2 \rceil \cdot 3 \cdot r_s(v) - \lceil z_s(v)/2 \rceil \cdot r_s(v) \\ = \quad & \lceil z_s(v)/2 \rceil \cdot 2 \cdot r_s(v) \\ \geq \quad & z_s(v) \cdot r_s(v) \\ \geq \quad & c_s(v) \ . \end{aligned}$$

The above inequalities follow directly from the choice of $z_s(v)$ and $r_s(v)$. Since the decrease in read cost is larger than the storage cost $c_s(v)$ for the new copy on v, the sum of read and storage cost does not increase during the second phase of the algorithm, which yields the claim. ■

This completes the proof of the lemma. ■

Thus, the theorem follows from lemmas 3.3, 3.4, 3.7, and 3.8. ■

3.4 The optimal algorithm for trees

In this section, we present an algorithm that calculates an optimal placement for the static data management problem on an arbitrary rooted tree $T = (V, E)$. Our algorithm places all objects independently from each other. Thus, fix an object x.

This section is organized as follows. First, we present an algorithm for the read-only case, i.e., $f_w(v, x) = 0$, for all $v \in V$.While this special case can be solved by using the uncapacitated facility location algorithm of Tamir [Tam96] for trees, we present a new algorithm which we then adapt to solve the general case of reads and writes.

3.4.1 The read-only case

In the following, we present an algorithm for the read-only case, i.e., $f_w(v, x) = 0$, for all $v \in V$. Let diam(T) denote the *unweighted diameter* of T, i.e., the maximum number of edges on a path connecting two arbitrary nodes in T, and let deg(T) denote its *maximum node degree*. For a node v, let T_v denote the subtree rooted at v, i.e., the connected component containing v if the edge incident to v and its father is removed. Then let $|T_v|$ denote the number of nodes in T_v.

A *placement* P for a subtree T_v consists of a set $\mathcal{C}$ of nodes holding copies, a set $\mathcal{R}_P^{\mathrm{as}}$ of requests assigned to copies in T_v and a set $\mathcal{R}_P^{\mathrm{out}}$ of outgoing requests, i.e., requests that are not assigned to any copy in T_v. The *cost of* P is defined as $\mathrm{cost}(P) := \sum_{u \in \mathcal{C}} \mathrm{c_s}(u) + \sum_{r \in \mathcal{R}_P^{\mathrm{as}}} \mathrm{c_t}(h(r), s(r)) + \sum_{r \in \mathcal{R}_P^{\mathrm{out}}} \mathrm{c_t}(h(r), v)$. Observe that this definition of the cost of P is not equal to the cost for all requests issued in T_v plus the cost for the copies stored in T_v, which would be the straightforward definition. The advantage of our definition is that the cost of P does not depend on the placement in $T \setminus T_v$. This allows us to use a dynamic programming approach to compute the optimal cost for T.

For this approach we need the following additional notations and definitions. For a placement P of a subtree T_v, we define the *copy distance* d_P as the weighted distance from node v to the closest copy in T_v. If there is no copy in T_v then we define $d_P := \infty$. A placement P of a subtree T_v is called *naturally assigned* if it fulfills the following conditions.

- If a request r is assigned to a copy in T_v then this is the closest copy to $h(r)$.
- All requests that pass a node are either assigned to the same copy in T_v or belong all to the set of outgoing requests.

Obviously, there always exists an optimal placement for T_v that is naturally assigned.

Our tree algorithm is based on the key observation that the optimal placement for a subtree T_v does not too heavily depend on the placement decisions made for $T \setminus T_v$. In fact it only depends on a few parameters. Thus, only a restricted number of placements for T_v have to be considered for calculating the optimal placement for T. This observation can be formalized as follows. We call a set $\mathcal{S}_{T_v}$ of placements for T_v *sufficient* if in each naturally assigned placement for T the placement for T_v can be replaced by a placement from $\mathcal{S}_{T_v}$ without increasing the total cost. Hence, only the placements in $\mathcal{S}_{T_v}$ have to be considered by an algorithm that searches for an optimal placement for T.

Lemma 3.11 *For any subtree T_v exists a sufficient set $\mathcal{S}_{T_v}$ with $|\mathcal{S}_{T_v}| \leq 2 \cdot |T_v| + 1$.*

Proof. In order to show the lemma we first derive some characteristics of placements that are contained in the sufficient set. For each $D \in \mathbb{R}_0^+$ we define an *optimal* *export placement E_v^D for weighted distance D* as

$$E_v^D := \arg \min_{\text{placement } P} \{\text{cost}(P) + |\mathcal{R}_P^{\text{out}}| \cdot D\} \enspace .$$

Furthermore, for each $R \in \mathbb{N}_0$ we define an *optimal import placement I_v^R for request-quantity R* as

$$I_v^R := \arg \min_{\text{placement } P} \{\text{cost}(P) + d_P \cdot R\} \enspace .$$

We choose the set $\mathcal{S}_{T_v}$ such that it contains an optimal export placement E_v^D for each $D \in \mathbb{R}_0^+$ and an optimal import placement I_v^R for each $R \in \mathbb{N}_0$. We have to show that this can be done with only $2 \cdot |T_v| + 1$ different placements and that by this definition $\mathcal{S}_{T_v}$ is a sufficient set.

First, we prove that $\mathcal{S}_{T_v}$ is sufficient. Suppose that we are given a naturally assigned placement P for T. Let P_{T_v} denote the corresponding subplacement for T_v. We have to replace the placement P_{T_v} by one of the placements from $\mathcal{S}_{T_v}$ without increasing the cost. We distinguish the following two cases: some requests issued in $T \setminus T_v$ are served in T_v and some requests issued in T_v are served in $T \setminus T_v$. Note that no other case can occur in a naturally assigned placement.

- Suppose that R requests issued in $T \setminus T_v$ are served in T_v.

  If the subplacement P_{T_v} is replaced by I_v^R, then the total cost changes by $\text{cost}(I_v^R) - \text{cost}(P_{T_v}) + R \cdot (d_{I_v^R} - d_{P_{T_v}}) \le 0$, according to the definition of I_v^R.

- Suppose that requests issued in T_v are served in $T \setminus T_v$ by a copy that has weighted distance D from v.

  If the subplacement P_{T_v} is replaced by E_v^D, then the total cost changes by $\text{cost}(E_v^D) - \text{cost}(P_{T_v}) + D \cdot (|\mathcal{R}_{E_v^D}^{\text{out}}| - |\mathcal{R}_{P_{T_v}}^{\text{out}}|) \le 0$, according to the definition of E_v^D.

Finally, we prove that $|\mathcal{S}_{T_v}| \le 2 \cdot |T_v| + 1$. First of all, $|T_v|$ placements suffice in order to contain all placements $I_v^R = \arg\min_{\text{placement } P}\{\text{cost}(P) + d_P \cdot R\}$, for each $R \in \mathbb{N}_0$, since the copy distance d_P from v to its nearest copy in T_v can only take $|T_v|$ distinct values, and for every value of d_P there is one optimal placement.

To prove that $|T_v| + 1$ placements suffice in order to contain all placements $E_v^D = \arg\min_{\text{placement } P}\{\text{cost}(P) + |\mathcal{R}_P^{\text{out}}| \cdot D\}$, for each $D \in \mathbb{R}_0^+$, we show in the following that $|\mathcal{R}_P^{\text{out}}|$ can only take $|T_v| + 1$ distinct values for placements that minimize $\text{cost}(P) + |\mathcal{R}_P^{\text{out}}| \cdot D$.

Let D_1 and $D_2 \in \mathbb{R}_0^+$ with $D_1 < D_2$. Define the placements P_1 and P_2 with $P_i := \arg\min_{\text{placement } P}\{\text{cost}(P) + |\mathcal{R}_P^{\text{out}}| \cdot D_i\}$, for each $i \in \{1, 2\}$. Further, we define $C_i := \text{cost}(P_i) + |\mathcal{R}_{P_i}^{\text{out}}| \cdot D_i$. All requests of a node are either assigned to the same copy or belong to the set of outgoing requests, since P_1 and P_2 are naturally assigned.

The term $|\mathcal{R}_P^{\text{out}}|$ can take only $|T_v| + 1$ different values because if the requests of a node u are assigned to a copy in P_1, then these requests are assigned to a copy in P_2, as well. This will be shown in the following. Assume for contradiction that the requests of u are assigned to a copy in P_1 and belong to the set of outgoing requests in P_2. We call a subtree $T_w \subseteq T_v$ *self-contained* if no request issued in T_w is assigned to a copy in $T_v \setminus T_w$ or belongs to the set of outgoing requests, and if no request issued in $T_v \setminus T_w$ is assigned to a copy in T_w. Obviously there exists a self-contained subtree T_w that contains u in the placement P_1. In placement P_2 the subplacement of T_w has changed such that the requests from u belong to the set of outgoing requests.

We show that by exchanging the different subplacements of T_w between the placements P_1 and P_2 either C_1 or C_2 could be reduced which is a contradiction to the definition of P_1 and P_2. Let P_1^w and P_2^w denote the subplacement used for T_w in the placement P_1 and P_2, respectively. If the subplacement P_2^w would be used in the placement P_1 the term C_1 would change by

$$\text{cost}(P_2^w) + |\mathcal{R}_{P_2^w}^{\text{out}}| \cdot (d(v, w) + D_1) - \text{cost}(P_1^w) \ .$$

Similarly, if the subplacement P_1^w would be used in the placement P_2 then C_2 would change by

$$\text{cost}(P_1^w) - |\mathcal{R}_{P_2^w}^{\text{out}}| \cdot (d(v, w) + D_2) - \text{cost}(P_2^w) \ .$$

Obviously, one of the above terms is smaller then 0, since $D_1 < D_2$. This is a contradiction.

Altogether, we need only $2 \cdot |T_v| + 1$ placements in order to guarantee that $\mathcal{S}_{T_v}$ contains an optimal import placement I_v^R for each $R \in \mathbb{N}_0$ and an optimal export placement E_v^D for each $D \in \mathbb{R}_0^+$. ■

Now, we give a short sketch of the algorithm that computes an optimal placement for T. Let v_r denote the root node of T, i.e., $T = T_{v_r}$. The algorithm recursively computes the relevant parameters of the placements belonging to the sufficient set $\mathcal{S}_{T_{v_r}}$. The *relevant parameters of a placement* P_v for T_v are parameters that are used for computing the total cost of a placement for $T_w \supset T_v$ that uses P_v as subplacement for the subtree T_v. If P_v is an import placement the relevant parameters are the cost of P_v, its copy distance, and the node holding the closest copy to v in T_v. If P_v is an export placement the relevant parameters are the cost of P_v, its number of outgoing requests, and an *optimality interval* $\mathcal{I}_{P_v}$, i.e., for $D \in \mathcal{I}_{P_v}$, the placement P_v is the optimal export placement E_v^D. Finally, the cost of an optimal placement for T is the cost of $E_{v_r}^{\infty}$.

Theorem 3.12 *The optimal placement for an arbitrary tree T can be computed in time $O(|T| \cdot \text{diam}(T) \cdot \log(\deg(T)))$.*

Proof. First, we restrict ourselves to binary trees. The following lemma gives an upper bound on the time needed to compute the relevant parameters of all placements in the sufficient set of a node, if the parameters of the placements, in the sufficient sets of the children of this node are given. Finally, we generalize our results to arbitrary trees.

Lemma 3.13 *For a binary subtree T_v of T the relevant parameters of all placements in the sufficient set $\mathcal{S}_{T_v}$ can be computed in time $O(|T_v|)$ if the relevant parameters of the placements belonging to the sufficient sets of v's children are given.*

Proof. We code the set $\mathcal{S}_{T_v}$ by a sequence of at most $|T_v|$ import and $|T_v| + 1$ export tuples that describe the corresponding optimal import and export placements of $\mathcal{S}_{T_v}$. An *import tuple* (C_P, d_P, v_P) consists of the cost C_P of the corresponding placement P, the node v_P holding the closest copy to v in T_v and the copy distance $d_P := \mathrm{c_t}(v, v_P)$. An *export tuple* $(C_P, |\mathcal{R}_P|, \mathcal{I}_P)$ consists of the cost C_P of the corresponding placement P, the number of outgoing requests $|\mathcal{R}_P|$ and an optimality interval $\mathcal{I}_P$.

Each sequence of import tuples is sorted according to their copy distances and each sequence of export tuples is sorted according to their optimality intervals. Note that the latter is possible, since, obviously, the optimality intervals do not intersect. Now, we describe how to construct these sequences for a subtree T_v.

Suppose that v is a leaf node. For an import placement the subtree T_v must contain a copy. Thus a copy has to be placed on v. The corresponding import tuple is $(\mathrm{c_s}(v), 0, v)$. For an export placement we distinguish two cases according to the weighted distance D to a copy within $T \setminus T_v$.

- Suppose that $D < \frac{\mathrm{c_s}(v)}{f_r(v)}$.
  Then the optimal export placement has no copy on v. The corresponding export tuple is $(0, f_r(v), [0, \ldots, \frac{\mathrm{c_s}(v)}{f_r(v)}))$.
- Suppose that $D \geq \frac{\mathrm{c_s}(v)}{f_r(v)}$.
  Then the optimal export placement has a copy on v. The corresponding export tuple is $(\mathrm{c_s}(v), 0, [\frac{\mathrm{c_s}(v)}{f_r(v)}, \ldots, \infty))$.

Now, suppose that v is an inner node. Let v_1 and v_2 denote the two children of v and define edge $e_1 = (v_1, v)$ and edge $e_2 = (v_2, v)$. First, we describe how to construct the tuples for the optimal import placements of $\mathcal{S}_{T_v}$.

Claim 3.14 *The sorted sequence of import tuples of T_v can be computed in time $O(|T_v|)$ if the sorted sequences of import and export tuples of T_{v_1} and T_{v_2} are given.*

Proof. In the proof of Lemma 3.11 it is shown that for each node $u \in T_v$ there exists at most one optimal import placement in which u holds the closest copy to v in T_v. In the following we will denote this placement with I_v^u. This notation will coexist with the notation I_v^R for the optimal import placement with request-quantity R. Note that $I_v^u = I_v^R$ for R in a certain, possibly empty interval. We will construct an import tuple for every placement I_v^u.

We start with $u = v$, i.e., the tuple $(C_{I_v^v}, d_{I_v^v}, v)$ of the optimal import placement I_v^v. The copy distance $d_{I_v^v}$ is 0. Furthermore, no requests enter the subtrees T_{v_1} and T_{v_2}. Thus, the cost $C_{I_v^v}$ is $\mathrm{c_s}(v) + C_1 + \mathrm{c_t}(e_1) \cdot |\mathcal{R}_1| + C_2 + \mathrm{c_t}(e_2) \cdot |\mathcal{R}_2|$ where $(C_1, |\mathcal{R}_1|, \mathcal{I}_1)$ is the tuple of $E_{v_1}^{\mathrm{c_t}(e_1)}$ and $(C_2, |\mathcal{R}_2|, \mathcal{I}_2)$ is the tuple of $E_{v_2}^{\mathrm{c_t}(e_2)}$.

Now assume without loss of generality that $u \in T_{v_1}$. Obviously, in this case no request enters the subtree T_{v_2}, because the placements are naturally assigned. Thus, we can construct the optimal import placement I_v^u by combining the placements $I_{v_1}^u$ for T_{v_1} and $E_{v_2}^{\mathrm{c_t}(u,v_2)}$ for T_{v_2}. The only problem is that all tuples corresponding to some I_v^u, with $u \in T_{v_1}$, have to be computed in time $O(|T_v|)$. In order to do this we exploit that the tuples of $I_{v_1}^u$ and $E_{v_2}^D$ are sorted. We traverse the sequence of import tuples in order of increasing copy distance and search for the required export tuple in the sequence of T_{v_2}. Since the tuples in this sequence are sorted in order of their optimality intervals, we can use linear search and start each search at the position the previous search stopped. Thus, we need only time $O(|T_{v_1}| + |T_{v_2}|) = O(|T_v|)$ for computing all I_v^u with $u \in T_{v_1}$. After all tuples have been computed for each $u \in T_{v_1}$ and each $u \in T_{v_2}$, there are two sorted sequences of import tuples. These sequences have to be merged to get a sorted sequence for T_v. This can be done in time $O(|T_v|)$. ■

Now, we describe how to construct the tuples of the optimal export placements of $\mathcal{S}_{T_v}$.

Claim 3.15 *The sorted sequence of export tuples of T_v can be computed in time $O(|T_v|)$ if the sorted sequences of import and export tuples of T_{v_1} and T_{v_2} are given.*

Proof. We start with the export tuple $(C_{E_v^\infty}, |\mathcal{R}_{E_v^\infty}|, I_{E_v^\infty})$ of the optimal export placement E_v^∞. The number of outgoing requests $|\mathcal{R}_{E_v^\infty}|$ is 0, due to the cost function $\mathrm{cost}(E_v^\infty) + |\mathcal{R}_{E_v^\infty}| \cdot \infty$ of E_v^∞. Since $|\mathcal{R}_{E_v^\infty}| = 0$ this placement can be viewed as an import placement with request-quantity 0. The optimal cost for this placement, i.e., $\mathrm{cost}(I_v^0)$, can be computed due to Claim 3.14. Thus, the cost $C_{E_v^\infty}$ is $\mathrm{cost}(I_v^0)$. The optimality interval $I_{E_v^\infty}$ for this tuple will be computed later.

All other optimal export placements have some outgoing requests. Obviously, in such a placement no request enters the subtrees T_{v_1} and T_{v_2}, because the placements are naturally assigned. Therefore each remaining optimal export placement can be obtained by combining two optimal export placements, one of T_{v_1} and one of T_{v_2}. More formally the export tuple for E_v^D can be constructed from the combination of the export tuples for $E_{v_1}^{D+\mathrm{c_t}(e_1)}$ and $E_{v_2}^{D+\mathrm{c_t}(e_2)}$.

In order to compute all required combinations in time $O(|T_v|)$ we first shift the optimality intervals of all export tuples of T_{v_1} and T_{v_2} by $-\mathrm{c_t}(e_1)$ and $-\mathrm{c_t}(e_2)$, respectively. Then the sequences of export tuples of T_{v_1} and T_{v_2} are traversed in increasing order of their shifted optimality intervals. If the shifted optimality intervals of one export tuple $(C_1, |\mathcal{R}_1|, \mathcal{I}_1)$ of T_{v_1} and one export tuple $(C_2, |\mathcal{R}_2|, \mathcal{I}_2)$ of T_{v_2} intersect, they are combined for a new export tuple $(C, |\mathcal{R}|, \mathcal{I})$ of T_v as follows. The optimality interval $\mathcal{I}$ is simply the intersection of the two shifted optimality intervals $\mathcal{I}_1$ and $\mathcal{I}_2$, the cost C is $C_1 + C_2 + |\mathcal{R}_1| \cdot \mathrm{c_t}(e_1) + |\mathcal{R}_2| \cdot \mathrm{c_t}(e_2)$, and the number of outgoing requests $|\mathcal{R}|$ is $|\mathcal{R}_1| + |\mathcal{R}_2| + f_r(v)$.

Finally, it remains to determine the optimality interval for the export tuple of E_v^∞ and to adjust the sequence of export tuples, accordingly. This is done as follows. For each export tuple $E = (C_E, |\mathcal{R}_E|, \mathcal{I}_E)$ with $|\mathcal{R}_E| > 0$ corresponding not to E_v^∞ we compute the weighted distance D_E for which $C_E + D_E \cdot |\mathcal{R}_E| = \mathrm{cost}(E_v^\infty)$. If D_E is smaller than the lower bound of $\mathcal{I}_E$ the tuple E is deleted because the corresponding export placement is not optimal. Obviously, there exists exactly one tuple $E^* = (C_{E^*}, \mathcal{R}_{E^*}, \mathcal{I}_{E^*})$ with $D_{E^*} \in \mathcal{I}_{E^*}$. The upper bound of the optimality interval $\mathcal{I}_{E^*}$ is set to D_E^*. Further, the optimality interval of the export tuple corresponding to E_v^∞ is set to $[D_{E^*}, \infty)$.

Altogether, all these computations can be done in time $O(|T_v|)$ and give the sorted sequence of export tuples of T_v. ■

Claims 3.14 and 3.15 yield the lemma. ■

Applying this lemma the relevant parameters of an optimal placement OPT on a binary tree T can be computed in time $O(\sum_v |T_v|) = O(|T| \cdot \mathrm{diam}(|T|))$. To compute the whole placement we use the fact that for any tree T_v there is at most one export and one import tuple indicating that v holds a copy. Thus, if we know which tuple from T_v is used to compute the cost of OPT, we know whether v holds a copy in OPT.

This is done as follows. For every tuple $\mathcal{T}$ we memorize from which tuples $\mathcal{T}_1$ and $\mathcal{T}_2$ it is constructed. This can easily be realized by assigning two pointers to $\mathcal{T}$ pointing to $\mathcal{T}_1$ and $\mathcal{T}_2$. Finally, every tuple used to compute the cost of OPT can be reconstructed by following the pointers starting at the tuple with optimal cost.

Obviously, an arbitrary tree T can be simulated on a binary tree with $O(|T|)$ nodes and diameter $O(\mathrm{diam}(T) \cdot \log(\deg(T)))$, which yields the overall running time of $O(|T| \cdot \mathrm{diam}(T) \cdot \log(\deg(T)))$ for the computation of an optimal placement for an arbitrary tree T. ■

3.4.2 The general case

In the following, we show how to adapt the algorithm of the previous section to the general case where nodes can issue read and write requests. A write request issued by node v increases the total cost by $\sum_{e \in E_{\mathrm{ST}}} \mathrm{c_t}(e)$ where E_{ST} denotes the edge set of the minimum Steiner tree connecting v with all nodes holding a copy. The *write cost* $\text{cost}_{\text{wr}}(P)$ of a placement P for a subtree $T_v = (V_{T_v}, E_{T_v})$ is defined as the cost due to write messages along edges in T_v, i.e., the write request issued by v increases $\text{cost}_{\text{wr}}(P)$ by $\sum_{e \in E_{\text{ST}} \cap E_{T_v}} \text{c}_\text{t}(e)$.

Unfortunately, this definition of the write cost of a placement for T_v depends on the placement for $T \setminus T_v$. More precisely, it depends on whether at least one copy is placed in $T \setminus T_v$ or not. Let $\text{cost}^1_{\text{wr}}(P)$ and $\text{cost}^0_{\text{wr}}(P)$ denote the write cost of P under the condition that at least one or no copy is placed in $T \setminus T_v$, respectively. These definitions are independent of the placement in $T \setminus T_v$ and either $\text{cost}_{\text{wr}}(P) = \text{cost}^0_{\text{wr}}(P)$ or $\text{cost}_{\text{wr}}(P) = \text{cost}^1_{\text{wr}}(P)$.

Adopting the notations and definitions from the previous section we call a set $\mathcal{S}_{T_v}$ *sufficient*, if in each naturally assigned placement of T the placement of T_v can be replaced by a placement in $\mathcal{S}_{T_v}$ without increasing the total cost. Now, we show that there exists a sufficient set with small cardinality.

Lemma 3.16 *For any subtree T_v exists a sufficient set $\mathcal{S}_{T_v}$ with $|\mathcal{S}_{T_v}| \leq 3 \cdot |T_v| + 2$.*

Proof. Analogous to the proof of Lemma 3.11, we first characterize the placements that are contained in the sufficient set. For each $D \in \mathbb{R}^+_0$, the set $\mathcal{S}_{T_v}$ contains a placement

$$E^D_v := \arg \min_{\text{placement } P} \{\text{cost}(P) + \text{cost}^1_{\text{wr}}(P) + |\mathcal{R}^{\text{out}}_P| \cdot D\}$$

which is called the *optimal export placement for weighted distance D*. In addition, the set $\mathcal{S}_{T_v}$ contains the *optimal export placement E_v* in which no node of T_v holds a copy. Furthermore, for each $R \in \mathbb{N}_0$, the set $\mathcal{S}_{T_v}$ contains a placement

$$I^{0,R}_v := \arg \min_{\text{placement } P} \{\text{cost}(P) + \text{cost}^0_{\text{wr}}(P) + d_P \cdot R\}$$

which is called the *optimal 0-import placement for request-quantity R* and a placement

$$I^{1,R}_v := \arg \min_{\text{placement } P} \{\text{cost}(P) + \text{cost}^1_{\text{wr}}(P) + d_P \cdot R\}$$

which is called the *optimal 1-import placement for request-quantity R*.

Now, we will prove that $\mathcal{S}_{T_v}$ is sufficient. Suppose that we are given a naturally assigned placement P for T. Let P_{T_v} denote the corresponding subplacement for T_v. We have to replace the placement P_{T_v} by one of the placements from $\mathcal{S}_{T_v}$ without increasing the cost. The proof is similar to the proof of Lemma 3.11. In the following we only list the different cases and specify what placement from $\mathcal{S}_{T_v}$ has to be selected for each case. It is easy to verify for each case that the respective selection does not increase the total cost.

The different cases are as follows:

- Some requests issued in $T \setminus T_v$ are served in T_v and $T \setminus T_v$ contains at least one copy.

  Replace P_{T_v} by $I_v^{1,R}$ for the appropriate value of R.
- Some requests issued in $T \setminus T_v$ are served in T_v and $T \setminus T_v$ contains no copy.

  Replace P_{T_v} by $I_v^{0,R}$ for the appropriate value of R.
- Some requests issued in T_v are served in $T \setminus T_v$ and T_v contains at least one copy.

  Replace P_{T_v} by E_v^D for the appropriate value of D.
- All requests issued in T_v are served in $T \setminus T_v$ and hence, T_v contains no copy.

  Then P_{T_v} equals E_v, i.e., the placement in which no node of T_v holds a copy.

Finally, we prove that $|\mathcal{S}_{T_v}| \leq 3 \cdot |T_v| + 2$. First of all, $2 \cdot |T_v|$ placements suffice in order to contain all placements $I_v^{0,R} = \arg\min_{\text{placement } P}\{\text{cost}(P) + \text{cost}^0_{\text{wr}}(P) + d_P \cdot R\}$ and $I_v^{1,R} = \arg\min_{\text{placement } P}\{\text{cost}(P) + \text{cost}^1_{\text{wr}}(P) + d_P \cdot R\}$, for each $R \in \mathbb{N}_0$, since the copy distance d_P from v to its nearest copy in T_v can only take $|T_v|$ distinct values, and for every value of d_P there is one optimal placement.

The proof that $|T_v| + 1$ placements suffice in order to contain all placements $E_v^D = \arg\min_{\text{placement } P}\{\text{cost}(P) + \text{cost}^1_{\text{wr}}(P) + |\mathcal{R}_P^{\text{out}}| \cdot D\}$, for each $D \in \mathbb{R}_0^+$, is analogous to the proof of Lemma 3.11. Adding the placement E_v gives $|\mathcal{S}_{T_v}| \leq 3 \cdot |T_v| + 2$. ■

Now, we give a short sketch of the algorithm that computes an optimal placement for T. Let v_r denote the root node of T, i.e., $T = T_{v_r}$. The algorithm recursively computes the relevant parameters of the placements belonging to the sufficient set $\mathcal{S}_{T_{v_r}}$. The *relevant parameters of a placement P_v for T_v* are parameters that are used for computing the total cost of a placement for $T_w \supset T_v$ that uses P_v as subplacement for the subtree T_v. If P_v is an i-import placement with $i \in \{0, 1\}$ the relevant parameters are the costs $\text{cost}(P_v)$ and $\text{cost}^i_{\text{wr}}(P_v)$, the copy distance d_{P_v}, and the node v_{P_v} holding the closest copy to v in T_v. If P_v is an export placement the relevant parameters are the costs $\text{cost}(P_v)$ and $\text{cost}^1_{\text{wr}}(P_v)$, the number of outgoing requests $|\mathcal{R}_{P_v}^{\text{out}}|$, and an optimality interval $\mathcal{I}_{P_v}$. Finally, the cost of an optimal placement for T is the cost of $E_{v_r}^{\infty}$.

To compute the whole placement we use the fact that for any tree T_v there are at most one export tuple and at most two import tuples indicating that v holds a copy. Thus, if we know which tuple from T_v is used to compute the cost of OPT, we know whether v holds a copy in OPT.

Theorem 3.17 *The optimal placement for an arbitrary tree T can be computed in time $O(|T| \cdot \text{diam}(T) \cdot \log(\text{deg}(T)))$.*

$\mathcal{S}_{T_v}$	$\mathcal{S}_{T_{v_1}}$	$\mathcal{S}_{T_{v_2}}$	copy on v
E_v	E_{v_1}	E_{v_2}	no
E_v^D	$E_{v_1}^{D+c_t(v,v_1)}$	$E_{v_2}^{D+c_t(v,v_2)}$	no
$I_v^{0,v} = I_v^{1,v}$	$E_{v_1}^{c_t(v,v_1)}$ E_{v_1} $E_{v_1}^{c_t(v,v_1)}$ E_{v_1}	$E_{v_2}^{c_t(v,v_2)}$ $E_{v_2}^{c_t(v,v_2)}$ E_{v_2} E_{v_2}	yes
$I_v^{0,u}$ with $u \in T_{v_1}$	$I_{v_1}^{0,u}$	E_{v_2}	no
$I_v^{1,u}$ with $u \in T_{v_1}$	$I_{v_1}^{1,u}$	$E_{v_2}^{c_t(u,v_2)}$	no
$I_v^{0,u}$ with $u \in T_{v_2}$	E_{v_1}	$I_{v_2}^{0,u}$	no
$I_v^{1,u}$ with $u \in T_{v_2}$	$E_{v_1}^{c_t(u,v_1)}$	$I_{v_2}^{1,u}$	no

Table 3.1: Combinations of placements from $\mathcal{S}_{T_{v_1}}$ and $\mathcal{S}_{T_{v_2}}$ that have to be considered for computing all placement in $\mathcal{S}_{T_v}$.

Proof. The proof is analogous to the proof of Theorem 3.12. We sketch the algorithm for binary trees. Let v denote an internal node of T and let v_1 and v_2 denote the children of v. The algorithm computes the relevant parameters for the placements in $\mathcal{S}_{T_v}$ by combining relevant parameters for placements from $\mathcal{S}_{T_{v_1}}$ and $\mathcal{S}_{T_{v_2}}$. This has to be done in time $O(|T_v|)$.

One approach would be to first combine each subplacement from $\mathcal{S}_{T_{v_1}}$ with each placement from $\mathcal{S}_{T_{v_2}}$ and then, to choose the placements which belong to $\mathcal{S}_{T_v}$ from this collection of placements. But then the algorithm would have to consider $2 \cdot |\mathcal{S}_{T_{v_1}}| \cdot |\mathcal{S}_{T_{v_2}}|$ different placements for T_v. The factor of 2 is due to the fact that the algorithm can decide to place a copy on v or not. This could not be done in time $O(|T_v|)$.

Therefore, the algorithm exploits the fact that only few combinations of subplacements have to be considered. Table 3.1 contains the combinations of subplacements that are tested for a placement in $\mathcal{S}_{T_v}$. In this table $I_v^{i,u}$ denotes the optimal i-import placement, with $i \in \{0, 1\}$, in which the node $u \in T_v$ holds the closest copy to v in T_v. Obviously, there is only one optimal placement with this property for each $u \in T_v$.

Table 3.1 shows that for each placement from $\mathcal{S}_{T_v}$ only a constant number of combinations have to be computed. In order to prove that the algorithm needs time $O(|T_v|)$ for computing the relevant parameters for the placements in $\mathcal{S}_{T_v}$ it remains to show that each combination can be computed in amortized constant time. This part of the proof is similar to the proof of Theorem 3.12 and is omitted since it gives no new algorithmic insights. ■

3.5 Conclusions

We have devised static data management strategies that aim to minimize the monetary cost induced by a placement of shared objects in a distributed system. Our strategies for tree networks calculate the optimal placement in polynomial time via dynamic programming. For general topologies we presented an algorithm that achieves a constant approximation ratio with respect to the cost-optimal solution.

For future work it would be very interesting to extend our cost-based model to a dynamic scenario in which, analogous to the model of Chapter 2, the read and write requests are not known in advance but arrive online during the running time of a parallel application. If storing of data is for free, i.e., $c_s(v) = 0$ for all nodes $v \in V$, this model is equivalent to models that minimize the total communication load in the network (see, e.g., [ABF93]).

However, if storing of data is not for free, completely new aspects have to be considered by a data management strategy. A reasonable approach is, e.g., to charge the fee for storing data with respect to the amount of utilized memory and with respect to the time this memory is used. In such a scenario, it is not sufficient to simply model the requests via a sequence (i.e., to specify only the order among requests), but it is necessary to model "time" in some way, as this is required for calculating the fee. Furthermore, copies of shared object may have to be migrated even if there is no current request to the corresponding object, because copies may have to be moved to a "cheaper" place. These issues make the development of cost-based data management strategies in dynamic scenarios a very challenging task.

Further interesting modifications of our model are, e.g., the inclusion of memory capacity constraints and the development of distributed algorithms that only use local information.

For tree networks it has been shown (see [SF02]) that a dynamic programming approach, very similar to our strategy, enables a facility location algorithm with running time $O(n \log n)$. It is interesting whether a similar approach can be used to improve the running time for k-median algorithms on trees, as well. Currently, the best known algorithm (see [Tam96]) for k-median has running time of $O(k \cdot n^2)$, which gives a fairly large gap between the fastest facility location algorithm and the fastest k-median algorithm. Perhaps this gap can be narrowed by transferring our techniques for facility location to the k-median problem. A key problem for such a transfer is the notion of a *sufficient set* as defined on page 79. It has to be shown that also for the k-median there is always a sufficient set with small cardinality. This is more involved than in the case of facility location, since the restriction of at most k open facilities in the whole tree, increases the number of placements that have to be considered for a subtree, as the optimal placement depends on the number of facilities that may be opened.

Bibliography

[AA95] B. Awerbuch and Y. Azar. Competitive multicast routing. *Wireless Networks*, 1:107–114, 1995.

[AAF$^+$97] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997. Also in *Proc. 25th ACM STOC*, 1993, pp. 623–631.

[ABF93] B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*, pages 164–173, 1993.

[ABF98] B. Awerbuch, Y. Bartal, and A. Fiat. Distributed paging for general networks. *Journal of Algorithms*, 28(1):67–104, 1998. Also in *Proc. 7th ACM-SIAM SODA*, 1996, pp. 574–583.

[AC03] D. Applegate and E. Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs. In *Proceedings of the ACM Symposium on Communications Architectures & Protocols (SIGCOMM)*, 2003.

[ACF$^+$03] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke. Optimal oblivious routing in polynomial time. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC)*, pages 383–388, 2003.

[AL94] B. Awerbuch and T. Leighton. Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks. In *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC)*, pages 487–496, 1994.

[ALMN91] W. A. 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. Also in *Proc. 2nd ACM SPAA*, 1990, pp. 55–64.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[AR98] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998.

[BA91] M. Baumslag and F. S. Annexstein. A unified framework for off-line permutation routing in parallel networks. *Mathematical Systems Theory*, 24(4):233–251, 1991. Also in *Proc. 2nd ACM SPAA*, 1990, pp. 398–406.

[BFR95] Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. *Journal of Computer and System Sciences*, 51(3):341–358, 1995. Also in *Proc. 24th ACM STOC*, 1992, pp. 39–50.

[BH85] 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.

[BKR03] M. Bienkowski, M. Korzeniowski, and H. Räcke. A practical algorithm for constructing oblivious routing schemes. In *Proceedings of the 15th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 24–33, 2003.

[BL97] Y. Bartal and S. Leonardi. On-line routing in all-optical networks. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 516–526, 1997.

[BR01] I. Baev and R. Rajaraman. Approximation algorithms for data placement in arbitrary networks. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 661–670, 2001.

[CG99] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 378–388, 1999.

[CKS01] I. Cidon, S. Kutten, and R. Soffer. Optimal allocation of electronic content. In *Proceedings of the 20th IEEE INFOCOM*, pages 1773–1780, 2001.

[CV02] R. Carr and S. Vempala. Randomized metarounding. *Random Structures and Algorithms*, 20(3):343–352, 2002. Also in *Proc. 32nd ACM STOC*, 2000, pp. 58-62.

[DF82] D. Dowdy and D. Foster. Comparative models of the file assignment problem. *Computing Surveys*, 14(2):287–313, 1982.

[FK00] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 105–115, 2000.

[GK98] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 300–309, 1998.

[GK99] S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. *Journal of Algorithms*, 31(1):228–248, 1999.

[GLS88] B. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer Verlag, New York, 1988.

[HHR03] C. Harrelson, K. Hildrum, and S. Rao. A polynomial-time tree decomposition to minimize congestion. In *Proceedings of the 15th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 34–43, 2003.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58(301):13–30, 1963.

[JGN99] W. E. Johnston, D. Gannon, and B. Nitzberg. Grids as production computing environments: The engineering aspects of nasa's information power grid. In *Proceedings of the 8thIEEE Symposium on High Performance Distributed Computing*, 1999.

[JMS02] K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC)*, pages 731–740, 2002.

[KKT90] C. Kaklamanis, D. Krizanc, and A. 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.

[KPR93] P. Klein, S. A. Plotkin, and S. Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the 25th ACM Symposium on Theory of Computing (STOC)*, pages 682–690, 1993.

[KPR98] M. Korupolu, G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facilty location problems. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1–10, 1998.

[KRW03] C. Krick, H. Räcke, and M. Westermann. Approximation algorithms for data management in networks. *Theory of Computing Systems*, 2003. to appear.

[Leo98] S. Leonardi. On-line network routing. In A. Fiat and G. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *LNCS*, pages 242–267. Springer, 1998.

[LLR95] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. Also in *Proc. 35th IEEE FOCS*, 1994, pp. 577-591.

[LMR94] F. T. Leighton, B. M. Maggs, and S. B. Rao. Packet routing and job-shop scheduling in o(congestion+dilation) steps. *Combinatorica*, 14(2):167–180, 1994.

[LMRR94] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao. Randomized routing and sorting on fixed-connection networks. *Journal of Algorithms*, 17(1):157–205, 1994.

[LR99] F. T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999. Also in *Proc. 29th IEEE FOCS*, 1988, pp. 422–431.

[MMP$^+$03] B. M. Maggs, G. L. Miller, O. Parekh, R. Ravi, and S. L. M. Woo. Solving symmetric diagonally-dominant systems by preconditioning. manuscript, 2003.

[MMVW97] 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.

[MVW99] F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Provably good and practical strategies for non-uniform data management in networks. In *Proceedings of the 7th European Symposium on Algorithms (ESA)*, pages 89–100, 1999.

[MVW00] F. Meyer auf der Heide, B. Vöcking, and M. Westermann. Caching in networks. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 430–439, 2000.

[MYZ02] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 292–242, 2002.

[OR97] R. Ostrovsky and Y. Rabani. Universal o(congestion + dilation + $\log^{1+epsilon} n$) local control packet switching algorithms. In *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, pages 644–653, 1997.

[Räc02] H. Räcke. Minimizing congestion in general networks. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 43–52, 2002.

[RT87] P. Raghavan and C. D. Thompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7:365–374, 1987.

[Sch98] C. Scheideler. *Universal Routing Strategies for Interconnection Networks*. Springer Verlag, Heidelberg, 1998.

[SF02] R. Shah and M. Farach-Colton. Undiscretized dynamic programming: Faster algorithms for facility location and related problems on trees. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 108–115, 2002.

[ST85] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[ST00] A. Srinivasan and C. Teo. A constant-factor approximation algorithm for packet routing and balancing local vs. global criteria. *SIAM Journal on Computing*, 30(6):2051–2068, 2000. Also in *Proc. 29th ACM STOC*, 1997, pp. 636-643.

[STA97] D. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, pages 265–274, 1997.

[SV98] C. Scheideler and B. Vöcking. Universal continuous routing strategies. *Theory of Computing Systems*, 31(4):425–449, 1998. Also in *Proc. 8th ACM SPAA*, 1996, pp. 142–151.

[Tam96] A. Tamir. An $o(pn^2)$ algorithm for the p-median and related problems on tree graphs. *Operations Research Letters*, 19:59–94, 1996.

[UW87] E. Upfal and A. Wigderson. How to share memory in a distributed system. *Journal of the ACM*, 34(1):116–127, 1987.

[VB81] 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.

[Vöc98] Berthold Vöcking. *Static and Dynamic Data Management in Networks*. PhD thesis, Universität Paderborn, 1998.

[Wes00] Matthias Westermann. *Caching in Networks: Non-Uniform Algorithms and Memory Capacity Constraints*. PhD thesis, Universität Paderborn, 2000.

[WM91] O. Wolfson and A. Milo. The multicast policy and its relationship to replicated data placement. *ACM Transactions on Data Base Systems (TODS)*, 16(1):181–205, 1991.

Appendix

A.1 Proof of Claim 2.16 on page 45

Claim A.18 $\dfrac{\mathrm{out}(U^*)}{w_{\ell+1}(U^*)} \leq 4\lambda \cdot \dfrac{\mathrm{cap}(A,B)}{w_{\ell+1}(A)}$.

Proof. Let ℓ denote the level of the cluster H. In order to prove the claim we have to relate $\mathrm{cap}(A,B)$ to $\mathrm{out}(U^*)$. We do this by successively relating $\mathrm{cap}(A,B)$ to $\mathrm{w}_\ell(U^* \cap A)$, $\mathrm{w}_\ell(U^* \setminus A)$, and $\mathrm{cap}(U^*, H \setminus U^*)$. Since $\mathrm{out}(U^*) = \mathrm{w}_\ell(U^* \cap A) + \mathrm{w}_\ell(U^* \setminus A) + \mathrm{cap}(U^*, H \setminus U^*)$ (see Figure A.1) we get the desired relation.

Firstly, we can exploit that H fulfills the precondition and $|A| \leq \frac{1}{2}|H|$. This gives

$$\lambda \cdot \mathrm{cap}(A,B) \geq \mathrm{w}_\ell(A) \geq \mathrm{w}_\ell(U^* \cap A) \ , \tag{A.1}$$

which gives a lower bound on $\mathrm{cap}(A,B)$ in terms of $\mathrm{w}_\ell(U^* \cap A)$.

For deriving the other two relations we observe that the subclusters H_i also fulfill the precondition. Further, from the definition of I_S and I_L it follows that $A_i \leq \frac{3}{4}|H_i|$ for $i \in I_S$, and $B_i \leq \frac{3}{4}|H_i|$ for $i \in I_L$. Therefore we can apply the precondition for set A_i if H_i has a small intersection with A, and for B_i if H_i has a large intersection with A (and, hence, a small intersection with B). This gives the following inequalities.

$$\forall i \in I_S \quad (\lambda + 1) \cdot \mathrm{cap}(A_i, B_i) \geq w_{\ell+1}(A_i) + \mathrm{cap}(A_i, B_i) = \mathrm{out}(A_i) \tag{$*$}$$

$$\forall i \in I_L \quad (\lambda + 1) \cdot \mathrm{cap}(A_i, B_i) \geq w_{\ell+1}(B_i) + \mathrm{cap}(A_i, B_i) = \mathrm{out}(B_i) \tag{$**$}$$

We will utilize these inequalities to relate $\mathrm{cap}(A,B)$ to $\mathrm{w}_\ell(U^* \setminus A)$ and $\mathrm{cap}(U^*, H \setminus U^*)$, respectively.

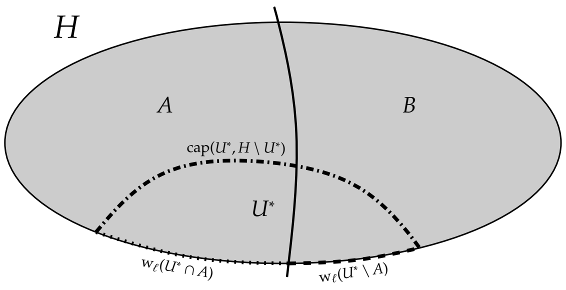

Figure A.1: The edges leaving U^* either contribute to $\mathrm{w}_\ell(U^* \cap A)$, $\mathrm{w}_\ell(U^* \setminus A)$, or $\mathrm{cap}(U^*, H \setminus U^*)$.

First, we obtain,

$$\begin{aligned}(\lambda+1)\cdot \mathrm{cap}(A,B) &\geq (\lambda+1)\cdot \sum_{i\in I_L} \mathrm{cap}(A_i,B_i) + (\lambda+1)\cdot \sum_{i\in I_S} \mathrm{cap}(A_i,B_i)\\ &\geq \sum_{i\in I_L} \mathrm{w}_{\ell+1}(B_i) + \sum_{i\in I_S} \mathrm{out}(A_i) \ .\end{aligned}$$

Utilizing $\mathrm{w}_{\ell+1}(B_i) \geq \sum_{i\in I_L} \mathrm{w}_\ell(B_i) = \mathrm{w}_\ell(\uplus_{i\in I_L} B_i) = \mathrm{w}_\ell(U^* \setminus A)$, gives

$$(\lambda+1)\cdot \mathrm{cap}(A,B) \geq \mathrm{w}_\ell(U^* \setminus A) + \sum_{i\in I_S} \mathrm{out}(A_i) \ , \tag{A.2}$$

which relates $\mathrm{cap}(A,B)$ to $\mathrm{w}_\ell(U^* \setminus A)$.

Now, we derive a relation between $\mathrm{cap}(A,B)$ and $\mathrm{cap}(U^*, H \setminus U^*)$:

$$\begin{aligned}\mathrm{cap}(U^*, H\setminus U^*) &= \mathrm{cap}\left(\biguplus_{i\in I_L} H_i, \biguplus_{i\in I_S} H_i\right)\\ &= \mathrm{cap}\left(\biguplus_{i\in I_L} A_i, \biguplus_{i\in I_S} B_i\right) + \mathrm{cap}\left(\biguplus_{i\in I_L} A_i, \biguplus_{i\in I_S} A_i\right)\\ &\qquad + \mathrm{cap}\left(\biguplus_{i\in I_L} B_i, \biguplus_{i\in I_S} H_i\right)\\ &\leq \mathrm{cap}(A,B) + \sum_{i\in I_S} \mathrm{out}(A_i) + \sum_{i\in I_L} \mathrm{out}(B_i) \ .\end{aligned}$$

Applying inequalities $(*)$ and $(**)$ to $\mathrm{out}(A_i)$ and $\mathrm{out}(B_i)$, respectively, gives the following relation between $\operatorname{cap}(A,B)$ and $\operatorname{cap}(U^*, H \setminus U^*)$:

$$\begin{aligned}\operatorname{cap}(U^*, H \setminus U^*) &\le \operatorname{cap}(A,B) + (\lambda+1)\cdot\sum_{i\in I_S}\operatorname{cap}(A_i,B_i) + (\lambda+1)\cdot\sum_{i\in I_L}\operatorname{cap}(A_i,B_i)\\ &\le \operatorname{cap}(A,B) + (\lambda+1)\cdot\operatorname{cap}(A,B)\\ &= (\lambda+2)\cdot\operatorname{cap}(A,B)\end{aligned} \tag{A.3}$$

Now, we combine equations (A.1), (A.2), and (A.3) in order to obtain the relation between $\operatorname{cap}(A,B)$ and $\operatorname{out}(U^*)$.

$$\begin{aligned}\operatorname{out}(U^*) + \sum_{i\in I_S}\operatorname{out}(A_i) &\le \operatorname{cap}(U^*, H\setminus U^*) + \mathrm{w}_\ell(U^*\cap A)\\ &\quad + \mathrm{w}_\ell(U^*\setminus A) + \sum_{i\in I_S}\operatorname{out}(A_i)\\ &\le \operatorname{cap}(A,B)\,((\lambda+2)+\lambda+(\lambda+1))\\ &\le 4\lambda\cdot\operatorname{cap}(A,B)\end{aligned} \tag{A.4}$$

Here, we utilized $\lambda \ge 3$ for the last step.

Finally, we are able to estimate the ratio $\frac{\mathrm{w}_{\ell+1}(A)}{\operatorname{cap}(A,B)}$, as needed for the claim.

$$\begin{aligned}\frac{\mathrm{w}_{\ell+1}(A)}{\operatorname{cap}(A,B)} &= \frac{\sum_i \operatorname{cap}(A_i,\overline{H_i})}{\operatorname{cap}(A,B)}\\ &= \frac{\sum_{i\in I_L}\operatorname{cap}(A_i,\overline{H_i}) + \sum_{i\in I_S}\operatorname{cap}(A_i,\overline{H_i})}{\operatorname{cap}(A,B)}\\ &\le 4\lambda\cdot\frac{\sum_{i\in I_L}\operatorname{cap}(A_i,\overline{H_i}) + \sum_{i\in I_S}\operatorname{cap}(A_i,\overline{H_i})}{\operatorname{out}(U^*) + \sum_{i\in I_S}\operatorname{out}(A_i)}\quad.\end{aligned}$$

So far, we have utilized Equation (A.4), i.e., the relation between $\operatorname{cap}(A,B)$ and $\operatorname{out}(U^*)$. Now, we further exploit that $\operatorname{cap}(A_i,\overline{H_i}) \le \operatorname{out}(A_i)$ and that $\operatorname{cap}(A_i,\overline{H_i}) \le \operatorname{cap}(H_i,\overline{H_i})$. We get

$$\frac{\mathrm{w}_{\ell+1}(A)}{\operatorname{cap}(A,B)} \le 4\lambda\cdot\frac{\sum_{i\in I_L}\operatorname{cap}(H_i,\overline{H_i}) + \sum_{i\in I_S}\operatorname{out}(A_i)}{\operatorname{out}(U^*) + \sum_{i\in I_S}\operatorname{out}(A_i)}\quad,$$

which can be further simplified to

$$\begin{aligned}\frac{\mathrm{w}_{\ell+1}(A)}{\mathrm{cap}(A,B)} &= 4\lambda \cdot \frac{\mathrm{w}_{\ell+1}(U^*) + \sum_{i \in I_S} \mathrm{out}(A_i)}{\mathrm{out}(U^*) + \sum_{i \in I_S} \mathrm{out}(A_i)} \\ &\leq 4\lambda \cdot \frac{w_{\ell+1}(U^*)}{\mathrm{out}(U^*)} \ ,\end{aligned}$$

where the last inequality follows since $\mathrm{w}_{\ell+1}(U^*) \geq \mathrm{out}(U^*)$. This finishes the proof of the claim. ■