UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Power-Aware Online File Allocation in Dynamic Networks

Dissertation

by

Jan Mehler

Faculty of Computer Science, Electrical Engineering and Mathematics
Department of Computer Science and Heinz Nixdorf Institute
University of Paderborn, Germany

July 2010

Zusammenfassung

Sowohl die Vernetzung von mobilen drahtlosen Geräten wie Smartphones und PDAs als auch die Verbreitung von Sensornetzwerken nimmt zur Zeit stark zu. Eine wesentliche Anforderung an solche mobilen ad hoc Netzwerke besteht darin, den Netzwerkknoten eine gemeinsame Nutzung von Daten zu ermöglichen. Beim von Bartal eingeführten File Allocation Problem hat ein Datenverwaltungssystem die Möglichkeit nach Bedarf beliebig viele Kopien eines Datums auf den Knoten des Netzwerks zu erzeugen und auch wieder zu löschen. Da die Knoten eines mobilen ad hoc Netzwerks in der Regel nur eine stark beschränkte Energiereserve besitzen, besteht unser Ziel darin Algorithmen zu entwickeln, die den bei der Bedienung einer Folge von Lese- und Schreibanfragen der Netzwerkknoten anfallenden Energiebedarf, möglichst gering halten. Um dies zu erreichen muss ein Algorithmus Kopien so im Netzwerk platzieren, dass sie zwar möglichst nahe an den zugreifenden Knoten liegen, aber gleichzeitig eine Aktualisierung aller Kopien nicht zu teuer wird. Wir verallgemeinern das File Allocation Problem von Bartal auf Netzwerke, die sich mit der Zeit verändern. Dabei besteht eine wesentliche Herausforderung darin, dass weder bekannt ist welche Anfragen in Zukunft gestellt werden noch wie sich das Netzwerk verändern wird. Wir untersuchen die Qualität verschiedener online Algorithmen für das File Allocation Problem in dynamischen Netzwerken sowohl theoretisch als auch mittels simulationsbasierter Experimente.

Reviewers:

- **Prof. Dr. Friedhelm Meyer auf der Heide, University of Paderborn, Germany**
- **Prof. Dr. Christian Scheideler, University of Paderborn, Germany**

Acknowledgments

First of all, I would like to thank my supervisor Professor Friedhelm Meyer auf der Heide for giving me the opportunity to write this thesis and for his persistent optimism and encouragements. I especially appreciate that he gave me the freedom to do my work in my own style and pace.

I want to thank all the members of the working group "Algorithms and Complexity" I encountered during my time in Paderborn for providing a great working environment. I will miss especially the enjoyable lunch breaks.

In particular, I would like to thank Bastian Degener for several good advices and for making my life more adventurous. Furthermore, I want to thank Peter Pietrzyk for proofreading my dissertation and especially for his everlasting moral support.

Paderborn, July 2010 *Jan Mehler*

Contents

Notation

Frequently Used Variables and Notations

$G = (V, E)$	A (weighted) graph consisting of nodes V and edges $E \subseteq V^2$.
i, j, k	Nodes of a network.
c	Center node of a star network.
$d_t(e)$	Weight of edge e at time t.
d_t^i	Distance of node i to the center c at time t in a star network.
$d_t(i, j)$	The distance between node i and node j at time t.
$d_t(R, S)$	The distance between $R \subseteq V$ and $S \subseteq V$ at time t.
$\mathcal{ST}(R)$	A minimum Steiner tree for $R \subseteq V$.
$st_t(R)$	The weight of a minimum Steiner tree for $R \subseteq V$ at time t.
D	File size.
p	Stand-by costs of a participating node in one step.
L	Period of validity of a lease.
δ	Maximum change of an edge weight in one step.
$\sigma = (d_t, a_t)_t$	An input sequence.
a_t	Data access request in step t.
Alg	A (randomized) algorithm.
Det	A deterministic algorithm.
Adv	An adversary.
Opt	An optimal offline algorithm.
R^t_{Alg}	Replica set of algorithm Alg at the end of step t.
$C_{\text{Alg}}(\sigma)$	The costs incurred by Alg on serving input sequence σ.

CHAPTER 1

Introduction

The availability of lightweight mobile devices equipped with wireless transceivers significantly increases since the beginning of the 21st century. The first generation of these devices could not communicate directly with each other. They could only communicate with stationary base stations which were connected to a wired network. So, they were completely dependent on a wired communication infrastructure. By now, more and more devices are able to communicate directly with other devices within their communication range. This allows to form widespread networks without the necessity of a costly and inflexible stationary infrastructure. Such networks are called *mobile ad hoc networks* (*MANET*).

A MANET consists of a set of mobile wireless nodes equipped with wireless (e.g. radio or infrared) transceivers [CM99]. Examples for such networks are ad hoc networks formed by wireless gadgets (e.g. PDAs, netbooks, mobile phones) or wireless sensor networks. A wireless sensor network consists of a large number of spatially distributed autonomous devices which gather data from their environment and intelligently forward relevant data for analysis [ASSC02].

Common to all MANETs is that every node has its own power source and that the transmission range of the nodes is limited [CM99]. In most cases power will be provided by a battery with a limited amount of energy. Therefore, the scarcity of energy constitutes a major challenge for the operation of MANETs. In the majority of cases the wireless communication constitutes a significant part of the energy consumption of the wireless nodes. The limited transmission range implies that data transfers necessarily have to use multi-hop routing paths. A data transfer between two nodes therefore induces not only a power consumption in the sending respectively receiving nodes but also in every hop of the used path. Thus, every service provided in a MANET should try to minimize the amount of transferred data.

We investigate the problem of providing access to a shared file (e.g. a database)

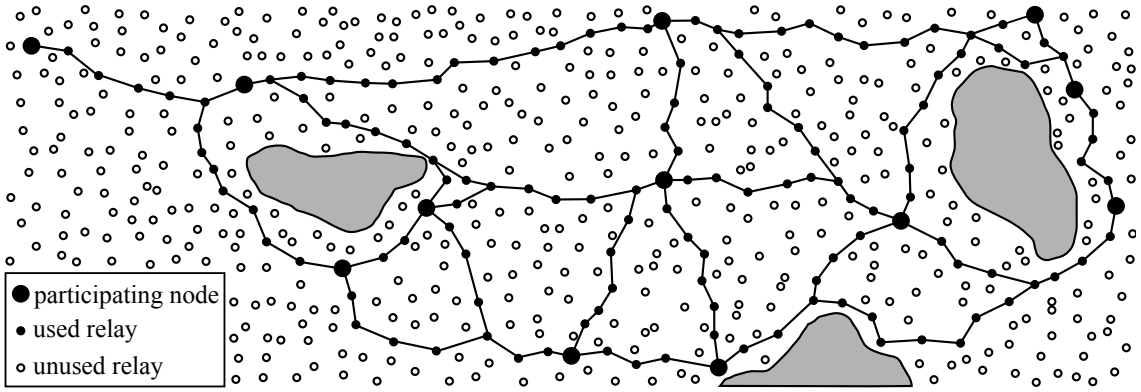


Figure 1.1: A set of nodes shares a file in a MANET. The participating nodes are interconnected via paths of relays.

to a subset of the nodes of a MANET. We call this subset of nodes the *set of participating nodes*. The file consists of a fixed number of data units and can be stored on any non-empty subset of the participating nodes. All the participating nodes of the MANET can read and modify the file. Every such access affects only a single unit of the file. In order to avoid data transfers, an algorithm can create/delete additional copies of the file and place them on the participating nodes. A read request is fulfilled either by a copy of the file which was prior placed on the requesting node or by a data transfer from one of the available copies. A write request needs to update all the copies of the file. Our goal is to minimize the overall energy consumption of the nodes of the MANET induced by the data management system.

The nodes of the MANET can have two different roles in the data management system (cf. Figure 1.1): participating nodes and relays. The participating nodes are interested in the contents of the shared file. They can issue read and write requests to the file and can hold a copy of the file. We assume that during a run of the data management system the set of participating nodes is fixed.[1] All the nodes of the MANET (including the participating nodes) are relays. They are used for data transfers between the participating nodes. Which relays are actually used is determined by the routing protocol of the MANET. In this thesis we do not deal with a concrete routing protocol. We assume that an arbitrary routing protocol, which is capable of routing messages between any two connected nodes, is implemented in the MANET (e.g. LLS [ADM04] and MLS [FW06]). Relays are typically not aware of their involvement in the data management system.

The described scenario induces two major challenges for the design of data

[1] Our results can easily be adapted to the case that mobile nodes can join and leave the set of participating nodes.

management strategies. Firstly, a data management strategy has to decide online where and when to place copies, i.e. without knowing what requests are issued in the future. Secondly, it has to take the mobility of the nodes of the MANET into account.

1.1 The file allocation problem in static networks

The *file allocation problem* in static networks introduced by Bartal et al. [BFR95] is defined as follows. A single indivisible file consisting of several data units is shared by a set of nodes (e.g. processors, computers) which are interconnected by a static network, i.e. neither the topology nor the edge weights change over time. Each node can access (read/write) the file and can hold a copy of the file. The input of the file allocation problem consists of a sequence of read/write accesses issued by the nodes of the network. It is assumed that these accesses arrive sequential and that each access affects only a single unit of the file. If a node issues a read request, the requested data has to be send from one of the nodes holding a copy to the requesting node. Thus, the incurred cost is defined by the distance of the reading node to the closest copy of the file. If a node modifies the file, all copies have to be updated. Thus, the cost of a write request is the weight of a minimum Steiner tree spanning all the copies and the writing node. In order to minimize the overall costs, an algorithm can create and delete copies between consecutive requests. Creating a new copy means that the whole file has to be transferred from a node holding a copy to the destination of the copy. This incurs a cost of D times the length of a shortest path between the source and the destination. Deleting a copy is only allowed if at least one copy of the file remains. Hence the deletion of a copy needs an understanding between the node whose copy shall be deleted and at least one other node holding a copy. This implies a cost of the distance of the passing copy to its nearest neighbor copy.[2]

A simple reduction of the minimum Steiner tree problem shows that the offline version of the file allocation problem is NP-hard.

1.2 Competitive analysis

In practice the input of the file allocation problem is not given at once but arises step-by-step. Furthermore, the decision which action should be performed has to be made immediately and can not be deferred until the whole input is known. Problems with these characteristics are called *online problems*. An algorithm for

[2] Our model differs here from the model of Bartal [Bar94] where deletions are free.

an online problem which considers the immediate execution of actions in every step based only on prior requests is called *online algorithm*. There are lots of online problems considered in the literature. Most prominently are the paging [ST85], list accessing [ST85, AW98] and k-server [MMS88] problems.

The most common method to measure the performance of online algorithms is the *competitive analysis* [ST85, KMRS88, BEY98]. It consists of a comparison of an online algorithm to an optimal offline algorithm which knows the whole input beforehand. Formally, let ALG be a (randomized) online algorithm and let OPT be an optimal offline algorithm. Furthermore, let $C_{\text{ALG}}(\sigma)$ resp. $C_{\text{OPT}}(\sigma)$ denote the costs which ALG resp. OPT incur while serving input sequence σ. ALG is called *γ-competitive* if an $\alpha \geq 0$ exists such that for all input sequences σ

$$\mathrm{E}\left[C_{\text{ALG}}(\sigma)\right] \leq \gamma \cdot \mathrm{E}\left[C_{\text{OPT}}(\sigma)\right] + \alpha$$

holds. The expectation is thereby taken over the random bits of ALG. If this inequality is even valid for $\alpha = 0$, then ALG is called *strictly γ-competitive*. Since the *competitive ratio* of an online algorithm is a worst-case performance measure, measurements on practical inputs often show a better performance than the theoretical analysis suggests [AL99].

A common view on competitive analysis is to assume a malicious *adversary* ADV which tries to fool an online algorithm ALG. ADV therefore generates both an input sequence σ_{ALG} which is costly for ALG and a solution for σ_{ALG} which causes only low costs. In the case of randomized online algorithms three types of adversaries are commonly used [RS94, BDBK+94, BEY98]. They differ in the knowledge of the actual actions of ALG and in the point of time when they have to construct their solution. An *oblivious* adversary has to generate σ_{ALG} and an optimal offline solution before ALG is started. Both the *adaptive-online* and the *adaptive-offline* adversaries create the input interleaved with the execution of ALG. In every step they choose the next request based on the knowledge of the actions ALG has performed in the past. The adaptive-offline adversary creates an optimal offline solution after the whole input is fixed. The adaptive-online algorithm creates its solution step-by-step like the online algorithm. It has to decide which actions to perform right after it fixes a request, but before it sees ALG's reaction on this request.

The adaptive-offline adversary is more powerful than the adaptive-online adversary which in turn is more powerful than the oblivious adversary [BDBK+94].

1.3 Dynamic networks

There are different reasons why the properties of a network can change. We discuss here the most common reasons for dynamics in different types of networks.

A wired network, which exclusively executes a single application, can change over time because of link respectively node failures or because of intended topology changes. In all these cases the routing paths through the network have to be adapted to the new topology. While these changes normally occur only rarely, their effect on the length of the routing paths can be drastic. Imagine for example a ring network consisting of n nodes. When a single link fails the length of the path between its two adjacent nodes rises from 1 to $n-1$. But since those changes occur only rarely, a manual adaption of the file allocation strategy can be feasible.

The situation is more complicated if the wired network is not used exclusively by one application, e.g. the application is executed over the Internet or a shared LAN/MAN/WAN. Then attributes like the available bandwidth or the latency of a link changes with the amount of data transferred over it. These changes usually do not occur abruptly and are less drastic than a topology change.

In wireless networks mobility of the nodes constitutes another source for dynamics. In a MANET the topology of the network is defined by the transmission ranges of the nodes. Whenever two nodes can communicate with each other this constitute a link of the network. Since the nodes can move arbitrarily the topology of the network changes continuously. Furthermore, failures of nodes occur more often than in wired networks since the nodes mostly depend on finite energy supplies and their environment is generally unsafer. This leads to ever changing routing paths in th network.

Especially in the field of peer-to-peer systems, the usage of overlay networks is wide spread. An overlay network is a logical network which is build on top of another network. The topology of an overlay network is mostly independent of the underlying network and can change over time. Reasons for changes of an overlay network are entering respectively leaving nodes, and measures for perpetuating certain characteristics of the topology, e.g. randomness or expansion [Mah10]. When two nodes of the overlay network communicate with each other, the messages are routed through the overlay network by a routing protocol specific to the overlay network. Thus, the actual routing paths in the physical network depend additionally on the routing paths through the overlay network.

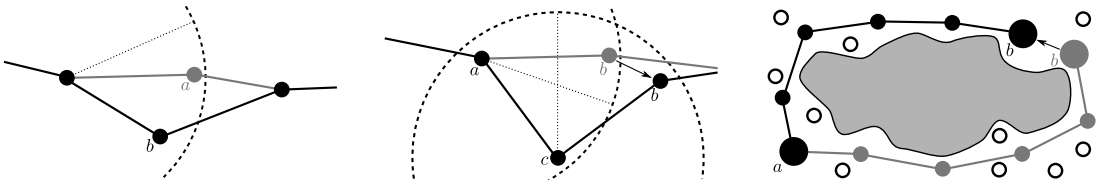
Our model of dynamic networks

We are mostly interested in file allocation in MANETs and use the energy consumption used to transfer data as a distance measure. In this thesis we assume that the file allocation is performed in an overlay network with a fixed and sim-

ple topology in the MANET. This allows us to neglect the continuous topology changes of the MANET and deal only with the changes of the distances. Furthermore, the simplicity of the topology avoids the necessity to deal with the tracking of the positions of the files in the network, which is still an open problem.

The power consumption of a wireless transceiver can be divided into three parts [WS04, WMY06]: the power consumption for sending data, the power consumption for receiving data, and the power consumption for staying in idle mode. The actual amount of energy consumed thereby mainly depends on the characteristics of the transceiver at hand. We assume that all the mobile nodes are equipped with identical transceivers which always use the same amount of energy for transmissions. Thus, the topology of the MANET can be modeled as an unit disk graph and the power consumption for sending respectively receiving a message of unit size is a fixed constant.

The energy needed for the transfer of a data unit between two participating nodes is defined by the used multi-hop path. We assume that this path is determined by the routing protocol of the MANET. Every relay on the path through the MANET has to receive the message from its predecessor and send the message to the next node of the path. The overall power consumption of the transfer is the sum of the power consumptions of the nodes on the routing path. In this way we can determine for any point in time and any pair of participating nodes the energy which would be consumed if a data transfer of unit size would be performed between them. We call this value the *energy-distance* of the two participating nodes.



(a) Node *a* disappears. The path is fixed with node *b*.

(b) Node *b* moves out of the transmission range of node *a*. The path is fixed with node *c*.

(c) The routing protocol switches to a new path between node *a* and node *b* around the obstacle.

Figure 1.2: Examples for topology adaptions. The gray paths are adapted to the black ones.

A change of the energy-distance of a node can have different reasons (cf. Figure 1.2): If a node of a path disappears (cf. Figure 1.2(a)) or moves in such a way that it either gets dispensable or the path gets disconnected (cf. Figure 1.2(b)), a slight topology change is sufficient. In the case of a dispensable node the path can be shortened by removing the node from the path. In the other cases the path can

be fixed by using one or more nearby nodes. More drastic changes of the energy-distance occur if the routing protocol changes big parts of a routing path. This could for example happen if the path circumvents an obstacle (cf. Figure 1.2(c)), if the used routing protocol updates outdated paths, or if a path is disconnected and no local fix is possible. The analysis of the algorithms presented in this thesis can basically deal with all possible jumps in the energy-distance. But the higher the changes of the energy-distances are the higher is the guaranteed competitive ratio of the algorithms.

While our model basically does not restrict the changes of the energy-distances of the participating nodes in a single step, it is not possible to investigate a worst case measure with unrestricted adversaries. An unrestricted adversary could for example choose a participating node on which the online algorithm does not hold a copy, place itself a copy on it, move it in a single step very far away from all the other nodes and then issue a read request from this participating node. Clearly the adversary could cause any online algorithm an unbounded competitive ratio this way. We therefore restrict the changes of the energy-distances an adversaries may generate in a single step.

Definition 1. *Let $\delta \geq 0$. An adversary* Adv *is called δ-restricted iff* Adv *generates only input sequences such that the energy-distances of every edge is always at least δ and changes at most by δ in one step.*

1.4 Related work

Since the placement of data in networks has a major impact on the performance of distributed systems, a lot of research has been made on data management strategies. We will provide here a brief overview of different data management models and especially of the online file allocation problem.

Data management

Early data placement models used a pre-computed static placement of copies of the file, i.e. the positions of the copies are fixed throughout the whole runtime of the system. These models required good estimation of the request characteristics of the nodes and could not deal with fluctuations of these characteristics. An overview of different static file assignment problems can be found in the survey by Dowdy and Foster [DF82].

The next step in data management was the dynamic placement of files. Here copies of the file could be migrated, created and deleted during runtime. The survey of Gravish and Sheng [GS90] gives an overview of different offline models for file placement problems. Common to all these models is that the calculation of

the placement of copies is performed offline. Thus, for these models it is important to have a good estimation of the request characteristics and their fluctuations.

The arising field of online algorithms and competitive analysis [ST85, BEY98] offered a way to get rid of the requirement of prior knowledge about the request characteristics. In 1989 Black and Sleator introduced the replication and the migration problems [BS89]. And in 1992 Bartal, Fiat and Rabani introduced the file allocation and distributed paging problems [BFR95]. These four related problems are all online problems and do not assume any knowledge about the input sequences. A survey about these online problems can be found in [Bar98a]. We have already described the file allocation problem in Subsection 1.1. The replication and the migration problems are basically special cases of the file allocation problem. In the replication problem only read requests are issued and hence there is no reason to delete any copies. The migration problem allows only one copy in the network which can be migrated between nodes. Bartal shows in Theorem 4.2.1 of [Bar94] that the migration problem is basically equivalent to the file allocation problem if only write requests are issued since then multiple copies provide no benefit. The distributed paging problem is a generalization of the file allocation problem where more than one file is present and the nodes have memories of restricted size.

Online file allocation

Bartal, Fiat and Rabani introduced the online file allocation problem in [BFR95]. There they describe the randomized algorithm SteinerBased which works on arbitrary networks. SteinerBased uses internally an arbitrary algorithm for the online Steiner tree problem introduced by Imase and Waxman in [IW91]. Imase and Waxman also introduced the greedy Steiner tree algorithm and showed that it is $O(\log n)$-competitive. Furthermore, they showed that there are networks and inputs such that every online algorithm is at least $\Omega(\log n)$-competitive. Bartal et al. derived a lower bound of $\Omega(\log n)$ for the file allocation problem on arbitrary networks from this lower bound for the online Steiner tree problem. Furthermore, they showed that SteinerBased is $\left(2+\sqrt{3}\right)\cdot c$-competitive against adaptive online adversaries if the used Steiner tree algorithm is strictly c-competitive. In [ABF93] Awerbuch, Bartal and Fiat showed an improved analysis of the greedy Steiner tree algorithm which shows that it is strictly $O(\min(\log n, \log(\mathit{diameter})))$-competitive. This implies that SteinerBased combined with the greedy Steiner tree algorithm is $O(\min(\log n, \log(\mathit{diameter})))$-competitive against adaptive online adversaries. Westbrook and Yan showed in [WY95] that SteinerBased combined with the greedy Steiner tree algorithm is $O(\log\log n)$-competitive in unweighted networks with diameter $O(\log n)$. In [ABF93] Awerbuch, Bartal and Fiat gave an $O(\min(\log n, \log(\mathit{diameter}))$-competitive deterministic algorithm for arbitrary

networks. They also described a distributed version of the algorithm which is $O\left(\log^4 n\right)$-competitive.

In [BFR95] Bartal et al. also investigated the file allocation problem on uniform networks, i.e. complete networks with unit edge weights. An example for such an uniform network is a set of processors with local memory which are interconnected via a shared bus. They introduced the deterministic algorithm Count, which is adapted to a dynamic star network in Subsection 3.1.2.3, and showed that it is 3-competitive as follows: They defined a biunique mapping of the overall costs incurred by Count to the processors. For a processor p they partitioned the input into phases. A phase is the time between two copy deletions on node p. Then they showed that in each phase the costs of p incurred by Count are at most 3-times the costs of an optimal offline algorithm. This proof heavily relies on constant edge weights during a phase and therefore can not be extended to dynamic networks. They also provided a matching lower bound of 3 for randomized algorithms against adaptive online adversaries. This lower bound uses only two adjacent nodes and therefore also holds for trees.

A common approach to handle data management problems on complex or unstructured networks is to approximate the network by a tree embedding [Bar98b, MadHVW97] and perform the data management on this tree. Therefore file allocation algorithms for trees are of special interest. In [BFR95] Bartal et al. gave a memoryless randomized algorithm for tree networks which is 3-competitive against adaptive online adversaries. Lund, Reingold, Westbrook and Yan presented in [LRWY99] a deterministic 3-competitive algorithm based on a work function. They also provided a lower bound of $2+\frac{1}{D}$ against oblivious adversaries. In [MVW99] Krick, Meyer auf der Heide, Räcke, Vöcking and Westermann gave another deterministic 3-competitive algorithm which additionally is distributed. As far as we are aware, closing the gap between the lower $2+\frac{1}{D}$ and upper 3 bound for randomized algorithms against oblivious adversaries is still an open problem.

Data management in dynamic networks

In [ABF98] Awerbuch, Bartal and Fiat introduced an algorithm for file allocation in dynamic networks. Their dynamics are limited to the ability of the nodes to issue requests. Inactive nodes are not able to issue requests, but they can still be used for communication. So, their network is actually static.

Bienkowski, Korzeniowski and Meyer auf der Heide introduced in [BKM04] the dynamic page migration problem. They modeled the dynamic network by a set of n nodes which are placed in a metric space. The nodes can change their positions in the metric over time. The movement is restricted by the assumption that during a fixed time interval the position can change by at most δ.

In [BKM04] Bienkowski et al. gave an $O\Big(\min\big(n\sqrt{D}, D, \lambda\big)\Big)$-competitive randomized algorithm against adaptive online adversaries, where λ is the maximum distance allowed in the metric. There they also provided a matching lower bound of $\Omega\big(n\sqrt{D}, D, \lambda\big)$. In [BDK05] Bienkowski, Dynia and Korzeniowski presented a deterministic $O\big(n\sqrt{D}\big)$-competitive algorithm. They also provided a lower bound of $\Omega\Big(\sqrt{D\log n}, D^{2/3}, \lambda\Big)$ for randomized algorithms and a $O\Big(\sqrt{D}\log n\Big)$-competitive randomized algorithm against oblivious adversaries. Bienkowski and Byrka improved this algorithm in [BJ05] to a $O\Big(\sqrt{D\log n}\Big)$-competitive algorithm against oblivious adversaries.

In [BK05, BKM04] Bienkowski and Korzeniowski investigated the dynamic page migration under Brownian motion. In this scenario the movement of the nodes is modeled by random walks on a ring. They gave a deterministic algorithm which is $O\Big(\min\Big(\sqrt{B}, \sqrt{\frac{D}{B}}, n\Big)\Big)\cdot \operatorname{polylog}(D, B, n)$-competitive with high probability on rings with diameter $B \geq \sqrt[3]{D}$.

In [Bie05a] Bienkowski reduced the power of the adversary by taking away its ability to generate adversarial requests. Instead, the adversary has to choose a probability distribution $\pi : \{1, \ldots, n\} \to [0, 1]$ which is used to generate randomly a request in every step. They gave a deterministic algorithm and show that it is strictly $O(1)$-competitive with probability $1 - O(D^{-\gamma})$ for any constant γ if the input sequence is sufficiently long. Furthermore, they showed that the expected competitive ratio is also constant.

A complete overview of the results about the dynamic page migration problem can be found in [Bie05b, BBKadH09, BM05]. As far as we know these are the only theoretical results for online data management problems in dynamic networks known today.

1.5 Our contribution

We introduce three different models of the file allocation problem in dynamic networks. The first model is a straight forward extension of the file allocation problem to dynamic networks. The second model comprehends stand-by power consumptions of the participating nodes. Furthermore, we introduce an overlay network which is used for the communication between the participating nodes. In the third model the copies of a file are not permanent but are leased for a fixed duration.

Chapter 2 gives a straightforward generalization of the file allocation problem to dynamic networks. The only difference between this model and the classic one described in Section 1.1 is that the edge weights can slightly change in every step.

We show that in this model no competitive online algorithms are possible even for networks consisting of two nodes. Furthermore, we discuss in detail that the main difficulty of an online algorithm for the file allocation problem in dynamic networks is, that it can hardly deduce from an adversarial input sequence where to place copies.

In Chapter 3 we extend the straightforward model by extra costs p, which are incurred by every participating node in every step of the input sequence. These costs can for example be motivated by unavoidable stand-by costs of the transceivers of the participating nodes. We investigate two scenarios of file allocation systems in a MANET. The two scenarios mainly differ in the usage of overlay networks with different topologies.

In the first scenario the participating nodes are connected to a stationary server which holds a primary copy of the file. This scenario implies that the participating nodes and the server form an overlay network with a star topology where the center node always holds a copy. We analyse the performance of the two deterministic algorithms Follow and Count and the randomized algorithm RandomizedFollow and show that all three are $O\left(\frac{D}{p}\delta\right)$-competitive. We give a matching lower bound of $1 + \frac{D}{p}\delta$ for the natural class of "demand-driven" online algorithms which basically create only copies on participating nodes which accessed the file recently. Furthermore, we give a general lower bound of $\Omega\left(\frac{\frac{D}{p}\delta}{\lg\left(\frac{D}{p}\delta\right)}\right)$ for randomized algorithms against oblivious adversaries. These results have previously been published in [MM09].

In the second scenario, we assume that the participating nodes of the MANET form an overlay network with a tree topology. We analyse the randomized online algorithm RandomizedTree and show that it is $\max\left(3, \frac{3D+3}{p}\left(1 - \frac{1}{n}\right)\delta\right)$-competitive against adaptive-online adversaries. This is again optimal for demand-driven algorithms since the lower bound for star networks holds also for trees.

Chapter 4 presents the description and results of simulations of the two algorithms Follow and Count for the star topology. We therefor simulated a MANET using the sensor network simulator Shawn [KPB+05, FKFP07]. The file is primarily stored on a stationary server and some of the mobile nodes access it. Because the communication range of the mobile nodes is limited, they have to communicate via multi-hop paths with the server. We used the "mobility model based on social network theory" from [MM07] to generate the movement of the nodes. The participating nodes generate file accesses stochastically and independent of each other. The simulations show that the competitive ratio for realistic inputs is much lower than the worst-case analysis suggests. Especially, the performance of Count proved to be reasonable good for practical application.

In Chapter 5 we introduce another model for file allocation in dynamic net-

works. We are again looking at a star topology where the center node always holds a copy. But here we do not assume step costs of the participating nodes. Instead, we assume that a copy placed on a participating node is only leased for a fixed number of L steps. When this time expires, the participating node can either drop the copy or renew the lease for another L steps. Dropping the copy is free, but a renewal costs the current distance of the participating node to the center. The concept of leases is especially able to reduce the duration of inconsistencies which can occur when a participating node holding a copy is temporarily not connected. We give a lower bound of $\Omega(L)$ for the competitive ratio of randomized algorithms against oblivious adversaries. Furthermore, we analyse two simple deterministic algorithms which can be combined to a $O(LD)$-competitive algorithm.

CHAPTER 2

Limitations of Extension to Dynamic Networks

In this chapter, we investigate a straightforward extension of the file allocation problem to dynamic networks. We consider a set of participating nodes in a MANET which is interconnected via an overlay network with an arbitrary but fixed topology. Thus, our model is basically the same as the classical model for static networks. The only difference is that the weights of the edges may change over time. We show in Section 2.2 that there can be no competitive online algorithms against an adversary which controls both the edge weights and the request sequence. This means that a straightforward generalization of the online file allocation problem to dynamic networks is not suitable. This stands in contrast to the online page migration problem where the (nearly) straightforward generalization of Bienkowski [Bie05b] was very fruitful.

2.1 Our model

The *file allocation problem in a dynamic network* is defined as follows: We are looking at a connected network $G = (\{1, \ldots, n\}, E)$ of n nodes. All the nodes of the network can access an indivisible file which initially is stored on an arbitrary node $s \in \{1, \ldots, n\}$. During the runtime of the system, an algorithm ALG can arbitrarily place copies of the file on the nodes. The *replica set* $R^t_{\text{ALG}} \subseteq \{1, \ldots, n\}$ is defined as the set of nodes where algorithm ALG has a copy at the end of step t.

The input consists of the initial edge weights d_0 and a finite sequence σ of pairs (d_t, a_t), $1 \leq t \leq |\sigma|$, of a weight-function and an access request. It is presented to the online algorithm in equidistant discrete time steps. The function $d_t : E \to \mathbb{R}^n_{>0}$ defines the weights of the edges in step t. The access request $a_t \in \{-, r_i, w_i \mid i \in \{1, \ldots, n\}\}$ either indicates that no access occurs $(-)$ or a read (r) respectively

a write (w) access is issued at node i. Every step t is divided into the following three phases, which are executed strictly in the given order: First the distances of the edges are updated to d_t, then request a_t is served, and finally, the algorithm may replicate and/or delete copies.

We consider the shared file as an indivisible container of a fixed number of data units of uniform size. A single read respectively write access always affects just one unit of the file. While a read request can be fulfilled by a single copy, a write request has to update all the copies. Every transfer of such a data unit costs the length of the used path. The costs $C_{\text{ALG}}(\sigma_t)$ incurred by ALG in step t are composed of two components: the transfer costs for fulfilling the data access and the transfer costs for creating and deleting copies. The costs $C_{\text{ALG}}(a_t)$ for serving a data access from $i \in \{1, \ldots, n\}$ are naturally defined as

$$C^t_{\text{ALG}}(-) := 0 \qquad C^t_{\text{ALG}}(r_i) := d_t\Big(i,\ R^{t-1}_{\text{ALG}}\Big) \qquad C^t_{\text{ALG}}(w_i) := st_t\Big(R^{t-1}_{\text{ALG}} \cup \{i\}\Big)$$

A replication of the file from node i to node j costs $D \cdot d_t(i,\ j)$, where $D \geq 1$ is a fixed constant depending on the size of the file. The deletion of a copy can either be free as in the classical model of Bartal [BFR92] or costs the distance to the next copy as in the models in the following chapters. The lower bound in the following section works for both variants. We denote by $C_{\text{ALG}}(\sigma) := \sum_{t=1}^{|\sigma|} C_{\text{ALG}}(\sigma_t)$ the overall costs incurred by algorithm ALG while serving input σ.

2.2 Lower bound

Here, we will give a simple proof that no deterministic algorithm can be competitive in this model. The main idea of this proof can also be extended to randomized algorithms against oblivious adversaries as in [BBKadH09]. The impossibility of competitive randomized algorithms against oblivious adversaries in this model is also implicitly shown by Theorem 3.9 ($p = 0$) and Theorem 5.3 ($L = \infty$).

Theorem 2.1. *For every connected network $G = (\{1, \ldots, n\}, E)$ with $n \geq 2$, no deterministic online algorithm for the file allocation problem in a dynamic network can be competitive.*

Proof. Let DET be an arbitrary deterministic online algorithm for the file allocation problem in a dynamic network and let $i, j \in \{1, \ldots, n\}$ be two arbitrary nodes in G. We describe a δ-restricted adversary ADV generating an input sequence which consists of arbitrary many phases τ. We show for every phase that $C_{\text{DET}}(\tau) \geq \frac{D}{D+1}\gamma \cdot C_{\text{ADV}}(\tau)$ holds for any $\gamma \in \mathbb{N}_{>0}$.

We assume that the distance of i to j is initially δ and that the start copy is on i. If this is not the case the adversary can establish this situation by moving i and j to distance δ, changing R_{Adv} to $\{i\}$ and forcing Det to do the same as described below. This would incur Adv at most a constant cost.

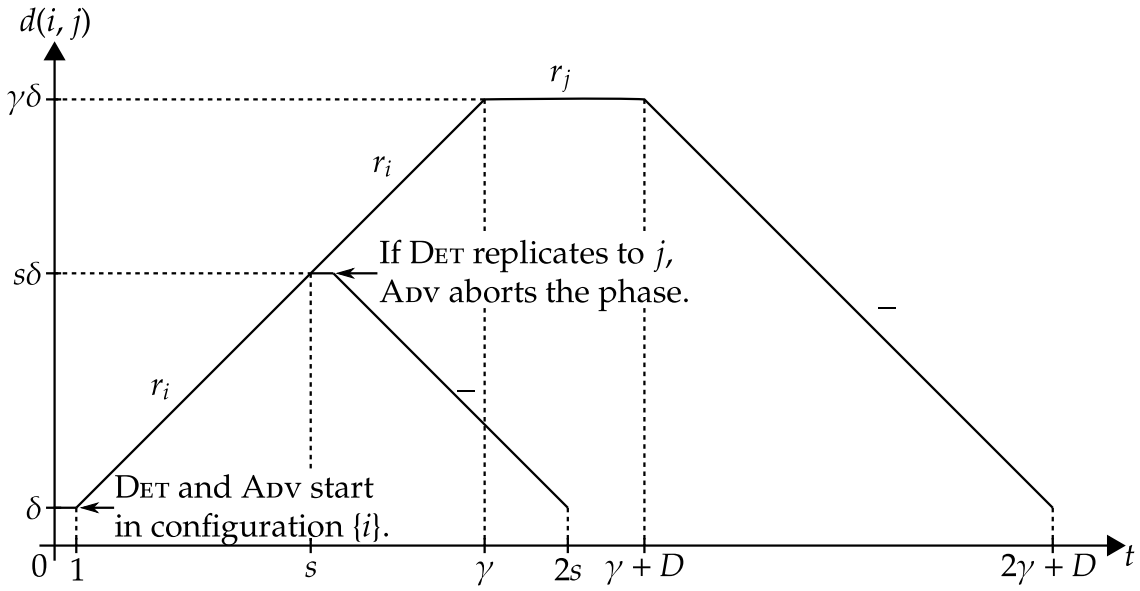


Figure 2.1: A single phase of the lower bound.

Every phase τ (see Figure 2.1) starts with a distance of δ between i and j. Furthermore, both Det and Adv solely have a copy on node i. A phase consists of an expansion-stage, an optional request-stage and a contraction-stage. In the expansion-stage the adversary moves i and j apart with speed δ, i.e. $d(i, j) = t \cdot \delta$ in step t of the expansion-stage, and issues read requests from i. In the request-stage the nodes do not move and j issues read requests for D steps. In the contraction-stage i and j move back to their initial distance of δ and no requests are issued.

If Det creates a copy on j in step $s \leq \gamma$ of the expansion-stage, Adv immediately changes to the contraction-stage. This means that Det incurs at least a cost of $D \cdot s\delta$ for creating the copy on j. Adv does nothing during the phase. And since there were only read requests from i and Adv permanently had a copy on i, Adv does not incur any costs at all. Thus,

$$C_{\text{Det}}(\tau) \geq D \cdot s\delta > 0 = \gamma \cdot C_{\text{Adv}}(\tau)$$

Otherwise, i and j reached distance $\gamma\delta$ and Det did not place a copy on j. Then Adv starts the request-stage, i.e. issues read request from j at distance $\gamma\delta$. Since Det has no copy on j at the beginning of the request-stage, it either has to fulfill these requests by transferring data for every request or to create a copy on j. In

both cases the costs of DET are at least $D \cdot \gamma\delta$. ADV on the other hand knows about the upcoming request-stage beforehand and therefore creates a copy on j in the first step of the expansion-stage which costs $D\delta$. It deletes this copy right after the contraction stage for δ. Thus depending on the chosen cost model, the costs of ADV are $D\delta$ if deletions are free and $(D+1)\delta$ otherwise. This results in

$$C_{\text{DET}}(\tau) \geq D \cdot \gamma\delta \geq \frac{D}{D+1}\gamma \cdot C_{\text{ADV}}(\tau)$$

Thus, for every phase $C_{\text{DET}}(\tau) \geq \frac{D}{D+1}\gamma \cdot C_{\text{ADV}}(\tau)$ holds. Between two consecutive phases ADV forces DET to store an exclusive copy on node i. ADV does this by consecutively issuing write requests from i. As long as DET has copies on any other node than i, it has to pay for these copies. ADV on the other hand avoids any costs by deleting a possibly existing copy on j. Thus, DET has to delete eventually all copies on other nodes than i or will incur an arbitrary high cost, what implies that it can not be competitive. □

2.3 Conclusion

The given lower bound shows that the file allocation problem is more difficult to handle online than the page migration problem. The reason for this is that the adversary can hide its own configuration very well. When an adversary issues a long series of requests from a fixed node, the online algorithm knows that the adversary has a copy of the file near that node. Since in the page migration problem there is always only one copy, the online algorithm has a good approximation of the configuration of the adversary. In the file allocation problem on the other hand, the adversary could have additional copies. But there is no clue for the online algorithm how many additional copies exist and where they are placed. The lack of information about the configuration of the adversary makes the file allocation problem very difficult for online algorithms. A similar lack of information would occur in the page migration problem if the model would not force the adversary to issue a request in every step.

In the following chapters we give and analyse two variants of the file allocation problem in dynamic networks which lessen the described difficulty. The version in Chapter 3 introduces unavoidable step costs and the version in Chapter 5 adds an expiry date to copies. Both variants prevent that the adversary moves hidden copies far away from the center of requests for free. This makes competitive online algorithms possible.

CHAPTER 3

File Allocation with Step Costs

In this chapter we introduce step costs for the participating nodes to the file allocation problem in dynamic networks. We assume that the transceiver of every participating node consumes a constant stand-by power p in every step. This stand-by power consumptions imply unavoidable costs for the adversary in every step. While this does not prevent an adversary to cheaply create copies and hide them from the online algorithm as seen in the last chapter, it means that an adversary can not move such hidden copies far away from the current center of requests for free anymore. As we will see, these stand-by costs allow online algorithms to be competitive.

The definition of energy-distances implies that, even given a fixed maximum speed of the mobile nodes, the energy-distance of a participating node can change drastically during a single step (cf. Figure 1.2(c)). Therefore, we try to relate the costs of our online algorithms to the costs of an optimal offline algorithm OPT in a way that takes the changes of energy-distances into account. Here, we measure the quality of an online algorithm by the following refinement of the competitive ratio:

Definition 2. *Let* ALG *be any online algorithm and* OPT *be an optimal offline algorithm.* ALG *has* amortized competitive ratio $\gamma : \mathbb{R} \to \mathbb{R}_{\geq 1}$ *if an* $\alpha \geq 0$ *exists such that for any input sequence* σ

$$\sum_{t=1}^{|\sigma|} C_{\text{ALG}}(\sigma_t) \leq \sum_{t=1}^{|\sigma|} \gamma(\bar{\delta}_t) \cdot C_{\text{OPT}}(\sigma_t) + \alpha$$

holds, where $\bar{\delta}_t$ *is the average change of the edge weights of the network in step* t*.*

We will examine two different overlay networks with fixed topologies. The first one uses a star topology where the center node holds a primary copy and the

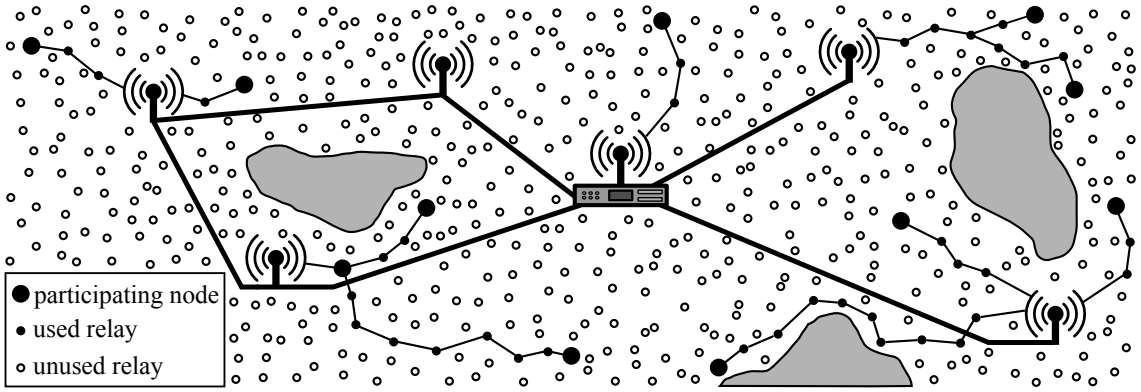


Figure 3.1: A stationary server provides a shared file to the members of a MANET. Every participating node is connected to one access point via a path of relays.

second one is a tree topology. Both topologies have in common that the Steiner tree for any subset of nodes is unique and easy to find.

The star topology arises from the following scenario: A stationary server is connected via one or more stationary access points to a mobile ad hoc network (see Figure 3.1). On this server a primary copy of the file is stored at all times. The participating nodes need secured access to the file. The wireless communication between the server and the participating nodes therefore is cryptographically authenticated and secured. This induces that all the communication has to involve the server, i.e. there are no direct data exchanges between two participating nodes. A file allocation algorithm can create copies of the file on the participating nodes which can then be used to fulfill read requests from this participating node.

We describe and analyse the deterministic algorithms COUNT and FOLLOW and the randomized algorithm RANDOMIZEDFOLLOW. They are all $O\left(\frac{D}{p}\delta\right)$-competitive against δ-restricted adversaries. Common to all three algorithms is that they create copies only on a participating node right after it issued a request. Intuitively, this demand-driven behavior seems reasonable. In Subsection 3.1.2 we give a more general definition of demand-driven algorithms which probably includes all practically useful online algorithms. We give a very strong lower bound for those demand-driven algorithms of $1 + \frac{D}{p}\delta$.

In Subsection 3.1.3 we show a lower bound of $1 + \frac{\frac{D}{p}\delta}{8\left\lceil\left(1+\frac{D}{p}\delta\right)\right\rceil}$ for general online algorithms. This lower bound holds especially for algorithms which are not demand-driven.

The tree topology investigated in Section 3.2 is a first attempt to perform file allocation in a serverless MANET. Figure 3.2 gives an example of such a tree in a MANET. The participating nodes thereby form an overlay network with a tree topology. Two participating nodes can only communicate with each other

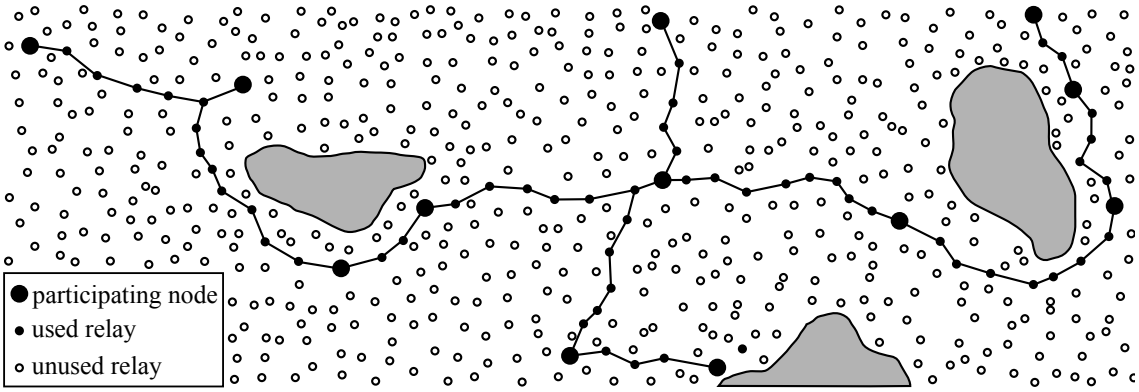


Figure 3.2: A file is shared on a tree which is embedded into a MANET. Every participating node is connected to its tree neighbors via a path of relays.

via a path in the overlay network. The simplicity of the tree topology avoids the necessity of a data tracking service which is able to find approximations of nearest copies for read requests and minimum Steiner trees for updates.

In Subsection 3.2.2 we describe and analyse the memoryless randomized algorithm RandomizedTree on a dynamic tree and show that it is also $O\left(\frac{D}{p}\delta\right)$-competitive.

3.1 File allocation in a dynamic star network

In this section we introduce and analyse our model for the file allocation problem with stand-by costs in a dynamic star network.

3.1.1 Our model

In order to determine the costs of a data management strategy in step t for the scenario described above, we only have to know the following information about the configuration: the current replica set, i.e. the set of participating nodes holding a copy of the file, and the vector of current distances $d_t = [d_t^1, \ldots, d_t^n]$, where d_t^i denotes the energy-distance from participating node i to the server in step t. Such a configuration can be described via a star network: the center of the star represents the server, the n peripheral nodes represent the participating nodes and the distances between the peripheral nodes and the center represent the current energy-distances of the participating nodes to the servers.

The *file allocation problem in a dynamic star network (*Fads*)* is defined as follows: We are given a star network consisting of a center node c (the server) and n peripheral nodes $1, \ldots, n$. All the nodes of the star can access an indivisible file

which initially is stored only on c. During the runtime of the system, an algorithm ALG can arbitrarily place copies of the file on the peripheral nodes. The center node always holds a copy. The *replica set* $R^t_{\text{ALG}} \subseteq \{c, 1, \ldots, n\}$ is defined as the set of nodes where algorithm ALG has a copy at the end of step t.

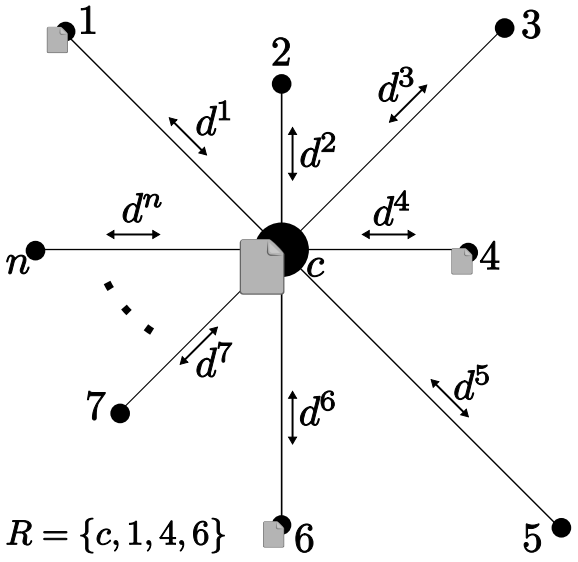


Figure 3.3: The file allocation system is modeled as a star with dynamic edge weights.

The input consists of the initial distances d_0 and a finite sequence σ of pairs (d_t, a_t), $1 \leq t \leq |\sigma|$, of a vector of distances and an access request. It is presented to the online algorithm in equidistant discrete time steps. The vector $d_t = [d_t^1, \ldots, d_t^n] \in \mathbb{R}^n_{>0}$ defines the distances of the peripheral nodes to the center in step t. The access request $a_t \in \{-, r_i, w_i \mid i \in \{c, 1, \ldots, n\}\}$ either indicates that no access occurs $(-)$ or a read (r) respectively a write (w) access is issued at node i. Every step t is divided into the following three phases, which are executed strictly in the given order: First the distances between the peripheral nodes and c are updated to d_t, then request a_t is served, and finally, the algorithm may replicate and/or delete copies.

We consider the shared file as an indivisible container of a fixed number of data units of uniform size. A single read respectively write access always affects just one unit of the file. While a read request can be fulfilled by a single copy, a write request has to update all the copies. Every transfer of such a data unit costs the corresponding distance.

The costs $C_{\text{ALG}}(\sigma_t)$ incurred by algorithm ALG in step t are composed of three components: a fixed stand-by cost of p for every peripheral node, the transfer costs for fulfilling the data access and the transfer costs for creating and deleting copies. We will occasionally use the following split-up of the costs $C_{\text{ALG}}(\sigma_t) = np + \tilde{C}_{\text{ALG}}(\sigma_t)$,

where $\tilde{C}_{\text{Alg}}(\sigma_t)$ denotes the costs incurred by Alg for serving the request and performing actions. The costs $\tilde{C}_{\text{Alg}}(a_t)$ incurred by Alg for serving a data access from $i \in \{c, 1, \ldots, n\}$ are naturally defined by

$$\tilde{C}^t_{\text{Alg}}(-) := 0, \qquad \tilde{C}^t_{\text{Alg}}(r_i) := \begin{cases} 0, & \text{if } i \in R^{t-1}_{\text{Alg}} \\ d^i_t, & \text{otherwise} \end{cases}, \qquad \tilde{C}^t_{\text{Alg}}(w_i) := \sum_{j \in \left(R^{t-1}_{\text{Alg}} \cup \{i\}\right) \setminus \{c\}} d^j_t$$

Notice that $\tilde{C}_{\text{Alg}}(r_c)$ is always 0 and therefore a read access from the center node is equivalent to no access $(-)$. Furthermore, $\tilde{C}_{\text{Alg}}(w_i)$ is always at least d^i_t since the center always holds a copy. A replication of the file from the center to peripheral node i costs $D \cdot d^i_t$, where $D \geq 1$ is a fixed constant depending on the size of the file. If an algorithm decides to delete the copy of peripheral node i, both the center and i have to know this. Thus, a communication between these two nodes has to take place which costs d^i_t.[1] We denote by $C_{\text{Alg}}(\sigma)$ the overall costs incurred by algorithm Alg while serving input sequence σ.

3.1.2 Demand-driven algorithms

In this subsection we investigate the natural class of demand-driven algorithms.

Definition 3. *Let $r \geq 1$ and $\Delta \geq 0$. A file allocation algorithm is called* (r, Δ)-demand-driven *if it replicates only to peripheral nodes which have a distance of at most Δ to any node which issued a request during the previous r steps.*

As far as we know, all the investigated algorithms for the online file allocation problem in static networks are demand-driven. Furthermore, our intuition tells us that placing copies on a peripheral node which is far away from the current center of requests is not a good idea, because this means that an algorithm just speculates where future requests could occur.

3.1.2.1 Lower bound

The following lower bound for demand-driven algorithms works not only for deterministic, but also for randomized algorithms. It consists of an input sequence which is completely oblivious of the concrete online algorithm. In the following subsections, we give two up to a constant factor matching $O\left(\frac{D}{p}\delta\right)$-competitive $(1, 0)$-demand-driven algorithms.

Theorem 3.1. *Let $r \geq 1$ and $\Delta \geq 0$. The competitive ratio of any (r, Δ)-demand-driven algorithm Alg for Fads against δ-restricted oblivious adversaries is at least $1 + \frac{D+1}{p}\delta$.*

[1] This differs from the file allocation model of Bartal et al. [BFR92] where deletions are free.

Proof. In the following, we define a class of input sequences σ_T for ALG where the parameter $T \geq 1$ basically determines the length of σ_T. We show that for any constant α and $\epsilon > 0$ we can choose a T such that

$$C_{\text{ALG}}(\sigma_T) \geq \left(1 + \frac{D+1}{p}\delta - \epsilon\right) \cdot C_{\text{OPT}}(\sigma_T) + \alpha. \tag{3.1}$$

During the whole input sequence all the peripheral nodes perform the same movement, i.e. $d_t^i = d_t^j$ for all $1 \leq i, j \leq n$ and $t \geq 0$. The input sequence σ_T (see Figure 3.4) starts with all peripheral nodes at distance $\Delta + \delta$ and with no copies on the peripheral nodes. σ_T consists of two stages: In the expansion-stage the peripheral nodes move away from c with speed δ for T steps. During this stage only c issues read requests. In the reading-stage each peripheral node consecutively issues $D + 1$ read requests. Note that σ_T is beside Δ completely independent from ALG's behavior.

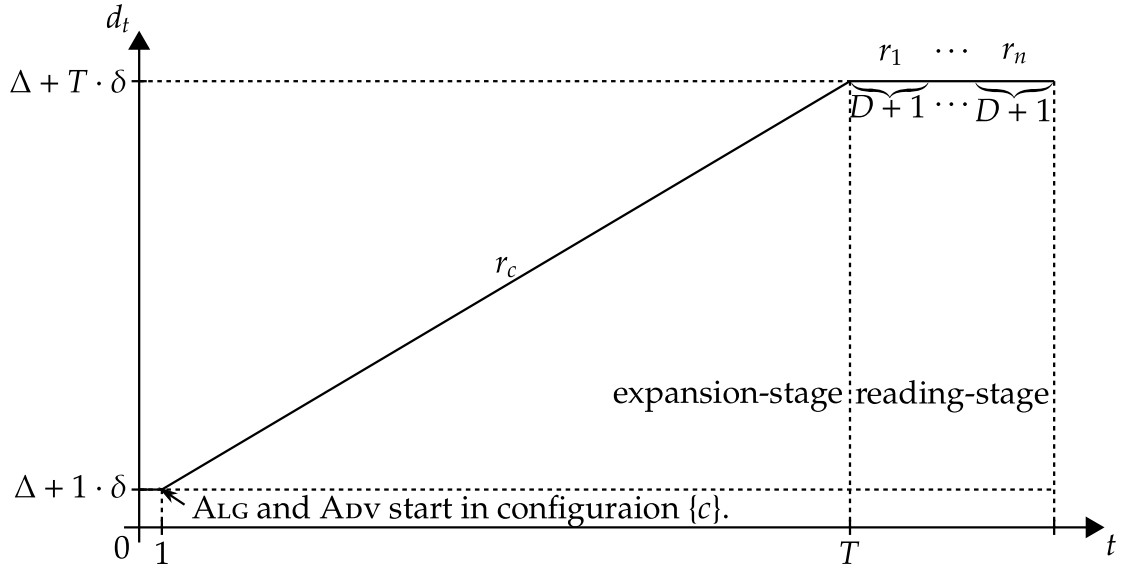


Figure 3.4: A single phase σ_T of the lower bound for demand-driven algorithms.

Obviously, an optimal offline algorithm OPT creates a copy on every peripheral node at the beginning of the expansion-stage, because it knows that these nodes will read the file at distance $\Delta + T\delta$. So the costs of OPT are bounded by

$$C_{\text{OPT}}(\sigma_T) \leq \underbrace{nD(\Delta + 1 \cdot \delta)}_{\text{replication}} + \underbrace{Tnp}_{\text{expansion-stage}} + \underbrace{n(D+1)np}_{\text{reading-stage}}.$$

Since the distances of the peripheral nodes to the requesting node are greater than Δ during the expansion-stage, a (r, Δ)-demand-driven algorithm will not create copies on any peripheral nodes before the reading-stage. So ALG has to pay in the reading-stage $\Delta + T \cdot \delta$ for the first read request from a peripheral node and

then either can create a copy on this peripheral node or serve all $D+1$ requests by sending the requested data from c. In either case it incurs a cost of at least $(D+1)(\Delta + T \cdot \delta)$. Thus,

$$C_{\text{Alg}}(\sigma_T) \geq \underbrace{(T + n(D+1))}_{\text{\# of steps}} np + \underbrace{n(D+1)(\Delta + T \cdot \delta)}_{\text{reading}}$$
$$> (np + n(D+1)\delta) \cdot T.$$

Altogether we therefore have for any constant α

$$\frac{C_{\text{Alg}}(\sigma_T) - \alpha}{C_{\text{Opt}}(\sigma_T)} > \frac{(np + n(D+1)\delta) \cdot T - \alpha}{np \cdot T + n^2(D+1)p + nD(\Delta + 1 \cdot \delta)}$$
$$\underset{T\to\infty}{\to} 1 + \frac{D+1}{p}\delta.$$

□

The given input sequence can also be repeated several times like in the proof of Theorem 2.1. Then the lower bound drops to $1 + \frac{D+1}{2p}\delta$ since the adversary has to move the peripheral nodes back to the initial position in every phase.

3.1.2.2 Algorithm Follow

Algorithm Follow[2] (see Algorithm 1) is based on the "dynamic caching strategy" of [MMVW97]. It consists of the following two independent rules:

(i) If a request is issued by a peripheral node which does not hold a copy, a new copy is placed on this peripheral node.

(ii) If a node issues a write request, all the copies on peripheral nodes other than the requesting one are deleted.

This means that the copies basically follow the issued requests.

Theorem 3.2. *Let σ be an arbitrary input sequence and* Opt *be an arbitrary optimal offline algorithm for* Fads. *Then*

$$C_{\text{F}}(\sigma) \leq \sum_{t=0}^{|\sigma|} \gamma_{\text{F}}(\bar{\delta}_t) \cdot C_{\text{Opt}}(\sigma_t)$$

holds, where $\bar{\delta}_t$ is the average change of the distances of the peripheral nodes in step t, i.e. $\bar{\delta}_t := \frac{1}{n}\sum_{i=1}^{n} \left| d_t^i - d_{t-1}^i \right|$ and

$$\gamma_{\text{F}}(\bar{\delta}_t) := \max\left(D+3,\ 1 + \frac{D+1}{p}\bar{\delta}_t\right).$$

[2] We abbreviate Follow with F in indices.

Algorithm 1 FOLLOW

```
1: On request (d_t, a_t) do
2:     Let i be the source of the current request a_t.
3:     if i ∉ R_F then
4:         Replicate to i.
5:     end if
6:     if a_t = w_i then
7:         for j ∈ R_F \ {c, i} do
8:             Delete copy on j.
9:         end for
10:    end if
11: end
```

The function $\gamma_{\mathrm{F}}(\bar{\delta}_t)$ is depicted in Figure 3.5. Note that the theorem does not induce that $C_{\mathrm{F}}(\sigma_t)$ has to be smaller than or equal to $\gamma_{\mathrm{F}}(\bar{\delta}_t) \cdot C_{\mathrm{OPT}}(\sigma_t)$ in every step t. $\gamma_{\mathrm{F}}(\bar{\delta}_t)$ rather constitutes the amortized competitive ratio of step t.

The following corollary is an immediate consequence of Theorem 3.2 since by definition a δ-restricted adversary can only produce input sequences such that $\bar{\delta}_t = \frac{1}{n}\sum_{i=1}^{n}\left|d_t^i - d_{t-1}^i\right| \leq \frac{1}{n}n\delta = \delta$ holds in every step t .

Corollary 3.3. FOLLOW *is strictly* $\max\left(D+3, 1+\frac{D+1}{p}\delta\right)$*-competitive against* δ*-restricted adversaries.*

Proof of Theorem 3.2. We will prove the theorem with the help of the potential function $\Phi_t := \sum_{i=1}^{n} \Phi_t^i \cdot d_t^i$, where

$$\Phi_t^i := \begin{cases} D+1 & \text{if } i \in R_{\mathrm{OPT}} \setminus R_{\mathrm{F}}, \\ 2 & \text{if } i \in R_{\mathrm{F}} \setminus R_{\mathrm{OPT}}, \\ 0 & \text{else.} \end{cases}$$

Notice that $\Phi_t \geq 0$ for $0 \leq t \leq |\sigma|$ and especially $\Phi_0 = 0$ in the natural case that FOLLOW and OPT start with the same replica set.

We divide every step into two parts. In the first part both FOLLOW and OPT pay p for every peripheral node and the peripheral nodes move to their new positions. In the second part the file access is served and the algorithms may reallocate copies. We denote by $\Phi_{t-\frac{1}{2}}$ the potential after the first part of step t, i.e. $\Phi_{t-\frac{1}{2}} = \sum_{i=1}^{n} \Phi_{t-1}^i \cdot d_t^i$.

Below we show that for the first part of any step t holds:

$$np + \Phi_{t-\frac{1}{2}} - \Phi_{t-1} \leq \left(1 + \frac{D+1}{p}\bar{\delta}_t\right) np. \tag{3.2}$$

And for the second part, we show:

$$\tilde{C}_{\mathrm{F}}(\sigma_t) + \Phi_t - \Phi_{t-\frac{1}{2}} \leq (D+3)\tilde{C}_{\mathrm{Opt}}(\sigma_t). \tag{3.3}$$

Notice that (3.3) especially says that Follow is $(D+3)$-competitive in a static star network.

Together this results in

$$\begin{aligned}
C_{\mathrm{F}}(\sigma) &\leq C_{\mathrm{F}}(\sigma) + \Phi_{|\sigma|} - \Phi_0 \\
&= \sum_{t=1}^{|\sigma|} \left(np + \tilde{C}_{\mathrm{F}}(\sigma_t)\right) + \Phi_{|\sigma|} - \Phi_0 \\
&= \sum_{t=1}^{|\sigma|} \left(\left(np + \Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right) + \left(\tilde{C}_{\mathrm{F}}(\sigma_t) + \Phi_t - \Phi_{t-\frac{1}{2}}\right)\right) \\
&\overset{(*)}{\leq} \sum_{t=1}^{|\sigma|} \left(\left(1 + \frac{D+1}{p}\bar{\delta}_t\right) np + (D+3)\tilde{C}_{\mathrm{Opt}}(\sigma_t)\right) \\
&\leq \sum_{t=1}^{|\sigma|} \max\left(D+3,\ 1 + \frac{D+1}{p}\bar{\delta}_t\right) C_{\mathrm{Opt}}(\sigma_t),
\end{aligned}$$

where $(*)$ follows from (3.2) and (3.3).

For the proof of (3.2), we observe that the costs of Follow and Opt in the first phase of a step are always np. Furthermore, the value of Φ changes only due to the changes of the distances since no actions are performed. Thus, we have:

$$\begin{aligned}
np + \Phi_{t-\frac{1}{2}} - \Phi_{t-1} &= np + \sum_{i=1}^{n} \Phi_{t-1}^i \cdot (d_t^i - d_{t-1}^i) \\
&\leq np + \sum_{i=1}^{n} (D+1) \cdot \left|d_t^i - d_{t-1}^i\right| \\
&= \left(1 + \frac{D+1}{np} \sum_{i=1}^{n} \left|d_t^i - d_{t-1}^i\right|\right) np \\
&= \left(1 + \frac{D+1}{p}\bar{\delta}_t\right) np
\end{aligned}$$

For the analysis of FOLLOW in the second part of a step, we use the following general cost allocation scheme: we allot the transfer costs incurred by any algorithm ALG as follows to the peripheral nodes. Peripheral node i is charged for all those costs that are caused by it. More precisely, peripheral node i pays for all costs incurred for creating and deleting copies on i. Furthermore, i is charged d_t^i in step t if i issues a read request and does not hold a copy, or if i holds a copy and a write request is issued by any node. We denote by $C^i_{\text{ALG}}(\sigma)$ the costs incurred by i while ALG serves the input sequence σ. Since all the costs incurred by ALG are mapped to exactly one peripheral node, $C_{\text{ALG}}(\sigma) = \sum_{i=1}^{n} C^i_{\text{ALG}}(\sigma)$ holds.

We proof (3.3) by showing that for all $1 \le i \le n$ holds

$$\tilde{C}^i_{\text{F}}(\sigma_t) + \left(\Phi^i_t - \Phi^i_{t-\frac{1}{2}}\right) d^i_t \le (D+3) \cdot \tilde{C}^i_{\text{OPT}}(\sigma_t). \tag{3.4}$$

This implies

$$\begin{aligned}
\tilde{C}_{\text{F}}(\sigma_t) + \Phi_t - \Phi_{t-\frac{1}{2}} &= \sum_{i=1}^{n} \left(\tilde{C}^i_{\text{F}}(\sigma_t) + \left(\Phi^i_t - \Phi^i_{t-\frac{1}{2}}\right) d^i_t\right) \\
&\le \sum_{i=1}^{n} (D+3) \cdot \tilde{C}^i_{\text{OPT}}(\sigma_t) \\
&= (D+3) \cdot C_{\text{OPT}}(\sigma_t).
\end{aligned}$$

We show (3.4) in two steps: First we analyze the costs and the change of Φ^i due to the request and the actions of FOLLOW. Thereafter we have a look at the change of Φ^i caused by an action of OPT.

The proof of (3.4) for the first step can be found in Table 3.1. In the second step, we have always $\tilde{C}^i_{\text{F}} = 0$. So, it is enough to look at the change of the potential Φ^i.

If OPT creates a new copy on peripheral node i, Φ^i only increases if FOLLOW does not hold a copy on i. Thus, $\Delta\Phi^i \le (D+1)$. So we have

$$\tilde{C}^i_{\text{F}} + \Delta\Phi^i d^i_t \le (D+1) d^i_t < (D+3) D d^i_t = (D+3) \cdot \tilde{C}^i_{\text{OPT}}.$$

If OPT deletes a copy on peripheral node i, Φ^i only increases if FOLLOW does hold a copy on i. Thus, $\Delta\Phi^i \le 2$. So we have

$$\tilde{C}^i_{\text{F}} + \Delta\Phi^i d^i_t \le 2 d^i_t < (D+3) d^i_t = (D+3) \cdot \tilde{C}^i_{\text{OPT}}. \qquad \square$$

3.1.2.3 Algorithm COUNT

Algorithm COUNT[3] (see Algorithm 2) is an adaption of the algorithm of the same name from [BFR92]. There it is shown that COUNT is 3-competitive on a static uniform network, i.e. a complete network with uniform distances.

[3] We abbreviate COUNT with C in indices.

Table 3.1: Changes of replica set R_F and corresponding values of Φ^i, $\tilde{C}^i_F$ and $\tilde{C}^i_{OPT}$ regarding peripheral node i in the second part of a step on request a_t.

$i \in R_F$	$i \in R'_F$	$i \in R_{OPT}$	$\tilde{C}^i_F(\sigma_t)$	$\Delta\Phi^i d^i_t$	$\tilde{C}^i_F(\sigma_t) + \Delta\Phi^i d^i_t$	$(D+3) \cdot \tilde{C}^i_{OPT}(\sigma_t)$
$a_t = r_j$ with $j \neq i$						
no	no	no	0	$(0-0)d^i_t$	0	0
no	no	yes	0	$((D+1)-(D+1))d^i_t$	0	0
yes	yes	no	0	$(2-2)d^i_t$	0	0
yes	yes	yes	0	$(0-0)d^i_t$	0	0
$a_t = r_i$						
no	yes	no	$(D+1)d^i_t$	$(2-0)d^i_t$	$(D+3)d^i_t$	$(D+3)d^i_t$
no	yes	yes	$(D+1)d^i_t$	$(0-(D+1))d^i_t$	0	0
yes	yes	no	0	$(2-2)d^i_t$	0	$(D+3)d^i_t$
yes	yes	yes	0	$(0-0)d^i_t$	0	0
$a_t = w_j$ with $j \neq i$						
no	no	no	0	$(0-0)d^i_t$	0	0
no	no	yes	0	$((D+1)-(D+1))d^i_t$	0	$(D+3)d^i_t$
yes	no	no	$2d^i_t$	$(0-2)d^i_t$	0	0
yes	no	yes	$2d^i_t$	$((D+1)-0)d^i_t$	$(D+3)d^i_t$	$(D+3)d^i_t$
$a_t = w_i$						
no	yes	no	$(D+1)d^i_t$	$(2-0)d^i_t$	$(D+3)d^i_t$	$(D+3)d^i_t$
no	yes	yes	$(D+1)d^i_t$	$(0-(D+1))d^i_t$	0	$(D+3)d^i_t$
yes	yes	no	d^i_t	$(2-2)d^i_t$	d^i_t	$(D+3)d^i_t$
yes	yes	yes	d^i_t	$(0-0)d^i_t$	d^i_t	$(D+3)d^i_t$

Count has a counter c^i for every peripheral node $i \in \{1, \ldots, n\}$, which may hold integer values ranging from 0 to $D+1$. On any request (read or write) from a peripheral node, Count increases the value of the corresponding counter by 1 (but not exceeding $D+1$). If c^i reaches $D+1$ and i does not already hold a copy, a new copy is created on i. On write requests from any node (including the center), Count additionally decreases the counters of all the other peripheral nodes by 1 (but not below 0). Thereafter all the copies on peripheral nodes with a counter value of 0 are deleted.

We assume that initially no peripheral node holds a copy and therefore all the counters are initialized with 0. Count can be implemented without the need of any extra communication by maintaining a counter c^i on peripheral node i while it holds a copy of the file and on the center node otherwise (c.f. the distributed implementation in Section 4.1).

The change of the ranges of the counters compared to the version in [BFR92] is solely necessary because of the different cost measure for the deletion of copies. While in [BFR92] deletions are free, we charge a cost of d^i_t for the deletion of the copy on peripheral node i. So after a copy was deleted at least $D+1$ read requests have to be issued until the creation and deletion of a new copy is amortized.

Algorithm 2 COUNT

```
Initialize c^i := 0 for all i ∈ {1, ..., n}.
On request (d_t, a_t) do
    Let i be the source of the current request a_t.
    if i ∈ {1, ..., n} then
        c^i := min(c^i + 1, D + 1)
        if ( c^i = D + 1 and i ∉ R_C ) then
            Replicate to i.
        end if
    end if
    if a_t = w_i then
        for j ∈ {1, ..., n} \ {i} do
            c^j := max(c^j − 1, 0)
            if c^j = 0 and j ∈ R_C then
                Delete the copy on j.
            end if
        end for
    end if
end
```

Theorem 3.4. *Let σ be an arbitrary input sequence and* OPT *be an arbitrary optimal offline algorithm for* FADS. *Then*

$$C_C(\sigma) \leq \sum_{t=1}^{|\sigma|} \gamma_C^r(\bar{\delta}_t) \cdot C_{OPT}(\sigma_t)$$

holds for any $0 \leq r \leq 1$, where $\bar{\delta}_t$ is the average change of the distances in step t, i.e. $\bar{\delta}_t := \frac{1}{n} \sum_{i=1}^{n} \left| d_t^i - d_{t-1}^i \right|$ and

$$\gamma_C^r(\bar{\delta}_t) := \max\left(rD + (3-r),\ 1 + \frac{(3-r)D + r}{p} \bar{\delta}_t \right).$$

Notice that the proof of Theorem 3.4 requires that r is fixed during the whole input sequence.

Corollary 3.5. COUNT *is strictly $\gamma_C(\delta)$-competitive against δ-restricted adversaries, where*

$$\gamma_C(\delta) := \min_{0 \leq r \leq 1} \gamma_C^r(\delta) = \begin{cases} 3 & \text{if } \delta \leq \frac{2}{3D} p, \\ \frac{p + 3(D+1)\delta}{p + \delta} & \text{if } \frac{2}{3D} p < \delta < \frac{D+1}{2D+1} p, \\ 1 + \frac{2D+1}{p} \delta & \text{if } \delta \geq \frac{D+1}{2D+1} p. \end{cases}$$

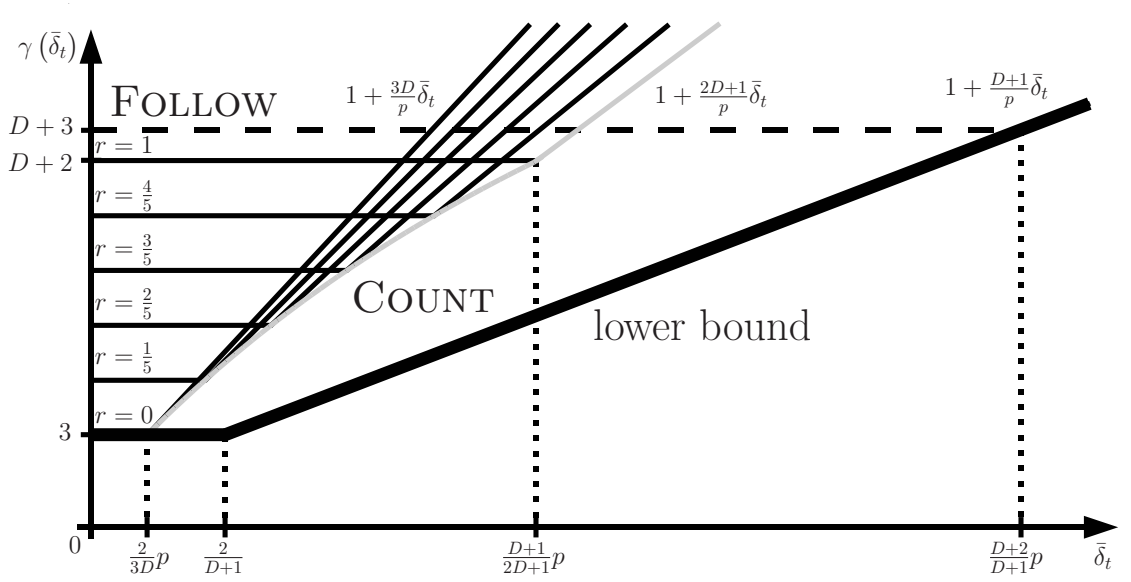


Figure 3.5: The amortized competitive ratios of Count for different values of r and Follow. The gray curve gives the competitive ratio of Count against $\bar{\delta}_t$-restricted adversaries. The bold curve depicts the lower bound for demand-driven algorithms.

Figure 3.5 shows the amortized competitive ratios $\gamma'_{\mathrm{C}}(\bar{\delta}_t)$ for different values of r compared to the amortized competitive ratio $\gamma_{\mathrm{F}}(\bar{\delta}_t)$ of Follow and the lower bound for demand-driven algorithms. It shows that for small values of $\bar{\delta}_t$ the competitive ratio of Count is optimal and the one of Follow depends on D and is therefore dissatisfactory. On the other hand the competitive ratio of Follow is optimal for large $\bar{\delta}_t$ while the competitive ratio of Count increases faster with increasing $\bar{\delta}_t$. Thus, Follow seems to be better in scenarios with high mobility of the peripheral nodes. The reason for Follow's good performance for large $\bar{\delta}_t$ is that it handles the lower bound for demand-driven algorithms (c.f. Subsection 3.1.2.1) in the best way a competitive demand-driven algorithm can.

Proof of Theorem 3.4. We show that the amortized competitive ratio of any step t of any input σ is smaller than or equal to $\max\left(rD + (3-r),\ 1 + \frac{(3-r)D+r}{p}\bar{\delta}_t\right)$ for an arbitrary but fixed $0 \le r \le 1$. The proof proceeds analogous to the proof of Theorem 3.2. We use the potential function $\Phi_t := \sum_{i=1}^{n} \Phi_t^i \cdot d_t^i$, where

$$\Phi_t^i := \begin{cases} 2c_t^i & \text{if } i \notin R_{\mathrm{C}}^{t-1} \text{ and } i \notin R_{\mathrm{Opt}}^{t-1}, \\ (3-r)D - c_t^i + r & \text{if } i \notin R_{\mathrm{C}}^{t-1} \text{ and } i \in R_{\mathrm{Opt}}^{t-1}, \\ c_t^i + 1 & \text{if } i \in R_{\mathrm{C}}^{t-1} \text{ and } i \notin R_{\mathrm{Opt}}^{t-1}, \\ (3-r)D - 2c_t^i + (r+1) & \text{if } i \in R_{\mathrm{C}}^{t-1} \text{ and } i \in R_{\mathrm{Opt}}^{t-1}. \end{cases}$$

Since $0 \leq c_t^i \leq D+1$ holds, we have $\Phi_t^i \geq 0$ for $1 \leq t \leq |\sigma|$ and $\Phi_0^i = 0$ for the natural case that COUNT and OPT start holding only a copy on the center node.

Before we proceed with the analysis of the two parts of a step, we need the following properties of COUNT.

Claim 3.6.

(a) If counter c^i is 0, then peripheral node i can not hold a copy.

(b) If counter c^i is $D+1$, then peripheral node i holds a copy.

Proof. We show both properties by induction over the number of steps. Initially all the counters are 0 and no peripheral node holds a copy. After the t-th step the following cases have to be considered: If counter c^i has not changed during step t, both claims follow from the induction hypothesis. If counter c^i has changed to 0 during step t, a node $j \neq i$ had issued a write request and thus COUNT ensures in line 10 that i does not hold a copy. If counter c^i has changed to $D+1$ during step t, i had issued a request and thus COUNT ensures in line 6 that i holds a copy. Otherwise c^i is neither 0 nor $D+1$ and therefore nothing is to be shown. □

$i \in R_C$	range of c^i	$i \in R_{OPT}$	$\Phi^i \leq$
no	$0 \leq c^i \leq D$	no	$2D$
no	$0 \leq c^i \leq D$	yes	$(3-r)D + r$
yes	$1 \leq c^i \leq D+1$	no	$D+2$
yes	$1 \leq c^i \leq D+1$	yes	$(3-r)D + (r-1)$

Table 3.2: Upper bounds of Φ^i for the different combinations of R_C, c^i and R_{OPT}.

With the help of the above claim, we can derive Table 3.2 and hence for the first part of any step t (movement and stand-by costs) holds

$$\begin{aligned} np + \Phi_{t-\frac{1}{2}} - \Phi_{t-1} &= np + \sum_{i=1}^{n} \Phi_{t-1}^i \cdot (d_t^i - d_{t-1}^i) \\ &\overset{\text{Table 3.2}}{\leq} np + \sum_{i=1}^{n} ((3-r)D + r) \cdot \left| d_t^i - d_{t-1}^i \right| \\ &= \left(1 + ((3-r)D + r) \frac{\sum_{i=1}^{n} \left| d_t^i - d_{t-1}^i \right|}{np} \right) np \\ &= \left(1 + \frac{(3-r)D + r}{p} \bar{\delta}_t \right) np. \end{aligned} \tag{3.5}$$

For the analysis of Count in the second part of a step (request and actions), we use the cost allocation scheme already used in the analysis of Follow. We show that for every peripheral node $1 \leq i \leq n$ holds

$$\tilde{C}^i_{\mathrm{C}}(\sigma_t) + \left(\Phi^i_t - \Phi^i_{t-\frac{1}{2}}0\right) d^i_t \leq (rD + (3-r)) \cdot \tilde{C}^i_{\mathrm{Opt}}(\sigma_t). \tag{3.6}$$

We show this in two steps: First, we analyse the costs and the change of Φ^i due to the request and the actions of Count. Thereafter, we have a look at the change of Φ^i caused by an action of Opt. The proof of (3.6) for the first step can be found in Table 3.4 and the proof for the second step in Table 3.3.

Table 3.3: Changes of replica set R_{Opt} and corresponding values of Φ^i, $\tilde{C}^i_{\mathrm{C}}$ and $\tilde{C}^i_{\mathrm{Opt}}$ regarding peripheral node i in the second part of step t.

$i \in R_{\mathrm{C}}$	c^i	$i \in R_{\mathrm{Opt}}$	$i \in R'_{\mathrm{Opt}}$	$\tilde{C}^i_{\mathrm{C}} + \Delta\Phi^i d^i_t$	$(rD + (3-r)) \cdot \tilde{C}^i_{\mathrm{Opt}}$
Opt creates a new copy on i.					
no	$c^i \geq 1$	no	yes	$((3-r)D - 3c^i + r)d^i_t \leq ((3-r)D - 3 + r)\, d^i_t$	$(rD + (3-r)) \cdot Dd^i_t$
yes	$c^i \geq 0$	no	yes	$((3-r)D - 3c^i + r)d^i_t \leq ((3-r)D + r)\, d^i_t$	$(rD + (3-r)) \cdot Dd^i_t$
Opt deletes copy on i.					
no	$c^i \leq D + 1$	yes	no	$(-(3-r)D + c^i + r)d^i_t \leq (rD + 3 - r)\, d^i_t$	$(rD + (3-r)) \cdot d^i_t$
yes	$c^i \leq D$	yes	no	$((3-r)D - c^i + 2 + r)d^i_t \leq (rD - r)\, d^i_t$	$(rD + (3-r)) \cdot d^i_t$

Together, this results in

$$\begin{aligned}
C_{\mathrm{C}}(\sigma) &\leq C_{\mathrm{C}}(\sigma) + \Phi_{|\sigma|} - \Phi_0 \\
&= \sum_{t=1}^{|\sigma|} \left(np + \tilde{C}_{\mathrm{C}}(\sigma_t)\right) + \Phi_{|\sigma|} - \Phi_0 \\
&= \sum_{t=1}^{|\sigma|} \left(\left(np + \Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right) + \left(\tilde{C}_{\mathrm{C}}(\sigma_t) + \Phi_t - \Phi_{t-\frac{1}{2}}\right)\right) \\
&\overset{(*)}{\leq} \sum_{t=1}^{|\sigma|} \left(\left(1 + \frac{(3-r)D + r}{p}\bar{\delta}_t\right) np + (rD + (3-r)) \cdot \tilde{C}_{\mathrm{Opt}}(\sigma_t)\right) \\
&\leq \sum_{t=1}^{|\sigma|} \max\left(rD + (3-r),\ 1 + \frac{(3-r)D + r}{p}\bar{\delta}_t\right) C_{\mathrm{Opt}}(\sigma_t),
\end{aligned}$$

where $(*)$ follows from (3.5) and (3.6). □

Table 3.4: Changes of configuration $\left(R_{\mathrm{C}},\ c^i\right)$ and corresponding values of $\Delta\Phi^i$, $\tilde{C}^i_{\mathrm{C}}$ and $\tilde{C}^i_{\mathrm{OPT}}$ regarding peripheral node i in the second part of step t on request a_t.

$i \in R_{\mathrm{C}}$	c^i	$i \in R'_{\mathrm{C}}$	$c^{i\prime}$	$i \in R_{\mathrm{OPT}}$	$\tilde{C}^i_{\mathrm{C}}(\sigma_t)$	$\Delta\Phi^i d^i_t$	$\tilde{C}^i_{\mathrm{C}}(\sigma_t) + \Delta\Phi^i d^i_t$	$(rD + (3-r))\,\tilde{C}^i_{\mathrm{OPT}}(\sigma_t)$
$a_t = r_j$ with $j \neq i$								
no	$\leq D$	no	c^i	no	0	$(2c^i - 2c^i)d^i_t$	0	0
no	$\leq D$	no	c^i	yes	0	$(((3-r)D - c^i + r) - ((3-r)D - c^i + r))d^i_t$	0	0
yes	≥ 1	yes	c^i	no	0	$((c^i + 1) - (c^i + 1))d^i_t$	0	0
yes	≥ 1	yes	c^i	yes	0	$(((3-r)D - 2c^i + (1+r)) - ((3-r)D - 2c^i + (1+r)))d^i_t$	0	0
$a_t = r_i$								
no	$\leq D-1$	no	$c^i + 1$	no	d^i_t	$((2(c^i + 1)) - 2c^i)d^i_t$	$3d^i_t$	$(rD + (3-r))\,d^i_t$
no	D	yes	$D+1$	no	$(D+1)d^i_t$	$(((D+1) + 1) - 2D)d^i_t$	$3d^i_t$	$(rD + (3-r))\,d^i_t$
no	$\leq D-1$	no	$c^i + 1$	yes	d^i_t	$(((3-r)D - (c^i + 1) + r) - ((3-r)D - c^i + r))d^i_t$	0	0
no	D	yes	$D+1$	yes	$(D+1)d^i_t$	$(((3-r)D - 2(D+1) + (1+r)) - ((3-r)D - D + r))d^i_t$	0	0
yes	$\leq D$	yes	$c^i + 1$	no	0	$(((c^i + 1) + 1) - (c^i + 1))d^i_t$	d^i_t	$(rD + (3-r))\,d^i_t$
yes	$D+1$	yes	$D+1$	no	0	$(((D+1) + 1) - ((D+1) + 1))d^i_t$	0	$(rD + (3-r))\,d^i_t$
yes	$\leq D$	yes	$c^i + 1$	yes	0	$(((3-r)D - 2(c^i + 1) + (1+r)) - ((3-r)D - 2c^i + (1+r)))d^i_t$	$-2d^i_t$	0
yes	$D+1$	yes	$D+1$	yes	0	$(((3-r)D - 2(D+1) + (1+r)) - ((3-r)D - 2(D+1) + (1+r)))d^i_t$	0	0
$a_t = w_j$ with $j \neq i$								
no	0	no	0	no	0	$(0 - 0)d^i_t$	0	0
n o	≥ 1	no	$c^i - 1$	no	0	$(2(c^i - 1) - 2c^i)d^i_t$	$-2d^i_t$	0
no	0	no	0	yes	0	$(((3-r)D - 0 + r) - ((3-r)D - 0 + r))d^i_t$	0	$(rD + (3-r))\,d^i_t$
no	≥ 1	no	$c^i - 1$	yes	0	$(((3-r)D - (c^i - 1) + r) - ((3-r)D - c^i + r))d^i_t$	d^i_t	$(rD + (3-r))\,d^i_t$
yes	1	no	0	no	$2d^i_t$	$(0 - (1 + 1))d^i_t$	0	0
yes	≥ 2	yes	$c^i - 1$	no	d^i_t	$(c^i - (c^i + 1))d^i_t$	0	0
yes	1	no	0	yes	$2d^i_t$	$(((3-r)D - 0 + r) - ((3-r)D - 2 + (1+r)))d^i_t$	$3d^i_t$	$(rD + (3-r))\,d^i_t$
yes	≥ 2	yes	$c^i - 1$	yes	d^i_t	$(((3-r)D - 2(c^i - 1) + (1+r)) - ((3-r)D - 2c^i + (1+r)))d^i_t$	$3d^i_t$	$(rD + (3-r))\,d^i_t$
$a_t = w_i$								
no	$\leq D-1$	no	$c^i + 1$	no	d^i_t	$(2(c^i + 1) - 2c^i)d^i_t$	$3d^i_t$	$(rD + (3-r))\,d^i_t$
no	D	yes	$D+1$	no	$(D+1)d^i_t$	$(((D+1) + 1) - 2D)d^i_t$	$3d^i_t$	$(rD + (3-r))\,d^i_t$
no	$\leq D-1$	no	$c^i + 1$	yes	d^i_t	$(((3-r)D - (c^i + 1) + r) - ((3-r)D - c^i + r))d^i_t$	0	$(rD + (3-r))\,d^i_t$
no	D	yes	$D+1$	yes	$(D+1)d^i_t$	$(((3-r)D - 2(D+1) + (1+r)) - ((3-r)D - D + r))d^i_t$	0	$(rD + (3-r))\,d^i_t$
yes	$\leq D$	yes	$c^i + 1$	no	d^i_t	$(((c^i + 1) + 1) - (c^i + 1))d^i_t$	$2d^i_t$	$(rD + (3-r))\,d^i_t$
yes	$D+1$	yes	$D+1$	no	d^i_t	$(((D+1) + 1) - ((D+1) + 1))d^i_t$	d^i_t	$(rD + (3-r))\,d^i_t$
yes	$\leq D$	yes	$c^i + 1$	yes	d^i_t	$(((3-r)D - 2(c^i + 1) + (1+r)) - ((3-r)D - 2c^i + (1+r)))d^i_t$	$-d^i_t$	$(rD + (3-r))\,d^i_t$
yes	$D+1$	yes	$D+1$	yes	d^i_t	$(((3-r)D - 2(D+1) + (1+r)) - ((3-r)D - 2(D+1) + (1+r)))d^i_t$	d^i_t	$(rD + (3-r))\,d^i_t$

3.1.2.4 Algorithm RandomizedFollow

Algorithm RandomizedFollow[4] (see Algorithm 3) is a randomized extension of Follow. It consists of the following two independent rules:

(i) If a request is issued at a peripheral node which does not hold a copy, place a new copy on this peripheral node with probability p_r.

(ii) If a node i issues a write request, each copy on a peripheral node other than i is deleted with probability p_d.

We denote by RandomizedFollow$_l$, with $l \geq 1$, the instance of RandomizedFollow with $p_r := \frac{l}{D+l}$ and $p_d := \frac{l}{D+l}$.

Algorithm 3 RandomizedFollow

```
1: On request (d_t, a_t) do
2:     Let i be the source of the current request a_t.
3:     if i ∉ R_RF then
4:         Replicate to i with probability p_r.
5:     end if
6:     if a_t = w_i then
7:         for j ∈ R_RF \ {c, i} do
8:             Delete copy on j with probability p_d.
9:         end for
10:    end if
11: end
```

Theorem 3.7. *Let σ be an arbitrary input sequence and* Adv *be an adaptive-online adversary for* Fads. *Then*

$$\mathrm{E}\left[C_{\mathrm{RF}_l}(\sigma)\right] \leq \sum_{t=1}^{|\sigma|} \gamma^r_{\mathrm{RF}_l}(\bar{\delta}_t) \cdot \mathrm{E}\left[C_{\mathrm{Adv}}(\sigma_t)\right]$$

holds for any $0 \leq r \leq \frac{1}{l}$, where $\bar{\delta}_t$ is the average change of the distances of the peripheral nodes in step t, i.e. $\bar{\delta}_t := \frac{1}{n}\sum_{i=1}^{n}\left|d^i_t - d^i_{t-1}\right|$ and

$$\gamma^r_{\mathrm{RF}_l}(\bar{\delta}_t) := \max\left(\left(1 - \frac{2}{l+1} + r\right)D + 3 - r,\ 1 + \frac{\left(1 + \frac{2}{l} - r\right)D + 1 - \frac{1}{l} + r}{p}\bar{\delta}_t\right).$$

[4] We abbreviate RandomizedFollow with RF in indices.

Notice that the expected number of read respectively write requests until RANDOMIZEDFOLLOW$_1$ creates respectively deletes a copy is $D+1$. Thus, the behavior of RANDOMIZEDFOLLOW$_1$ is basically similar to the one of COUNT. And in fact, the expected amoritzed competitive ratio $\max\left(rD+3-r,\ 1+\frac{(3-r)D+r}{p}\bar{\delta}_t\right), 0 \le r \le 1$, of RANDOMIZEDFOLLOW$_1$ is exactly the amortized competitive ratio of COUNT. Furthermore, if $l \to \infty$ both p_r and p_d approach 1 and therefore RANDOMIZEDFOLLOW approaches FOLLOW. And accordingly the amortized competitive ratio approaches $\max\left(D+3,\ 1+\frac{D+1}{p}\bar{\delta}_t\right)$, which is the amortized competitive ratio of FOLLOW.

Corollary 3.8. RANDOMIZEDFOLLOW$_l$ *is strictly* $\gamma_{\mathrm{RF}_l}(\delta)$*-competitive against* δ*-restricted adaptive-online adversaries, where*

$$\gamma_{\mathrm{RF}_l}(\delta) := \begin{cases} \left(1-\frac{2}{l+1}\right)D+3 & \text{if } \delta \le \frac{l\left(1-\frac{2}{l+1}\right)D+2l}{(l+2)D+l-1}p, \qquad (r=0) \\ \frac{1+\left(\left(2+\frac{2}{l}-\frac{2}{l+1}\right)D+4-\frac{1}{l}\right)\frac{\delta}{p}}{1+\frac{\delta}{p}} & \text{if } \frac{l\left(1-\frac{2}{l+1}\right)D+2l}{(l+2)D+l-1}p < \delta < \frac{l\left(1-\frac{2}{l+1}\right)D+2l+(D-1)}{(l+2)D+l-1-(D-1)}p, \quad (0<r<\frac{1}{l}) \\ 1+\left(\left(1+\frac{1}{l}\right)D+1\right)\frac{\delta}{p} & \text{if } \delta \ge \frac{l\left(1-\frac{2}{l+1}\right)D+2l+(D-1)}{(l+2)D+l-1-(D-1)}p. \qquad (r=\frac{1}{l}) \end{cases}$$

Proof of Theorem 3.7. We will prove the theorem with the help of the potential function $\Phi_t := \sum_{i=1}^{n} \Phi_t^i \cdot d_t^i$, where

$$\Phi_t^i := \begin{cases} 0 & \text{if } i \notin R_{\mathrm{RF}_l} \text{ and } i \notin R_{\mathrm{ADV}}, \\ \frac{1}{l}D+2 & \text{if } i \in R_{\mathrm{RF}_l} \text{ and } i \notin R_{\mathrm{ADV}}, \\ \left(1+\frac{2}{l}-r\right)D+1-\frac{1}{l}+r & \text{if } i \notin R_{\mathrm{RF}_l} \text{ and } i \in R_{\mathrm{ADV}}, \\ \left(\frac{1}{l}-r\right)D-\frac{1}{l}+r & \text{if } i \in R_{\mathrm{RF}_l} \text{ and } i \in R_{\mathrm{ADV}}. \end{cases}$$

Notice that $\Phi_t \ge 0$ for $1 \le t \le |\sigma|$ and especially $\Phi_0 = 0$ in the natural case that RANDOMIZEDFOLLOW and ADV start without copies on peripheral nodes.

We divide every step into two parts. In the first part both RANDOMIZEDFOLLOW$_l$ and ADV pay p for every peripheral node and the peripheral nodes move to their new positions. In the second part the file access is served and the algorithms may reallocate copies. We denote by $\Phi_{t-\frac{1}{2}}$ the potential after the first part of step t, i.e. $\Phi_{t-\frac{1}{2}} = \sum_{i=1}^{n} \Phi_{t-1}^i \cdot d_t^i$.

Below we show that for the first part of any step t holds:

$$np + \mathrm{E}\left[\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right] \le \left(1+\frac{\left(1+\frac{2}{l}-r\right)D+1-\frac{1}{l}+r}{p}\bar{\delta}_t\right)np. \tag{3.7}$$

And for the second part, we show:

$$\mathrm{E}\left[\tilde{C}_{\mathrm{RF}_l}(\sigma_t) + \Phi_t - \Phi_{t-\frac{1}{2}}\right] \le \left(\left(1-\frac{2}{l+1}+r\right)D+3-r\right)\cdot \mathrm{E}\left[\tilde{C}_{\mathrm{ADV}}(\sigma_t)\right]. \tag{3.8}$$

Inequality (3.8) especially says that RandomizedFollow$_l$ is $\left(\left(1-\frac{2}{l+1}+r\right)D+3-r\right)$-competitive against adaptive-online adversaries in a static star network.

Together this results in

$$
\begin{aligned}
&\mathrm{E}\left[C_{\mathrm{RF}_l}(\sigma)\right] \\
&\leq \mathrm{E}\left[C_{\mathrm{RF}_l}(\sigma)\right] + \mathrm{E}\left[\Phi_{|\sigma|} - \Phi_0\right] \\
&= \mathrm{E}\left[\sum_{t=1}^{|\sigma|}\left(np + \tilde{C}_{\mathrm{RF}_l}(\sigma_t)\right) + \Phi_{|\sigma|} - \Phi_0\right] \\
&= \mathrm{E}\left[\sum_{t=1}^{|\sigma|}\left(np + \left(\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right) + \left(\tilde{C}_{\mathrm{RF}_l}(\sigma_t) + \Phi_t - \Phi_{t-\frac{1}{2}}\right)\right)\right] \\
&= \sum_{t=1}^{|\sigma|}\left(\left(np + \mathrm{E}\left[\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right]\right) + \mathrm{E}\left[\tilde{C}_{\mathrm{RF}_l}(\sigma_t) + \Phi_t - \Phi_{t-\frac{1}{2}}\right]\right) \\
&\overset{(*)}{\leq} \sum_{t=1}^{|\sigma|}\left(\left(1 + \frac{\left(1+\frac{2}{l}-r\right)D + 1 - \frac{1}{l} + r}{p}\bar{\delta}_t\right)np + \left(\left(1-\frac{2}{l+1}+r\right)D+3-r\right)\mathrm{E}\left[\tilde{C}_{\mathrm{Adv}}(\sigma_t)\right]\right) \\
&\leq \sum_{t=1}^{|\sigma|}\max\left(\left(1-\frac{2}{l+1}+r\right)D+3-r,\ 1 + \frac{\left(1+\frac{2}{l}-r\right)D + 1 - \frac{1}{l} + r}{p}\bar{\delta}_t\right)\mathrm{E}\left[C_{\mathrm{Adv}}(\sigma_t)\right],
\end{aligned}
$$

where $(*)$ follows from (3.7) and (3.8).

For the proof of (3.7), we observe that the costs of RandomizedFollow$_l$ and Adv in the first phase of a step are always np. Furthermore, the value of Φ changes only due to the changes of the distances since no actions are performed. Thus, we have:

$$
\begin{aligned}
np + \mathrm{E}\left[\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right] &= np + \mathrm{E}\left[\sum_{i=1}^{n}\Phi_{t-1}^i \cdot (d_t^i - d_{t-1}^i)\right] \\
&\leq np + \sum_{i=1}^{n}\left(\left(1+\frac{2}{l}-r\right)D + 1 - \frac{1}{l} + r\right)\cdot\left|d_t^i - d_{t-1}^i\right| \\
&= \left(1 + \frac{\left(1+\frac{2}{l}-r\right)D + 1 - \frac{1}{l} + r}{np}\sum_{i=1}^{n}\left|d_t^i - d_{t-1}^i\right|\right)np \\
&= \left(1 + \frac{\left(1+\frac{2}{l}-r\right)D + 1 - \frac{1}{l} + r}{p}\bar{\delta}_t\right)np
\end{aligned}
$$

For the analysis of RandomizedFollow$_l$ in the second part of a step, we use the cost allocation scheme already used in the analysis of Follow and Count. We prove (3.8) by showing that for all $1 \leq i \leq n$ holds

$$
\mathrm{E}\left[\tilde{C}_{\mathrm{RF}}^i(\sigma_t) + \left(\Phi_t^i - \Phi_{t-\frac{1}{2}}^i\right)d_t^i\right] \leq \left(\left(1-\frac{2}{l+1}+r\right)D+3-r\right)\cdot\mathrm{E}\left[\tilde{C}_{\mathrm{Adv}}^i(\sigma_t)\right]. \tag{3.9}
$$

Table 3.5: Changes of replica set R_{RF_l} and corresponding values of $\mathrm{E}\left[\Delta\Phi^i d_t^i\right]$, $\mathrm{E}\left[\tilde{C}^i_{\mathrm{RF}_l}\right]$ and $\tilde{C}^i_{\mathrm{Adv}}$ regarding peripheral node i in the second part of step t on request a_t.

$i \in R_{\mathrm{RF}_l}$	$i \in R_{\mathrm{Adv}}$	$\mathrm{E}\left[\tilde{C}^i_{\mathrm{RF}_l}(\sigma_t)\right]$	$\Phi^i d_t^i$	$\mathrm{E}\left[\Phi^{i'} d_t^i\right]$	$\mathrm{E}\left[\tilde{C}^i_{\mathrm{RF}_l}(\sigma_t) + \Delta\Phi^i d_t^i\right]$	$\left(\left(1 - \frac{2}{l+1} + r\right)D + 3 - r\right)\tilde{C}^i_{\mathrm{Adv}}(\sigma_t)$
$a_t = r_j$ with $j \neq i$						
no	no	0	0	$\Phi^i d_t^i$	0	0
no	yes	0	$\left(\left(1 + \frac{2}{l} - r\right)D + 1 - \frac{1}{l} + r\right) d_t^i$	$\Phi^i d_t^i$	0	0
yes	no	0	$\left(\frac{1}{l}D + 2\right) d_t^i$	$\Phi^i d_t^i$	0	0
yes	yes	0	$\left(\left(\frac{1}{l} - r\right)D - \frac{1}{l} + r\right)D$	$\Phi^i d_t^i$	0	0
$a_t = r_i$						
no	no	$\left(1 + \frac{l}{D+l}D\right) d_t^i$	0	$\frac{l}{D+l}\left(\frac{1}{l}D + 2\right) d_t^i + \left(1 - \frac{l}{D+l}\right)\Phi^i d_t^i$	$\left(2 + \frac{l}{D+l}(D+1)\right) d_t^i$	$\left(\left(1 - \frac{2}{l+1} + r\right)D + 3 - r\right) d_t^i$
no	yes	$\left(1 + \frac{l}{D+l}D\right) d_t^i$	$\left(\left(1 + \frac{2}{l} - r\right)D + 1 - \frac{1}{l} + r\right) d_t^i$	$\frac{l}{D+l}\left(\left(\frac{1}{l} - r\right)D - \frac{1}{l} + r\right) d_t^i + \left(1 - \frac{l}{D+l}\right)\Phi^i d_t^i$	0	0
yes	no	0	$\left(\frac{1}{l}D + 2\right) d_t^i$	$\Phi^i d_t^i$	0	$\left(\left(1 - \frac{2}{l+1} + r\right)D + 3 - r\right) d_t^i$
yes	yes	0	$\left(\left(\frac{1}{l} - r\right)D - \frac{1}{l} + r\right)D$	$\Phi^i d_t^i$	0	0
$a_t = w_j$ with $j \neq i$						
no	no	0	0	$\Phi^i d_t^i$	0	0
no	yes	0	$\left(\left(1 + \frac{2}{l} - r\right)D + 1 - \frac{1}{l} + r\right) d_t^i$	$\Phi^i d_t^i$	0	$\left(\left(1 - \frac{2}{l+1} + r\right)D + 3 - r\right) d_t^i$
yes	no	$\left(1 + \frac{l}{D+l}\right) d_t^i$	$\left(\frac{1}{l}D + 2\right) d_t^i$	$0 - \left(1 - \frac{l}{D+l}\right)\Phi^i d_t^i$	0	0
yes	yes	$\left(1 + \frac{l}{D+l}\right) d_t^i$	$\left(\left(\frac{1}{l} - r\right)D - \frac{1}{l} + r\right)D$	$\frac{l}{D+l}\left(\left(1 + \frac{2}{l} - r\right)D + 1 - \frac{1}{l} + r\right) d_t^i + \left(1 + \frac{l}{D+l}D\right)\Phi^i d_t^i$	$\left(2 + \frac{l}{D+l}(D+1)\right) d_t^i$	$\left(\left(1 - \frac{2}{l+1} + r\right)D + 3 - r\right) d_t^i$
$a_t = w_i$						
no	no	$\left(1 + \frac{l}{D+l}D\right) d_t^i$	0	$\frac{l}{D+l}\left(\frac{1}{l}D + 2\right) d_t^i + \left(1 - \frac{l}{D+l}\right)\Phi^i d_t^i$	$\left(2 + \frac{l}{D+l}(D+1)\right) d_t^i$	$\left(\left(1 - \frac{2}{l+1} + r\right)D + 3 - r\right) d_t^i$
no	yes	$\left(1 + \frac{l}{D+l}D\right) d_t^i$	$\left(\left(1 + \frac{2}{l} - r\right)D + 1 - \frac{1}{l} + r\right) d_t^i$	$\frac{l}{D+l}\left(\left(\frac{1}{l} - r\right)D - \frac{1}{l} + r\right) d_t^i + \left(1 - \frac{l}{D+l}\right)\Phi^i d_t^i$	0	$\left(\left(1 - \frac{2}{l+1} + r\right)D + 3 - r\right) d_t^i$
yes	no	d_t^i	$\left(\frac{1}{l}D + 2\right) d_t^i$	$\Phi^i d_t^i$	d_t^i	$\left(\left(1 - \frac{2}{l+1} + r\right)D + 3 - r\right) d_t^i$
yes	yes	d_t^i	$\left(\left(\frac{1}{l} - r\right)D - \frac{1}{l} + r\right)D$	$\Phi^i d_t^i$	d_t^i	$\left(\left(1 - \frac{2}{l+1} + r\right)D + 3 - r\right) d_t^i$

We show this in two steps: First we analyze the costs and the change of Φ^i due to the request and the actions of RandomizedFollow$_l$. Thereafter we have a look at the change of Φ^i caused by an action of Adv. The proof of (3.9) for the first step can be found in Table 3.5 and the one for the second step in Table 3.6. □

Table 3.6: Changes of replica set R_{Adv} and corresponding values of $\Delta\Phi^i$ and $\tilde{C}^i_{\text{Adv}}$ regarding peripheral node i in the second part of step t.

$i \in R_{\text{RF}}$	$i \in R_{\text{Adv}}$	$i \in R'_{\text{Adv}}$	$\Delta\Phi^i d^i_t$	$\left(rD + 2 + l - r\frac{l^2+1}{l+1}\right) \cdot \tilde{C}^i_{\text{Adv}}$
Adv creates a new copy on i.				
no	no	yes	$\left((1 + \frac{2}{l} - r)D + 1 - \frac{1}{l} + r\right) d^i_t$	$\left((1 - \frac{2}{l+1} + r)D + 3 - r\right) \cdot D d^i_t$
yes	no	yes	$\left(-rD - 2 - \frac{1}{l} + r\right) d^i_t$	$\left((1 - \frac{2}{l+1} + r)D + 3 - r\right) \cdot D d^i_t$
Adv deletes copy on i.				
no	yes	no	$\left((-1 - \frac{2}{l} + r)D - 1 + \frac{1}{l} - r\right) d^i_t$	$\left((1 - \frac{2}{l+1} + r)D + 3 - r\right) \cdot d^i_t$
yes	yes	no	$\left(rD + 2 + \frac{1}{l} - r\right) d^i_t$	$\left((1 - \frac{2}{l+1} + r)D + 3 - r\right) \cdot d^i_t$

3.1.3 Lower bound for non-demand-driven algorithms

In the previous section we showed that every algorithm from the natural class of demand-driven algorithms has a competitive ratio of $\Omega\left(\frac{D}{p}\delta\right)$ and gave examples of demand-driven algorithms with a competitive ratio of $\Theta\left(\frac{D}{p}\delta\right)$. Here we give a slightly worse lower bound for arbitrary randomized online algorithms against δ-restricted oblivious adversaries. Thereby, we have to regard that a given algorithm may create a copy on a peripheral node even though this node is far away from the recent origins of requests.

Theorem 3.9. *The competitive ratio of any randomized algorithm for* Fads *against δ-restricted oblivious adversaries is at least* $\max\left(2 + \frac{1}{D},\ 1 + \frac{\frac{D}{p}\delta}{8\left\lceil \lg\left(1+\frac{D}{p}\delta\right)\right\rceil}\right)$.

This theorem especially implies that there can be no competitive algorithm for Fads if $\delta > 0$ and $p = 0$ as discussed in Chapter 2.

Proof. Let Alg be an arbitrary randomized algorithm for Fads. In the case that $1 + \frac{\frac{D}{p}\delta}{8\left\lceil \lg\left(1+\frac{D}{p}\delta\right)\right\rceil} \leq 2 + \frac{1}{D}$, the lower bound can be deduced from the lower bound for static networks given in [LRWY99]. For the other case, we describe an adversary Adv who constructs an input sequence consisting of an arbitrary number of phases. We show that the expected costs of Alg are at least $\left(1 + \frac{\frac{D}{p}\delta}{8\left\lceil \lg\left(1+\frac{D}{p}\delta\right)\right\rceil}\right)$-times

the costs of an optimal offline algorithm in every phase. We thereby upper bound the costs of an optimal offline algorithm by describing how ADV would serve the generated input.

Every phase starts with all the peripheral nodes at distance $S\delta$ for some fixed $S \geq 1$. During the phase all the peripheral nodes perform the same movement, i.e. $d_t^i = d_t^j$ for all $1 \leq i, j \leq n$ and $t \geq 0$.

Since ADV is oblivious, it does not know on which peripheral nodes ALG holds copies, but it can compute the probability that a peripheral node holds a copy. We denote by q_t^i the probability that peripheral node i holds a copy at the end of step t. Furthermore, we denote by $Q_t := \sum_{i=1}^{n} q_t^i$ the sum of these probabilities, i.e. the expected number of copies on peripheral nodes. Below we omit the index t and denote by q^i and Q the corresponding values at the end of the current step.

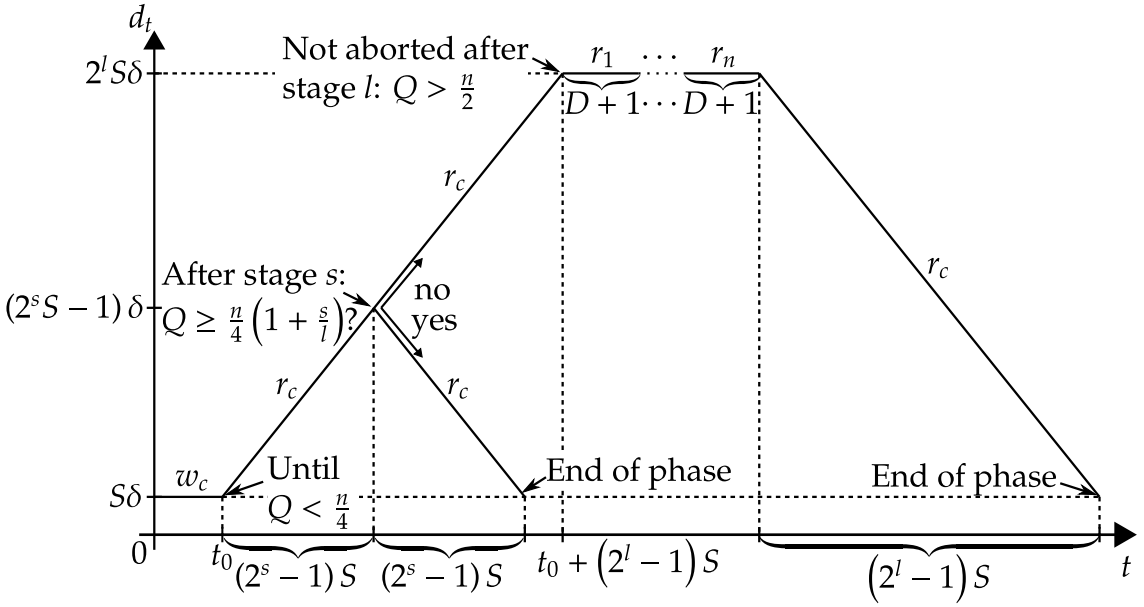


Figure 3.6: A single phase of the lower bound for general randomized algorithms.

A phase (see Figure 3.6) starts with a clearance-stage, in which ADV reduces the expected number of copies of ALG to less than $\frac{n}{4}$. ADV achieves this by issuing write requests from c until $Q < \frac{n}{4}$. Notice that ALG incurs an expected cost of at least $np + Q \cdot S\delta \geq np + \frac{n}{4} \cdot S\delta$ for every such write request. ADV on the other hand deletes all its copies on peripheral nodes at the end of each phase and therefore pays np in every step of the clearance stage. Thus, $E[C_{\text{ALG}}] \geq (1 + \frac{S}{4p}\delta)C_{\text{ADV}}$ holds for every step of the clearance stage. By choosing $S \geq \frac{D}{2\left\lceil \lg\left(1+\frac{D}{p}\delta\right)\right\rceil}$ we get $E[C_{\text{ALG}}] \geq (1 + \frac{\frac{D}{p}\delta}{8\left\lceil \lg\left(1+\frac{D}{p}\delta\right)\right\rceil})C_{\text{ADV}}$. We can therefore neglect the clearance stage in the

following analysis of the main part of the phase.

After the clearance-stage all the peripheral nodes move with speed δ away from the center. This expansion is divided into $l := \left\lceil \lg\left(1 + \frac{D}{p}\delta\right)\right\rceil$ stages. Stage s starts at distance $2^{s-1}S\delta$ and ends at distance $(2^sS - 1)\delta$. At the end of stage s Adv tests whether $Q \geq \frac{n}{4}(1 + \frac{s}{l})$. If so, Adv aborts the phase by moving the peripheral nodes back to the initial distance $S\delta$. Otherwise the next stage begins. If Alg reaches the end of stage l without abortion, i.e. the expected number of copies Q is lower than $\frac{n}{2}$, Adv issues consecutively D read requests from each of the peripheral nodes at distance $2^lS\delta$. Thereafter all the peripheral nodes move back to the initial distance $S\delta$.

Let us first assume that the phase is aborted after stage $1 \leq s \leq l$. Since there is no request from a peripheral node in an aborted phase, Adv does not create any copies. Thus, Adv pays $2(2^s - 1)Snp$ for the phase. For the expected costs of Alg we observe that Q is greater than $\frac{n}{4}(1 + \frac{s}{l})$ after stage s, i.e. it increased by at least $\frac{n}{4}\frac{s}{l}$ since the start of the expansion. Additionally, Q was smaller than or equal to $\frac{n}{4}(1 + \frac{r}{l})$ at the end of every stage $1 \leq r < s$, because the phase was not aborted in stage r. Thus, Q increased at most by $\frac{n}{4}\left(1 + \frac{r}{l}\right) - \frac{n}{4}\left(1 + \frac{r-1}{l}\right)$ in stage r. Since creating a copy gets more expensive over time, Alg would pay the least if it created copies as early as possible. Thus, the best behavior of Alg would be to create copies such that Q rises in the first step of stage r to $\frac{n}{4}\left(1 + \frac{r}{l}\right) - \epsilon$, where $\epsilon > 0$. Thus, we can lower bound the expected costs of Alg as follows:

$$
\begin{aligned}
E\left[C_{\text{Alg}}\right] &\geq \underbrace{2(2^s - 1)S}_{\text{\# of steps}} np + \sum_{r=1}^{s}\left(\frac{n}{4}\left(1 + \frac{r}{l}\right) - \frac{n}{4}\left(1 + \frac{r-1}{l}\right)\right) \cdot D \cdot 2^{r-1}S\delta \\
&= 2(2^s - 1)Snp + \frac{nD}{4l}S\delta\sum_{r=1}^{s}2^{r-1} \\
&= 2(2^s - 1)Snp + \frac{nD}{4l}\left(2^s - 1\right)S\delta \\
&= \left(1 + \frac{D}{p}\frac{1}{8l}\delta\right) \cdot 2\left(2^s - 1\right)Snp \\
&= \left(1 + \frac{\frac{D}{p}\delta}{8\left\lceil \lg\left(1 + \frac{D}{p}\delta\right)\right\rceil}\right)C_{\text{Adv}}
\end{aligned}
$$

In the other case Q is lower than $\frac{n}{2}$ after the l-th stage and hence Adv issues consecutively D read requests from every peripheral node. If Alg does not hold a copy on a peripheral node i at this time, it has to either send D single data units or to create a new copy on i. Thus, Alg incurs at least a cost of $D \cdot 2^lS\delta$ for peripheral node i. This implies an expected cost of at least $\sum_{i=1}^{n}(1 - q^i) \cdot D2^lS\delta > \frac{n}{2} \cdot D2^lS\delta$

for serving the read requests. ADV on the other hand creates copies on all the peripheral nodes at the first step of the expansion and deletes them at the end of the phase. Thus, the transfer costs of ADV are $n(D+1)S\delta$. Altogether we have

$$\begin{aligned}
\frac{E[C_{\text{ALG}}]}{C_{\text{ADV}}} &> \frac{\left(2(2^l-1)S+nD\right)np+\frac{n}{2}D2^lS\delta}{(2(2^l-1)S+nD)\,np+n(D+1)S\delta} \\
&= 1+\frac{\frac{n}{2}D2^lS\delta-n(D+1)S\delta}{(2(2^l-1)S+nD)\,np+n(D+1)S\delta} \\
&\underset{S\to\infty}{\to} 1+\frac{\frac{1}{2}D2^l\delta-(D+1)\delta}{2(2^l-1)p+(D+1)\delta} \\
&\geq 1+\frac{\frac{1}{2}D2^l\delta-2D\delta}{2(2^l-1)p+2D\delta} && (3.10)\\
&\geq 1+\frac{\frac{1}{2}D(1+\frac{D}{p}\delta)\delta-2D\delta}{2D\delta+2D\delta} && (3.11)\\
&= 1+\frac{\frac{D}{p}\delta}{8}-\frac{3}{8} \\
&> 1+\frac{\frac{D}{p}\delta}{8\left\lceil\lg\left(1+\frac{D}{p}\delta\right)\right\rceil}, && (3.12)
\end{aligned}$$

where (3.10) follows from $D \geq 1$ and (3.11) can be derived from $l \geq \lg\left(1+\frac{D}{p}\delta\right)$. Furthermore, (3.12) is valid for all $\frac{D}{p}\delta$ such that $1+\frac{\frac{D}{p}\delta}{8\left\lceil\lg\left(1+\frac{D}{p}\delta\right)\right\rceil} > 2+\frac{1}{D}$. □

3.2 File allocation in a dynamic tree network

In this section we extend the technique we used for analyzing algorithms in dynamic star networks to arbitrary dynamic tree networks. Such a dynamic tree network consists of a tree with a fixed topology where the weights of the edges can change over time.

3.2.1 Our model

The *file allocation problem in a dynamic tree network (*FADT*)* is defined as follows: We are given a tree network consisting of n nodes $1,\ldots,n$ which are interconnected via edges E. All the nodes of the tree can access an indivisible file which initially is stored on an arbitrary node $s \in \{1,\ldots,n\}$ of the tree. During the runtime of the system, an algorithm ALG can arbitrarily place copies of the file on the nodes. The *replica set* $R^t_{\text{ALG}} \subseteq \{1,\ldots,n\}$ is defined as the set of nodes where algorithm ALG has

a copy at the end of step t. There has to be at all times at least one copy in the tree, i.e $R^t_{\text{Alg}} \neq \emptyset$ for all $t \geq 0$.

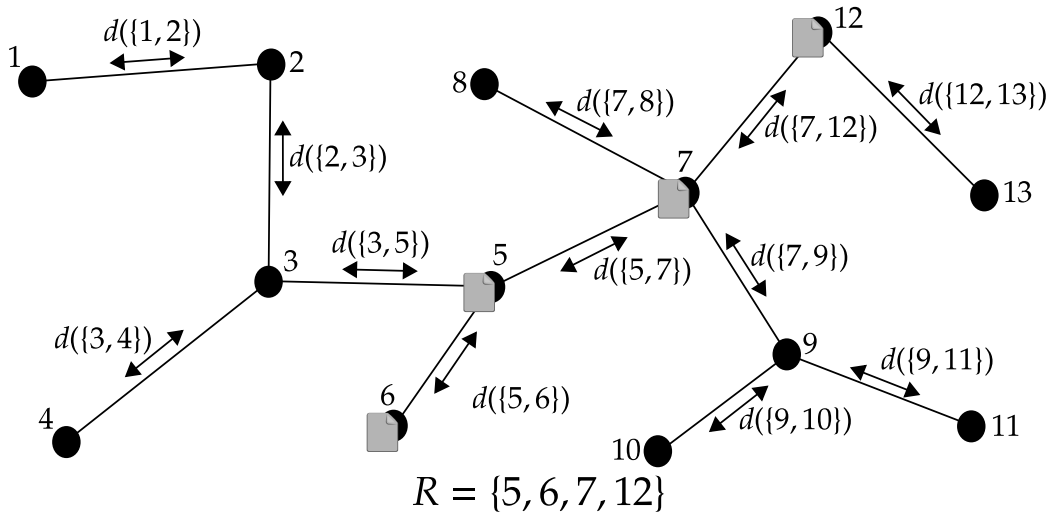


Figure 3.7: An example of a dynamic tree with 13 nodes.

The input consists of the initial edge weight function d_0 and a finite sequence σ of pairs (d_t, a_t), $1 \leq t \leq |\sigma|$, of an edge weight function and an access request. It is presented to the online algorithm in equidistant discrete time steps. The function $d_t : E \to \mathbb{R}^n_{>0}$ defines the edge weights in step t. The access request $a_t \in \{-, r_i, w_i \mid i \in \{1, \ldots, n\}\}$ either indicates that no access occurs $(-)$ or a read (r) respectively a write (w) access is issued at node i. Every step t is divided into the following three phases, which are executed strictly in the given order: First the edge weights are updated to d_t, then request a_t is served, and finally, the algorithm can create and/or delete copies.

We consider the shared file as an indivisible container of a fixed number of data units of uniform size. A single read respectively write access always affects just one unit of the file. While a read request can be fulfilled by a single copy, a write request has to update all the copies. Every transfer of such a data unit costs the corresponding distance.

The costs $C_{\text{Alg}}(\sigma_t)$ incurred by an algorithm Alg in step t are composed of three components: a fixed stand-by cost of p for every node, the transfer costs for fulfilling the data access and the transfer costs for creating and deleting copies. We will occasionally use the following split-up of the costs $C_{\text{Alg}}(\sigma_t) = np + \tilde{C}_{\text{Alg}}(\sigma_t)$, where $\tilde{C}_{\text{Alg}}(\sigma_t)$ denotes the costs incurred by Alg for serving the request and performing actions. The costs $\tilde{C}_{\text{Alg}}(a_t)$ for serving a data access from $i \in \{c, 1, \ldots, n\}$

are naturally defined by

$$\tilde{C}^t_{\text{ALG}}(-) := 0, \qquad \tilde{C}^t_{\text{ALG}}(r_i) := d_t\Big(i,\ R^{t-1}_{\text{ALG}}\Big) \qquad \tilde{C}^t_{\text{ALG}}(w_i) := st_t\Big(R^{t-1}_{\text{ALG}} \cup \{i\}\Big)$$

A replication of the file from node i to node j costs $D \cdot d_t(i,\ j)$, where $D \geq 1$ is a fixed constant depending on the size of the file. If an algorithm decides to delete the copy of a node i, it has to make sure that at least one copy remains in the tree. Thus, a communication between i and the current replica set R_{ALG} has to take place, which costs $d_t(i,\ R_{\text{ALG}} \setminus \{i\})$.[5] We denote by $C_{\text{ALG}}(\sigma)$ the overall costs incurred by algorithm ALG while serving input sequence σ.

3.2.2 Algorithm RANDOMIZEDTREE

The algorithm RANDOMIZEDTREE[6] (see Algorithm 4) was introduced and analyzed regarding static tree networks in [Bar94, BFR92]. We present here a version with slightly adapted probabilities, because our model involves costs for deletions. Notice that the set of replicas of RANDOMIZEDTREE R_{RT} is always a connected subtree.

Algorithm 4 RANDOMIZEDTREE

On request (d_t, a_t) **do**
  if ($a_t = r_i$ and $i \notin R_{\text{RT}}$) **then**
    Let $j \in R^{t-1}_{\text{RT}}$ be the nearest node to i with a copy.
    With probability $\frac{1}{D+1}$ place copies on the path from j to i.
  else if ($a_t = w_i$) **then**
    Let $j \in R^{t-1}_{\text{RT}}$ be the nearest node to i with a copy.
    With probability $\frac{1}{D+1}$ delete all copies but the one on j.
    With probability $\frac{1}{2}$ migrate the copy from j to i.
  end if
end

In the following, we abuse notation by writing $e \in \mathcal{ST}(R)$ and $i \in \mathcal{ST}(R)$ to denote either an edge or a node of the Steiner tree $\mathcal{ST}(R)$. It will be clear from the context whether an edge or a node is meant.

We need the following lemma in the analysis of RANDOMIZEDTREE, which easily follows from the uniqueness of Steiner trees in tree networks.

Lemma 3.10. *Let $S, T \subseteq \{1, \ldots, n\}$, $i \in \{1, \ldots, n\}$ and $t \in \mathbb{N}$. Then holds:*

$$st_t(S \cup \{i\}) = st_t(S) + d_t(i,\ \mathcal{ST}(S))\,. \tag{3.13}$$

$$st_t(S \cup T) = st_t(S) + st_t(T) - st_t(\mathcal{ST}(S) \cap \mathcal{ST}(T)) + d_t(\mathcal{ST}(S)\,,\ \mathcal{ST}(T)) \tag{3.14}$$

[5] This differs from the file allocation model of Bartal et al. [BFR92] where deletions are free.
[6] We abbreviate RANDOMIZEDTREE with RT in indices.

The following lemma allows us to assume in the analysis of RandomizedTree that the replica set of the adversary always constitutes a connected subtree.

Lemma 3.11. *Let* Alg *be an arbitrary file allocation algorithm for an arbitrary dynamic tree network* $(\{1,\dots,n\},\ E)$ *and* $s \in \{1,\dots,n\}$ *an arbitrary position of the initial copy. Furthermore, let* σ *be an arbitrary suitable input sequence and* $(\{s\}, R^1_{\text{Alg}}, \dots, R^{|\sigma|}_{\text{Alg}})$ *be the sequence of replica sets produced by* Alg *while serving* σ*. Then there is an algorithm* Alg′ *which produces the sequence of replica sets* $\left(\{s\}, \mathcal{ST}\left(R^1_{\text{Alg}}\right), \dots, \mathcal{ST}\left(R^\sigma_{\text{Alg}}\right)\right)$ *such that*

$$C_{\text{Alg}'}(\sigma) \le C_{\text{Alg}}(\sigma).$$

Proof. The proof of this lemma is mostly equivalent to the proof of Theorem 3.1 in [LRWY99]. We therefore discuss only the differences here. Their model differs from our model only in the stand-by costs, the deletion costs and the dynamics of the edge weights.

The stand-by costs are equal for Alg and Alg′ since they are completely oblivious of the behavior of an algorithm.

For the deletion costs of our model, assume that node i deletes its copy. Let j be the nearest copy in $R_{\text{Alg}} \setminus \{i\}$ to i. Then Alg′ deletes all its copies on the path from i to j which do not belong to $\mathcal{ST}(R_{\text{Alg}} \setminus \{i\})$. Thus, $C_{\text{Alg}'} \le d(i,\ j) = C_{\text{Alg}}$ holds for the deletion.

The dynamics of the edge weights do not interfere with the proof from [LRWY99] since in every step Alg′ transfers data only over edges which are also used by Alg. □

We can analyse the performance of RandomizedTree now.

Theorem 3.12. *Let* σ *be an arbitrary input sequence,* $s \in \{1,\dots,n\}$ *the initial position of the file and* Adv *be an adaptive-online adversary for the file allocation problem in an arbitrary dynamic tree* $(\{1,\dots,n\},\ E)$*. Then*

$$\mathrm{E}\left[C_{\text{RT}}(\sigma)\right] \le \sum_{t=1}^{|\sigma|} \gamma_{\text{RT}}\left(\bar{\delta}_t\right) \cdot \mathrm{E}\left[C_{\text{Adv}}(\sigma_t)\right]$$

holds, where $\bar{\delta}_t := \frac{1}{|E|} \sum_{e \in E} \left|d_t(e) - d_{t-1}(e)\right|$ *is the average change of edge weights in step* t *and* $\gamma_{\text{RT}}(\bar{\delta}_t) := \max\left(3,\ 1 + \frac{3D+3}{p}\left(1 - \frac{1}{n}\right)\bar{\delta}_t\right)$.

Corollary 3.13. RandomizedTree *is strictly* $\max\left(3,\ 1 + \frac{3D+3}{p}\left(1 - \frac{1}{n}\right)\delta\right)$*-competitive against* δ*-restricted adversaries.*

Proof of Theorem 3.12. We will regularly use in the following that the replica set R_{RT} of RANDOMIZEDTREE constitutes always a connected subtree, i.e. $\mathcal{ST}(R_{\mathrm{RT}}) = R_{\mathrm{RT}}$. Because of Lemma 3.11 we can assume that the same holds for the adversary ADV, i.e. $\mathcal{ST}(R_{\mathrm{ADV}}) = R_{\mathrm{ADV}}$.

We show the theorem using this potential function:

$$\Phi_t := 3(D+1)st_t\left(R^t_{\mathrm{RT}} \cup R^t_{\mathrm{ADV}}\right) - (2D+1)st_t\left(R^t_{\mathrm{RT}}\right) - 3st_t\left(R^t_{\mathrm{ADV}}\right). \tag{3.15}$$

Φ_0 is 0 since both algorithms start with a single copy on s. Furthermore, we have for all possible copy sets R_{RT}, R_{ADV} and edge-weights:

$$\begin{aligned}
\Phi_t &= 3(D+1)st_t(R_{\mathrm{RT}} \cup R_{\mathrm{ADV}}) - (2D+1)st_t(R_{\mathrm{RT}}) - 3st_t(R_{\mathrm{ADV}}) \\
&\geq 3(D+1)st_t(R_{\mathrm{RT}} \cup R_{\mathrm{ADV}}) - (2D+1)st_t(R_{\mathrm{RT}}) - (D+2)st_t(R_{\mathrm{ADV}}) \\
&\overset{(3.14)}{=} 3(D+1)\left(st_t(R_{\mathrm{RT}}) + st_t(R_{\mathrm{ADV}}) - st_t(R_{\mathrm{RT}} \cap R_{\mathrm{ADV}})\right. \\
&\quad \left. + d_t(R_{\mathrm{RT}},\, R_{\mathrm{ADV}})\right) - (2D+1)st_t(R_{\mathrm{RT}}) - (D+2)st_t(R_{\mathrm{ADV}}) \\
&\geq (D+2)st_t(R_{\mathrm{RT}}) + (2D+1)st_t(R_{\mathrm{ADV}}) - 3(D+1)st_t(R_{\mathrm{RT}} \cap R_{\mathrm{ADV}}) \\
&\geq 3(D+1)st_t(R_{\mathrm{RT}} \cap R_{\mathrm{ADV}}) - 3(D+1)st_t(R_{\mathrm{RT}} \cap R_{\mathrm{ADV}}) \\
&= 0.
\end{aligned}$$

We divide every step into two parts. In the first part both RANDOMIZEDTREE and ADV pay p for every node and the weights of the edges are updated. In the second part the file access is served and the algorithms may reallocate copies. We denote by $\Phi_{t-\frac{1}{2}}$ the potential after the first part of step t, i.e.

$$\Phi_{t-\frac{1}{2}} = 3(D+1)st_t\left(R^{t-1}_{\mathrm{RT}} \cup R^{t-1}_{\mathrm{ADV}}\right) - (2D+1)st_t\left(R^{t-1}_{\mathrm{RT}}\right) - 3st_t\left(R^{t-1}_{\mathrm{ADV}}\right).$$

Below we show that for the first part of any step t holds:

$$np + \mathrm{E}\left[\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right] \leq \left(1 + \frac{3D+3}{p}\left(1 - \frac{1}{n}\right)\bar{\delta}_t\right)np. \tag{3.16}$$

And for the second part, we show:

$$\mathrm{E}\left[\tilde{C}_{\mathrm{RT}}(a_t) + \Phi_t - \Phi_{t-\frac{1}{2}}\right] \leq 3 \cdot \mathrm{E}\left[\tilde{C}_{\mathrm{ADV}}(\sigma_t)\right]. \tag{3.17}$$

Notice that (3.8) especially says that RANDOMIZEDTREE is 3-competitive against adaptive-online adversaries in a static tree network.

Altogether, we then have

$$
\begin{aligned}
\mathrm{E}\left[C_{\mathrm{RT}}(\sigma)\right] &\leq \mathrm{E}\left[C_{\mathrm{RT}}(\sigma) + \Phi_{|\sigma|} - \Phi_0\right] \\
&= \sum_{t=1}^{|\sigma|} \left(np + \mathrm{E}\left[\tilde{C}_{\mathrm{RT}}(\sigma_t)\right]\right) + \mathrm{E}\left[\Phi_{|\sigma|} - \Phi_0\right] \\
&= \sum_{t=1}^{|\sigma|} \left(np + \mathrm{E}\left[\left(\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right) + \left(\tilde{C}_{\mathrm{RT}}(\sigma_t) + \Phi_t - \Phi_{t-\frac{1}{2}}\right)\right]\right) \\
&= \sum_{t=1}^{|\sigma|} \left(\left(np + \mathrm{E}\left[\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right]\right) + \mathrm{E}\left[\tilde{C}_{\mathrm{RT}}(\sigma_t) + \Phi_t - \Phi_{t-\frac{1}{2}}\right]\right) \\
&\overset{(*)}{\leq} \sum_{t=1}^{|\sigma|} \left(\left(1 + \frac{3D+3}{p}\left(1 - \frac{1}{n}\right)\bar{\delta}_t\right) np + 3\mathrm{E}\left[\tilde{C}_{\mathrm{Adv}}(\sigma_t)\right]\right) \\
&\leq \sum_{t=1}^{|\sigma|} \max\left(3,\ 1 + \frac{3D+3}{p}\left(1 - \frac{1}{n}\right)\bar{\delta}_t\right) \mathrm{E}\left[C_{\mathrm{Adv}}(\sigma_t)\right],
\end{aligned}
$$

where $(*)$ follows from (3.16) and (3.17).

For the proof of (3.16), we observe that the costs of RandomizedTree and Adv in the first phase of a step are always np. Furthermore, the value of Φ changes only due to the changes of the weights of the edges since no actions are performed. We first transform (3.16) as follows:

$$
\begin{aligned}
& np + \mathrm{E}\left[\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right] \leq \left(1 + \frac{3D+3}{p}\left(1 - \frac{1}{n}\right)\bar{\delta}_t\right) np \\
\iff \quad & \mathrm{E}\left[\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right] \leq \frac{3D+3}{p}\left(\frac{n-1}{n}\frac{1}{|E|}\sum_{e\in E}\left|d_t(e) - d_{t-1}(e)\right|\right) np \\
\iff \quad & \mathrm{E}\left[\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right] \leq (3D+3)\sum_{e\in E}\left|d_e^{t-1} - d_e^{t-1}\right|.
\end{aligned}
$$

This holds since

$$
\begin{aligned}
&\mathrm{E}\left[\Phi_{t-\frac{1}{2}} - \Phi_{t-1}\right] \\
&= \mathrm{E}\Big[\Big(3(D+1)st_t\Big(R_{\mathrm{RT}}^{t-1} \cup R_{\mathrm{Adv}}^{t-1}\Big) - (2D+1)st_t\Big(R_{\mathrm{RT}}^{t-1}\Big) - 3st_t\Big(R_{\mathrm{Adv}}^{t-1}\Big)\Big) \\
&\quad -\Big(3(D+1)st_{t-1}\Big(R_{\mathrm{RT}}^{t-1} \cup R_{\mathrm{Adv}}^{t-1}\Big) - (2D+1)st_{t-1}\Big(R_{\mathrm{RT}}^{t-1}\Big) - 3st_{t-1}\Big(R_{\mathrm{Adv}}^{t-1}\Big)\Big)\Big] \\
&= \mathrm{E}\left[3(D+1)\sum_{e\in\mathcal{ST}\left(R_{\mathrm{RT}}^{t-1}\cup R_{\mathrm{Adv}}^{t-1}\right)}\Big(d_t(e) - d_{t-1}(e)\Big) - (2D+1)\sum_{e\in R_{\mathrm{RT}}^{t-1}}\Big(d_t(e) - d_{t-1}(e)\Big) - 3\sum_{e\in R_{\mathrm{Adv}}^{t-1}}\Big(d_t(e) - d_{t-1}(e)\Big)\right] \\
&= \mathrm{E}\Bigg[[3(D+1) - (2D+1)]\sum_{e\in R_{\mathrm{RT}}^{t-1}\setminus R_{\mathrm{Adv}}^{t-1}}\Big(d_t(e) - d_{t-1}(e)\Big) + [3(D+1) - 3]\sum_{e\in R_{\mathrm{Adv}}^{t-1}\setminus R_{\mathrm{RT}}^{t-1}}\Big(d_t(e) - d_{t-1}(e)\Big) \\
&\qquad + [3(D+1) - (2D+1) - 3]\sum_{e\in R_{\mathrm{RT}}^{t-1}\cap R_{\mathrm{Adv}}^{t-1}}\Big(d_t(e) - d_{t-1}(e)\Big) + [3(D+1)]\sum_{e\in\mathcal{ST}\left(R_{\mathrm{RT}}^{t-1}\cup R_{\mathrm{Adv}}^{t-1}\right)\setminus\left(R_{\mathrm{RT}}^{t-1}\cup R_{\mathrm{Adv}}^{t-1}\right)}\Big(d_t(e) - d_{t-1}(e)\Big)\Bigg] \\
&\le \mathrm{E}\Bigg[(D+2)\sum_{e\in R_{\mathrm{RT}}^{t-1}\setminus R_{\mathrm{Adv}}^{t-1}}\Big|d_t(e) - d_{t-1}(e)\Big| + 3D\sum_{e\in R_{\mathrm{Adv}}^{t-1}\setminus R_{\mathrm{RT}}^{t-1}}\Big|d_t(e) - d_{t-1}(e)\Big| \\
&\qquad +(D-1)\sum_{e\in R_{\mathrm{RT}}^{t-1}\cap R_{\mathrm{Adv}}^{t-1}}\Big|d_t(e) - d_{t-1}(e)\Big| + 3(D+1)\sum_{e\in\mathcal{ST}\left(R_{\mathrm{RT}}^{t-1}\cup R_{\mathrm{Adv}}^{t-1}\right)\setminus\left(R_{\mathrm{Adv}}^{t-1}\cup R_{\mathrm{RT}}^{t-1}\right)}\Big|d_t(e) - d_{t-1}(e)\Big|\Bigg] \\
&\le \mathrm{E}\left[3(D+1)\sum_{e\in\mathcal{ST}\left(R_{\mathrm{RT}}^{t-1}\cup R_{\mathrm{Adv}}^{t-1}\right)}\Big|d_t(e) - d_{t-1}(e)\Big|\right] \\
&\le 3(D+1)\sum_{e\in E}\Big|d_t(e) - d_{t-1}(e)\Big|.
\end{aligned}
$$

For the proof of (3.17), we follow the proof of Theorem 4.4.2 of [Bar94]. We show (3.17) in two steps: First the request is served and RANDOMIZEDTREE performs actions, then the adversary can perform actions.

Request and RANDOMIZEDTREE's actions.
We will analyse the following components of the potential:

$$
\begin{aligned}
\Theta_t &:= st_t\Big(R_{\mathrm{RT}}^{t-1}\Big), \\
\Psi_t &:= st_t\Big(R_{\mathrm{RT}}^{t-1} \cup R_{\mathrm{Adv}}^{t-1}\Big) - st_t\Big(R_{\mathrm{RT}}^{t-1}\Big).
\end{aligned}
$$

Since R_{Adv} does not change in this step of the analysis, this leads to

$$\Delta\Phi = (D+2)\Delta\Theta + 3(D+1)\Delta\Psi.$$

No request.
Since RandomizedTree does nothing, there are neither any costs nor does the potential change. Thus,

$$\mathrm{E}\left[\tilde{C}_{\mathrm{RT}} + \Delta\Phi\right] = 0 = 3\tilde{C}_{\mathrm{Adv}}.$$

Read request.
Let i be the source of the read request. RandomizedTree first sends the requested data from R_{RT} to i and then creates copies on the way from R_{RT} to i with probability $\frac{1}{D+1}$. Thus, the expected costs of RandomizedTree are:

$$\mathrm{E}\left[\tilde{C}_{\mathrm{RT}}\right] = d_t\left(i,\ R_{\mathrm{RT}}^{t-1}\right) + \frac{1}{D+1} D d_t\left(i,\ R_{\mathrm{RT}}^{t-1}\right) = \frac{2D+1}{D+1} d_t\left(i,\ R_{\mathrm{RT}}^{t-1}\right).$$

Θ increases by $d_t\left(i,\ R_{\mathrm{RT}}^{t-1}\right)$ if RandomizedTree replicates to i. Otherwise, it does not change. Thus,

$$\begin{aligned}\mathrm{E}\left[\Delta\Theta\right] &= \frac{1}{D+1}\left(st_t\left(R_{\mathrm{RT}}^{t-1} \cup \{i\}\right) - st_t\left(R_{\mathrm{RT}}^{t-1}\right)\right) + \left(1 - \frac{1}{D+1}\right)\left(st_t\left(R_{\mathrm{RT}}^{t-1}\right) - st_t\left(R_{\mathrm{RT}}^{t-1}\right)\right) \\ &\overset{(3.13)}{=} \frac{1}{D+1} d_t\left(i,\ R_{\mathrm{RT}}^{t-1}\right).\end{aligned}$$

Ψ increases also only if RandomizedTree replicates to i. Thus,

$$\begin{aligned}\mathrm{E}\left[\Delta\Psi\right] &= \frac{1}{D+1}\left[\left(st_t\left(R_{\mathrm{RT}}^{t-1} \cup R_{\mathrm{Adv}}^{t-1} \cup \{i\}\right) - st_t\left(R_{\mathrm{RT}}^{t-1} \cup \{i\}\right)\right) - \left(st_t\left(R_{\mathrm{RT}}^{t-1} \cup R_{\mathrm{Adv}}^{t-1}\right) - st_t\left(R_{\mathrm{RT}}^{t-1}\right)\right)\right] \\ &= \frac{1}{D+1}\left[\left(st_t\left(R_{\mathrm{RT}}^{t-1} \cup R_{\mathrm{Adv}}^{t-1} \cup \{i\}\right) - st_t\left(R_{\mathrm{RT}}^{t-1} \cup R_{\mathrm{Adv}}^{t-1}\right)\right) - \left(st_t\left(R_{\mathrm{RT}}^{t-1} \cup \{i\}\right) - st_t\left(R_{\mathrm{RT}}^{t-1}\right)\right)\right] \\ &\overset{(3.13)}{=} \frac{1}{D+1}\left[d_t\left(i,\ \mathcal{ST}\left(R_{\mathrm{RT}}^{t-1} \cup R_{\mathrm{Adv}}^{t-1}\right)\right) - d_t\left(i,\ R_{\mathrm{RT}}^{t-1}\right)\right] \\ &\leq \frac{1}{D+1}\left[d_t\left(i,\ R_{\mathrm{Adv}}^{t-1}\right) - d_t\left(i,\ R_{\mathrm{RT}}^{t-1}\right)\right].\end{aligned}$$

Together, this results in

$$\begin{aligned}\mathrm{E}\left[\tilde{C}_{\mathrm{RT}} + \Delta\Phi\right] &= \mathrm{E}\left[\tilde{C}_{\mathrm{RT}}\right] + (D+2)\mathrm{E}\left[\Delta\Theta\right] + 3(D+1)\mathrm{E}\left[\Delta\Psi\right] \\ &\leq \frac{2D+1}{D+1} d_t\left(i,\ R_{\mathrm{RT}}^{t-1}\right) + \frac{D+2}{D+1} d_t\left(i,\ R_{\mathrm{RT}}^{t-1}\right) \\ &\quad + 3\frac{D+1}{D+1}\left(d_t\left(i,\ R_{\mathrm{Adv}}^{t-1}\right) - d_t\left(i,\ R_{\mathrm{RT}}^{t-1}\right)\right) \\ &= 3 d_t\left(i,\ R_{\mathrm{Adv}}^{t-1}\right) \\ &= 3\tilde{C}_{\mathrm{Adv}}.\end{aligned}$$

Write request.
Let i be the source of the write request and j be the nearest node to i in R_{RT}^{t-1}. On a write request, an update is first send from i to j and then from j to all the nodes in R_{RT}^{t-1}. This costs $st_t(R_{\text{RT}} \cup \{i\}) = d_t\left(i,\ R_{\text{RT}}^{t-1}\right) + st_t\left(R_{\text{RT}}^{t-1}\right)$. Then RANDOMIZEDTREE deletes all the copies but the one on j with probability $\frac{1}{D+1}$. This is done by sending delete messages starting from j over each edge in $\mathcal{ST}\left(R_{\text{RT}}^{t-1}\right)$ and therefore incurs a cost of $st_t\left(R_{\text{RT}}^{t-1}\right)$. Furthermore, the copy from j is migrated to i with probability $\frac{1}{2}$. This is done by a replication from j to i and a subsequent deletion of the copy on j initiated by i. Thus, the migration costs $(D+1)d_t\left(i,\ R_{\text{RT}}^{t-1}\right)$. Altogether, this results in

$$\begin{aligned}\mathrm{E}\left[C_{\text{RT}}\right] &= d_t\left(i,\ R_{\text{RT}}^{t-1}\right) + st\left(R_{\text{RT}}^{t-1}\right) + \frac{1}{D+1}\left(st_t\left(R_{\text{RT}}^{t-1}\right) + \frac{1}{2}(D+1)d_t\left(i,\ R_{\text{RT}}^{t-1}\right)\right)\\ &= \left(1 + \frac{1}{D+1}\right)st_t\left(R_{\text{RT}}^{t-1}\right) + \frac{3}{2}d_t\left(i,\ R_{\text{RT}}^{t-1}\right).\end{aligned}$$

The value of Θ changes to 0 if RANDOMIZEDTREE deletes all the copies but one (either j or i remains). Otherwise, it does not change. Thus,

$$\begin{aligned}\mathrm{E}\left[\Delta\Theta\right] &= \frac{1}{D+1}\left(0 - st_t\left(R_{\text{RT}}^{t-1}\right)\right) + \left(1 - \frac{1}{D+1}\right)\left(st_t\left(R_{\text{RT}}^{t-1}\right) - st_t\left(R_{\text{RT}}^{t-1}\right)\right)\\ &= -\frac{1}{D+1}st_t\left(R_{\text{RT}}^{t-1}\right).\end{aligned}$$

We divide the change of Ψ in two parts. $\Delta\Psi^1$ is the change of Ψ for the case that RANDOMIZEDTREE deletes all the copies but the one at j and does not migrate to i. And $\Delta\Psi^2$ is the change of Ψ for the case that RANDOMIZEDTREE deletes all the copies and migrates to i. Then we have

$$\mathrm{E}\left[\Delta\Psi\right] = \left(1 - \frac{1}{D+1}\right)\cdot 0 + \frac{1}{2(D+1)}\Delta\Psi^1 + \frac{1}{2(D+1)}\Delta\Psi^2.$$

For $\Delta\Psi^1$, we get

$$\begin{aligned}\Delta\Psi^1 &= \left(st_t\left(R_{\text{Adv}}^{t-1} \cup \{j\}\right) - st_t(\{j\})\right) - \left(st_t\left(R_{\text{RT}}^{t-1} \cup R_{\text{Adv}}^{t-1}\right) - st_t\left(R_{\text{RT}}^{t-1}\right)\right)\\ &\overset{(3.13)}{=} st_t\left(R_{\text{Adv}}^{t-1}\right) + d_t\left(j,\ R_{\text{Adv}}^{t-1}\right) + st_t\left(R_{\text{RT}}^{t-1}\right) - st_t\left(R_{\text{RT}}^{t-1} \cup R_{\text{Adv}}^{t-1}\right)\\ &\overset{(3.14)}{=} st_t\left(R_{\text{Adv}}^{t-1}\right) + d_t\left(j,\ R_{\text{Adv}}^{t-1}\right) + st_t\left(R_{\text{RT}}^{t-1}\right)\\ &\quad - \left(st_t\left(R_{\text{RT}}^{t-1}\right) + st_t\left(R_{\text{Adv}}^{t-1}\right) - st_t\left(R_{\text{RT}}^{t-1} \cap R_{\text{Adv}}^{t-1}\right) + d_t\left(R_{\text{RT}}^{t-1},\ R_{\text{Adv}}^{t-1}\right)\right)\\ &= st_t\left(R_{\text{RT}}^{t-1} \cap R_{\text{Adv}}^{t-1}\right) + d_t\left(j,\ R_{\text{Adv}}^{t-1}\right) - d_t\left(R_{\text{RT}}^{t-1},\ R_{\text{Adv}}^{t-1}\right)\\ &\leq st_t\left(R_{\text{Adv}}^{t-1}\right) + d_t\left(j,\ R_{\text{Adv}}^{t-1}\right) - d_t\left(R_{\text{RT}}^{t-1},\ R_{\text{Adv}}^{t-1}\right).\end{aligned}$$

For the analysis of $\Delta\Psi^2$ we assume that RandomizedTree first replicates from j to i and thereafter deletes all the copies but the one on i. Then we have

$$\begin{aligned}
\Delta\Psi^2 &= \Big[st_t\Big(R^{t-1}_{\text{Adv}} \cup \{i\}\Big) - st_t(\{i\})\Big] - \Big[st_t\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}}\Big) - st_t\Big(R^{t-1}_{\text{RT}}\Big)\Big] \\
&= \Big[st_t\Big(R^{t-1}_{\text{Adv}} \cup \{i\}\Big) - st_t(\{i\})\Big] - \Big[st_t\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}} \cup \{i\}\Big) - st_t\Big(R^{t-1}_{\text{RT}} \cup \{i\}\Big)\Big] \\
&\quad + \Big[st_t\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}} \cup \{i\}\Big) - st_t\Big(R^{t-1}_{\text{RT}} \cup \{i\}\Big)\Big] - \Big[st_t\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}}\Big) - st_t\Big(R^{t-1}_{\text{RT}}\Big)\Big] \\
&\leq st_t\Big(R^{t-1}_{\text{Adv}} \cup \{i\}\Big) + \Big[st_t\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}} \cup \{i\}\Big) - st_t\Big(R^{t-1}_{\text{RT}} \cup \{i\}\Big)\Big] - \Big[st_t\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}}\Big) - st_t\Big(R^{t-1}_{\text{RT}}\Big)\Big] \\
&= st_t\Big(R^{t-1}_{\text{Adv}} \cup \{i\}\Big) + \Big[st_t\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}} \cup \{i\}\Big) - st_t\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}}\Big)\Big] - \Big[st_t\Big(R^{t-1}_{\text{RT}} \cup \{i\}\Big) - st_t\Big(R^{t-1}_{\text{RT}}\Big)\Big] \\
&\overset{(3.13)}{=} st_t\Big(R^{t-1}_{\text{Adv}} \cup \{i\}\Big) + d_t\Big(i,\ \mathcal{ST}\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}}\Big)\Big) - d_t\Big(i,\ R^{t-1}_{\text{RT}}\Big).
\end{aligned}$$

Before we can bound the expected change of Ψ, we need to show

$$d_t\Big(j,\ R^{t-1}_{\text{Adv}}\Big) - d_t\Big(R^{t-1}_{\text{RT}},\ R^{t-1}_{\text{Adv}}\Big) + d_t\Big(i,\ \mathcal{ST}\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}}\Big)\Big) \leq d_t\Big(i,\ R^{t-1}_{\text{Adv}}\Big). \tag{3.18}$$

We show this by differentiating two cases. If $d_t\Big(R^{t-1}_{\text{RT}},\ R^{t-1}_{\text{Adv}}\Big) = d_t\Big(j,\ R^{t-1}_{\text{Adv}}\Big)$ then, we trivially get

$$\begin{aligned}
&d_t\Big(j,\ R^{t-1}_{\text{Adv}}\Big) - d_t\Big(R^{t-1}_{\text{RT}},\ R^{t-1}_{\text{Adv}}\Big) + d_t\Big(i,\ \mathcal{ST}\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}}\Big)\Big) \\
&= d_t\Big(i,\ \mathcal{ST}\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}}\Big)\Big) \\
&\leq d_t\Big(i,\ R^{t-1}_{\text{Adv}}\Big).
\end{aligned}$$

Otherwise, $d_t\Big(R^{t-1}_{\text{RT}},\ R^{t-1}_{\text{Adv}}\Big) < d_t\Big(j,\ R^{t-1}_{\text{Adv}}\Big)$ holds, i.e. there is a node in R^{t-1}_{RT}, which is closer to R^{t-1}_{Adv} than j is. Imagine that node i is the root of the tree $(\{1,\ldots,n\}, E)$. Then all the nodes in R^{t-1}_{RT} are below j since j is defined as the node nearest to i in R^{t-1}_{RT}. And since the distance between R^{t-1}_{RT} and R^{t-1}_{Adv} is not determined by j, the nodes of R^{t-1}_{Adv} also have to be below j. This implies that $d_t\Big(i,\ \mathcal{ST}\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}}\Big)\Big) = d_t(i,\ j)$ and $d\Big(i,\ R^{t-1}_{\text{Adv}}\Big) = d(i,\ j) + d\Big(j,\ R^{t-1}_{\text{Adv}}\Big)$. Thus,

$$\begin{aligned}
&d_t\Big(j,\ R^{t-1}_{\text{Adv}}\Big) - d_t\Big(R^{t-1}_{\text{RT}},\ R^{t-1}_{\text{Adv}}\Big) + d_t\Big(i,\ \mathcal{ST}\Big(R^{t-1}_{\text{RT}} \cup R^{t-1}_{\text{Adv}}\Big)\Big) \\
&= d_t(i,\ j) + d_t\Big(j,\ R^{t-1}_{\text{Adv}}\Big) - d_t\Big(R^{t-1}_{\text{RT}},\ R^{t-1}_{\text{Adv}}\Big) \\
&= d_t\Big(i,\ R^{t-1}_{\text{Adv}}\Big) - d_t\Big(R^{t-1}_{\text{RT}},\ R^{t-1}_{\text{Adv}}\Big) \\
&\leq d_t\Big(i,\ R^{t-1}_{\text{Adv}}\Big).
\end{aligned}$$

The expected change of Ψ can now be bounded by

$$\begin{aligned}
\mathrm{E}\left[\Delta\Psi\right] &= \left(1 - \frac{1}{D+1}\right)\cdot 0 + \frac{1}{2(D+1)}\cdot\Delta\Psi^1 + \frac{1}{2(D+1)}\cdot\Delta\Psi^2 \\
&\leq \frac{1}{2(D+1)}\Big[st_t\Big(R^{t-1}_{\text{ADV}}\Big) + d_t\Big(j,\ R^{t-1}_{\text{ADV}}\Big) - d_t\Big(R^{t-1}_{\text{RT}},\ R^{t-1}_{\text{ADV}}\Big) \\
&\qquad + st_t\Big(R^{t-1}_{\text{ADV}}\cup\{i\}\Big) + d_t\Big(i,\ \mathcal{ST}\Big(R^{t-1}_{\text{RT}}\cup R^{t-1}_{\text{ADV}}\Big)\Big) - d\Big(i,\ R^{t-1}_{\text{RT}}\Big)\Big] \\
&= \frac{1}{2(D+1)}\Big[st_t\Big(R^{t-1}_{\text{ADV}}\Big) + st_t\Big(R^{t-1}_{\text{ADV}}\cup\{i\}\Big) - d\Big(i,\ R^{t-1}_{\text{RT}}\Big) \\
&\qquad + d_t\Big(j,\ R^{t-1}_{\text{ADV}}\Big) - d_t\Big(R^{t-1}_{\text{RT}},\ R^{t-1}_{\text{ADV}}\Big) + d_t\Big(i,\ \mathcal{ST}\Big(R^{t-1}_{\text{RT}}\cup R^{t-1}_{\text{ADV}}\Big)\Big)\Big] \\
&\overset{(3.18)}{\leq} \frac{1}{2(D+1)}\left[st_t\Big(R^{t-1}_{\text{ADV}}\Big) + st_t\Big(R^{t-1}_{\text{ADV}}\cup\{i\}\Big) - d_t\Big(i,\ R^{t-1}_{\text{RT}}\Big) + d_t\Big(i,\ R^{t-1}_{\text{ADV}}\Big)\right] \\
&\overset{(3.13)}{=} \frac{1}{2(D+1)}\left[2st_t\Big(R^{t-1}_{\text{ADV}}\cup\{i\}\Big) - d_t\Big(i,\ R^{t-1}_{\text{RT}}\Big)\right].
\end{aligned}$$

Altogether, we have:

$$\begin{aligned}
\mathrm{E}\left[C_{\text{RT}} + \Delta\Phi\right] &= \mathrm{E}\left[C_{\text{RT}}\right] + (D+2)\mathrm{E}\left[\Delta\Theta\right] + 3(D+1)\mathrm{E}\left[\Delta\Psi\right] \\
&\leq \frac{D+2}{D+1}st_t\Big(R^{t-1}_{\text{RT}}\Big) + \frac{3}{2}d_t\Big(i,\ R^{t-1}_{\text{RT}}\Big) - \frac{D+2}{D+1}st_t\Big(R^{t-1}_{\text{RT}}\Big) \\
&\quad + \frac{3}{2}\left(2st_t\Big(R^{t-1}_{\text{ADV}}\cup\{i\}\Big) - d_t\Big(i,\ R^{t-1}_{\text{RT}}\Big)\right) \\
&= 3st_t\Big(R^{t-1}_{\text{ADV}}\cup\{i\}\Big) \\
&= 3C_{\text{ADV}}.
\end{aligned}$$

ADV's actions.
Without loss of generality we can treat each action performed by ADV separately.

ADV replicates.
Let $i \in \{1,\ldots,n\}\setminus R_{\text{ADV}}$ be the destination of the new copy. Since RANDOMIZEDTREE has no costs and its replica set does not change, we have

$$\begin{aligned}
\tilde{C}_{\text{RT}} + \Delta\Phi &= \Big[3(D+1)st_t(R_{\text{RT}}\cup(R_{\text{ADV}}\cup\{i\})) - (2D+1)st_t(R_{\text{RT}}) - 3st_t(R_{\text{ADV}}\cup\{i\})\Big] \\
&\quad - \Big[3(D+1)st_t(R_{\text{RT}}\cup R_{\text{ADV}}) - (2D+1)st_t(R_{\text{RT}}) - 3st_t(R_{\text{ADV}})\Big] \\
&= 3(D+1)\Big[st_t((R_{\text{RT}}\cup R_{\text{ADV}})\cup\{i\}) - st_t(R_{\text{RT}}\cup R_{\text{ADV}})\Big] \\
&\quad - 3\Big[st_t(R_{\text{ADV}}\cup\{i\}) - st_t(R_{\text{ADV}})\Big] \\
&\overset{(3.13)}{=} 3(D+1)d_t(i,\ \mathcal{ST}(R_{\text{RT}}\cup R_{\text{ADV}})) - 3d_t(i,\ R_{\text{ADV}}) \\
&\leq 3(D+1)d_t(i,\ R_{\text{ADV}}) - 3d_t(i,\ R_{\text{ADV}}) \\
&= 3Dd_t(i,\ R_{\text{ADV}}) \\
&= 3\tilde{C}_{\text{ADV}}.
\end{aligned}$$

Adv deletes.
Let $i \in R_{\text{Adv}}$ be the node which holds the copy to be deleted. Since RandomizedTree has no costs and its replica set does not change, we have

$$\begin{aligned}
\tilde{C}_{\text{RT}} + \Delta\Phi &= \Big[3(D+1)st_t(R_{\text{RT}} \cup (R_{\text{Adv}} \setminus \{i\})) - (2D+1)st_t(R_{\text{RT}}) - 3st_t(R_{\text{Adv}} \setminus \{i\})\Big] \\
&\quad - \Big[3(D+1)st_t(R_{\text{RT}} \cup R_{\text{Adv}}) - (2D+1)st_t(R_{\text{RT}}) - 3st_t(R_{\text{Adv}})\Big] \\
&= 3(D+1)\Big[st_t(R_{\text{RT}} \cup (R_{\text{Adv}} \setminus \{i\})) - st_t((R_{\text{RT}} \cup (R_{\text{Adv}} \setminus \{i\})) \cup \{i\})\Big] \\
&\quad - 3\Big[st_t(R_{\text{Adv}} \setminus \{i\}) - st_t((R_{\text{Adv}} \setminus i) \cup \{i\})\Big] \\
&\overset{(3.13)}{=} -3(D+1)d_t(i,\ \mathcal{ST}(R_{\text{RT}} \cup (R_{\text{Adv}} \setminus \{i\}))) + 3d_t(i,\ \mathcal{ST}(R_{\text{Adv}} \setminus \{i\})) \\
&\leq 3d_t(i,\ \mathcal{ST}(R_{\text{Adv}} \setminus \{i\})) \\
&\leq 3d_t(i,\ R_{\text{Adv}} \setminus \{i\}) \\
&= 3\tilde{C}_{\text{Adv}}.
\end{aligned}$$

This concludes the proof of (3.17) and therewith of the theorem. □

3.3 Conclusion and open problems

In this chapter we have considered a model for data management in dynamic networks which comprises stand-by costs of the participating nodes. We limited our examinations to networks with star respectively tree topologies, because such networks have unique and easy to find Steiner trees. The drawback of these topologies is that they do not allow to exploit locality in the MANET. In the star topology a participating node can only access the copy on the server and the possibly available own copy, but not the copy of a nearby participating node. Thus, creating a copy can be beneficial only for the participating node which receives the copy but not for its neighborhood. In the tree topology two participating nodes, which are close-by in the MANET, can have several hops between them in the tree. And since the communication only happens along the tree links, this can result in costly data transfers even though the actually communicating nodes are side by side in the MANET.

In order to exploit locality of requests in the MANET optimally, each participating node has to be able to communicate with any other participating node directly. This results in the difficulty that we can no longer reduce the MANET to an overlay network of the participating nodes with a fixed and well structured topology. This is because the topology of the overlay network, which is defined by the used routing protocol, can be arbitrary and even changes with the movement of the nodes. Moreover, a file allocation system in a network with cycles requires a distributed mechanism to track the available copies. Such a tracking protocol

has to be able to name a path to a nearby copy whenever a participating node reads the file. Furthermore, it has to find Steiner trees of all present copies and an arbitrary initiator of a write request to update the file. The tracking protocol given by Bartal et al. [Bar94] provides these services for arbitrary static networks. But it is not possible to generalize this protocol to dynamic networks since it relies on a precalculated data structure (called *regional matching* [AP95]) which comprises information about distances. Furthermore, their solution for the *cover problem*, which is used by the tracking protocol, depends on static edge weights and a fixed topology. Thus, it is still an open and surely challenging problem to find a distributed (or even local) tracking protocol for dynamic networks.

It is especially doubtful, whether such a tracking protocol can be analyzed using competitive analysis. One reason is that it is not clear how to define the optimal costs of the tracking problem. One could define the sum of the shortest distances respectively Steiner trees at the points of time where tracking queries are issued as the optimal costs. But then no online algorithm could be competitive, because obviously any online algorithm has to update its configuration regularly when the network considerably changes even if no actual tracking request is issued. Another possible definition for the optimal costs is that it is the lowest costs any algorithm with global (or even local) knowledge, which is able to return at any point of time and for any requesting node the optimal path respectively Steiner tree, can have. But this seems to be a too tough definition since optimal Steiner trees are quiet unstable under movement and therefore a lot of updates and communication would be necessary even for such an optimal algorithm. And in fact an approximation of the shortest path and Steiner trees would be sufficient for an actual implementation of a file allocation system.

Another possible model is to use a complete network as the overlay network of participating nodes where edge weights are defined by the shortest routing paths through the MANET, i.e. we are looking at the distance metric of the MANET[7]. A shortcoming of this model is that the weight of Steiner trees are inaccurate. This is because Steiner trees in the MANET could use non-participating nodes as Steiner nodes. But as shown in Section 6.1 of [PS02] the weight of a minimum Steiner tree in the metric constitutes a 2-approximation of the weight of a minimum Steiner tree in the MANET. The more severe problems of this model are that a participating node of the MANET would have to be aware of its energy-distance to all the other participating nodes and of the current locations of the copies.

Star topology

We have shown that the deterministic algorithms FOLLOW and COUNT are $O\left(\frac{D}{p}\delta\right)$-competitive against δ-restricted adversaries and that this is up to a constant factor

[7] This is actually only true if the routing protocol uses optimal routing paths.

optimal for demand-driven algorithms. Furthermore we have given a lower bound of $\Omega\left(\frac{\frac{D}{p}\delta}{lg(\frac{D}{p}\delta)}\right)$ against δ-restricted oblivious adversaries for general randomized algorithms. The obvious open question is: Does an $o\left(\frac{D}{p}\delta\right)$-competitive non-demand-driven algorithm exist or can a better lower bound be found?

Tree topology

We have shown that the randomized algorithm RandomizedTree is $O\left(\frac{D}{p}\delta\right)$-competitive against δ-restricted adaptive online algorithms. Since the lower bounds for the star network can be applied to tree networks as well, RandomizedTree is an optimal demand-driven online algorithm for file allocation in dynamic tree networks. As for star networks, it is an open problem, whether a non-demand driven algorithm with a competitive ratio of $o\left(\frac{D}{p}\delta\right)$ exists.

As discussed above, the fixed topology of the tree overlay does not allow to exploit locality in the MANET. It would therefore be interesting to find a mechanism to change the topology of the overlay tree in response to the dynamics of the MANET. Such a mechanism should maintain an approximation of a minimum Steiner tree of the participating nodes in the MANET. Whenever the topology of this tree changes, the file allocation system deletes all copies but one and restarts its file allocation algorithm with the new tree. Unfortunately the maintenance of a minimum Steiner tree is inherently communication intensive in a MANET since the nodes have only local information.

Chapter 4

Simulation Based Evaluation of File Allocation with Step Costs

In this chapter we describe an experimental evaluation of the algorithms Follow and Count described in Subsection 3.1.2 in a simulated MANET. This MANET consists of a stationary server and a set of mobile nodes. Some of these mobile nodes are participating nodes. A participating node can only access a copy stored on itself or the primary copy on the server. Thus, the file allocation is performed on an overlay network with a star topology as described in Section 3.1.1.

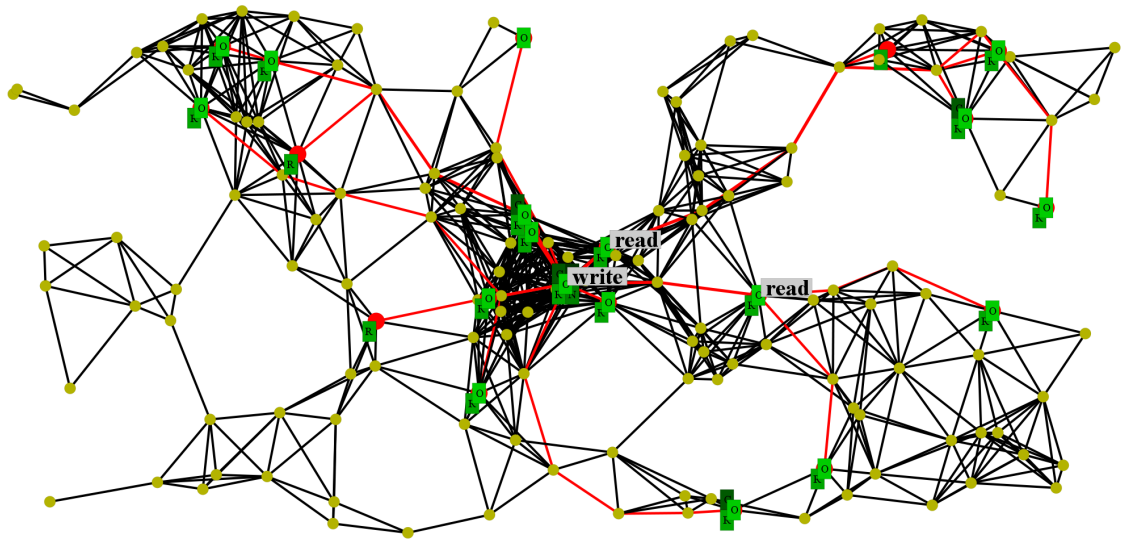


Figure 4.1: A screenshot of a simulation run. The mobile nodes and the server form a unit disk graph. The red nodes are the participating nodes and the red links represent the routing paths to the server located in the center of the MANET. The green squares represent the copies created by the simulated algorithms.

We used the customizable sensor network simulator Shawn [KPB+05, FKFP07] to simulate the behavior of the participating nodes, the server and the other relays. The movement of the nodes was generated by the "mobility model based on social network theory" of [MM07]. The file accesses of the participating nodes and the server were generated stochastically. Every participating node issued requests independent from the other participating nodes. This implies that, unlike our model in Section 3.1.1, more than one request can occur in a single step. In order to ease the evaluation of the experiments, only the server could issue write requests.

Figure 4.1 shows a screenshot of an example simulation run, which was performed in a smaller area and with fewer nodes than the actual experiments.

We will see, that the competitive ratio of Follow and Count for realistic inputs are much lower than the worst-case analysis suggests. While Follow has nevertheless a quiet poor performance, Count turns out to be reasonable good for practical application.

4.1 Evaluated file allocation algorithms

We investigate the performance of the two deterministic online algorithms Follow and Count described in subsections 3.1.2.2 respectively 3.1.2.3. We use the two trivial strategies DoNothing and ReplicateOnFirstRequest as performance references. DoNothing never creates any copies and therefore is equivalent to storing the file only on the server. ReplicateOnFirstRequest replicates to any participating node after its first request and keeps this copy forever. Hence, ReplicateOnFirstRequest is the contrary extreme of DoNothing.

Algorithm 5 Follow on the server

```
1:  R_Follow := {c};
2:  On event e do
3:      if ( e is a read-message from participating node i ) then
4:          Replicate to i;
5:          R_F := R_F ∪ {i};
6:      else if ( e is a write request from the server ) then
7:          for i ∈ R_F \ {c} do
8:              Send delete-message to i;
9:          end for
10:         R_F := {c};
11:     end if
12: end
```

Here, we give the implemented distributed versions of the algorithms. They consist of an algorithm that is executed on the server and one that is executed independently on each of the participating nodes. We thereby omit the handling of the requests, i.e. we only describe what the algorithms do to determine when to perform an action. Furthermore, we did not include code for handling write requests from the participating nodes since such requests do not occur in our experiments. Notice that more than one event can occur and will be handled in a single step, since we allow more than one request in a single step in our simulations.

Algorithm 5 gives the implementation of Follow on the server. The participating nodes are not involved in the decision about performing actions and therefore do not need to execute any extra algorithm.

Algorithm 6 Count on the server

```
1:  c^i := 0 for all i ∈ {1,...,n};
2:  R_Count := {c};
3:  On event e do
4:      if ( e is a read-message from participating node i ) then
5:          c^i := min(c^i + 1, D + 1);
6:          if ( c^i = D + 1 ) then
7:              Replicate to i;
8:              R_C := R_Count ∪ {i};
9:          end if
10:     else if ( e is a write request from the server ) then
11:         for i ∈ {1,...,n} \ R_C do
12:             c^i := max(c^i − 1, 0);
13:         end for
14:     else if ( e is a delete-message from participating node i ) then
15:         R_C := R_C \ {i};
16:         c^i := 0;
17:     end if
18: end
```

The distributed implementation of Count executes Algorithm 6 on the server and Algorithm 7 on every participating node. Initially no participating node holds a copy and therefore all the counters are initialized with 0. Notice that counter c^i is maintained on participating node i when i holds a copy and on the server otherwise. This does not require any additional communication, because on every replicate respectively delete operation the counter always has the value

Algorithm 7 COUNT on participating node i

```
1:  c^i := 0;
2:  holds_copy := false;
3:  On event e do
4:      if ( e is a read request from i ) then
5:          if ( holds_copy ) then
6:              c^i := min(c^i + 1, 0);
7:          end if
8:      else if ( e is a write-message from the server ) then
9:          c^i := max(c^i − 1, 0);
10:         if ( c^i = 0 ) then
11:             Send delete-message to the server.
12:             holds_copy := false;
13:         end if
14:     else if ( e is a replicate-message from the server ) then
15:         c^i := D + 1;
16:         holds_copy := true
17:     end if
18: end
```

$D+1$ respectively 0. Thus, the arrival of a replicate- respectively a delete-message implicitly gives the current value of the counter.

4.2 Simulation environment

We performed the simulations using the customizable sensor network simulator Shawn [KPB+05, FKFP07]. Unlike other network simulators, Shawn does not simulate the lower layers of the OSI Reference Model, but simulates the effects of these layers (e.g. packet loss, corruption and delay) by probabilistic models. We have chosen Shawn, because the simplified simulation of the lower layers allows a reasonable number of nodes and simulation rounds. Furthermore, Shawn allows to calculate the required routing paths using a centralized algorithm.

A set of 1500 mobile nodes are placed in a $2000m \times 2000m$ area which is partitioned into 15×15 squares of equal size. Each mobile node can communicate via a wireless transceiver with a communication radius of $100m$. Besides the mobile nodes, a stationary sever is positioned on the midpoint of the area, which also has a wireless transceiver. The server always holds the primary copy of the file.

The power consumption for sending respectively receiving a single unit of the file is chosen uniformly and independently at random between 0.95 and 1.05 for

each mobile node and direction at the beginning of the simulation. The overall power consumption of a data transfer consists of the sum of the power consumptions incurred for sending respectively receiving the data by any relay on the path between the server and the involved participating node. We thereby do not consider any power consumption of the server. For the routing of a data packages we always use the paths with the lowest overall power consumption, which are calculated by a centralized algorithm in every step. Furthermore, the stand-by power consumption of a participating node p is chosen uniformly and independently at random between 0.095 and 0.105. Unlike the model introduced in Section 3.1 each participating node has its individual stand-by power consumption.

A simulation run is performed in discrete time steps. Each step stands for $1s$. Each simulation run consists of 10.000 steps ($\approx 2.8h$). In each step every mobile node and the server may issue a request. We used different probabilistic input generation mechanisms which are described in the following sections. Unlike the model introduced in Section 3.1.1, only the server issues write request and only the participating nodes issue read requests. We omitted read requests from the server since they can always be answered by the server and therefore never cause any communication in the wireless network. A write request from participating node i does not differ from a write request from the server for all participating nodes but i. If we allowed write requests from the participating nodes, the number of write requests would depend on the number of participating nodes. Furthermore, it would be more difficult to force a fixed ratio of the number of write requests to the number of read requests. Since we primarily want to investigate the dependence of the performance of the described online algorithms on the write/read-ratio, we disallowed write requests from participating nodes.

The simulations are performed in four phases: In the first phase the movement of the 1500 mobile nodes is generated based on the mobility model described in the next section. Then, for each step the tree of minimum energy-distance paths from each node to the server are calculated using Dijkstra's algorithm [Dij59]. These trees are stored in the nodes such that they can perform the routing between the server and the mobile nodes without global knowledge. Nodes which are not connected to the server during any step are marked as such.

In the second phase we first select uniformly at random 50 participating nodes from the set of nodes which are connected to the server during the whole movement. Thereafter, the file accesses of the participating nodes are generated based on the probability distributions described later in this section. This completes the whole input sequence for one simulation run.

In the third phase an optimal offline solution is calculated for the current input sequence. This is done centrally based only on the energy-distances and file accesses, i.e. the optimal offline algorithm does not use information about node

positions or routing paths.

In the fourth phase the online algorithms and the offline solution are executed in parallel on the simulated MANET. In this phase no global information are used. The online algorithms execute the distributed algorithms described in Section 4.1 on the server and the participating nodes. Request- and action-messages issued by the server or a participating node are forwarded from hop to hop based on the precalculated routing paths. Each mobile node, but not the server, has one power consumption counter for each simulated algorithm. A transfer of a data unit increases the power consumption counters of the corresponding algorithm in the sending and the receiving node. The participating nodes additionally increase their counters in every step by their stand-by power consumption. At the end of the simulation the registered power consumptions of the mobile nodes are added up for every algorithm.

Since the movement and input generation are randomized, a single simulation run is not representative for a given instance of parameters. We therefore performed several simulation runs for each value of the independent variable. For every experiment we conducted at least three simulation batches. A batch consisted of at least ten simulation runs for every value of the independent variable. All the simulation runs of a batch used the same movement of nodes, power consumption parameters and routing paths, but different selections of participating nodes and request sequences. In other words, we executed simulation phase one once per batch and simulation phases two to four for each simulation run. Thus, if the independent variable can for example take five different values, we performed three batches and in each batch we performed ten simulation runs per value of the independent variable This results in $3 \cdot 5 \cdot 10 = 150$ performed simulation runs for this example experiment.

4.3 Mobility model

The movement of the nodes is generated according to the "mobility model based on social network theory" described in [MM07]. This mobility model is primarily based on an interaction matrix M which represents the degree of relatedness of every pair of nodes. Entry $m_{i,j}$ of M can take values in the range $[0, 1]$, where a value of 0 means that there is no interaction between nodes i and j and 1 indicates a strong interaction. The generation of M is based on the connected-caveman graph from [Wat99]. First, 300 isolated communities of 5 nodes each are created. The nodes of these communities are initially fully connected to each other. Then each edge is re-wired with probability 0.1 such that it points to another community. The resulting graph is called the *connectivity graph*. The value of $m_{i,j}$ is then chosen

uniformly at random either from the range $[0.9, 1]$ if the edge (i, j) exists in the connectivity graph, or otherwise from the range $[0, 0.1]$.

Initially each community is assigned to a random cell of the 15×15 grid. Each member of the community is placed at a random position in the chosen cell. Then for 1350 nodes a random speed is chosen from the range $[1\frac{m}{s}, 2.5\frac{m}{s}]$ which remains unchanged during the whole movement instance. The used velocity range correlates approximately to the walking speed of humans. The other 150 nodes are stationary. The first goal of every node is chosen randomly inside its initial cell. Whenever a node reaches its current goal, it chooses its next goal randomly using the following probability distribution: let $C_{p,q}$, $1 \leq p, q \leq 15$ be the set of nodes which have the cell (p, q) as its current goal. Then

$$Pr\,(i \text{ chooses cell } (r,s) \text{ as its next goal}) := \frac{\sum\limits_{j \in C_{r,s}} m_{i,j}}{\sum\limits_{p,q} \sum\limits_{j \in C_{p,q}} m_{i,j}}.$$

Thus, node i prefers cells which are the goal of nodes related to it. The nodes always start instantly to move to their next goal, i.e. in contrast to the random waypoint mobility model they do not pause.

Since the initial placement of nodes results in a rather untypical node distribution, we started the actual simulation of the file allocation system after the nodes moved for 1000 steps. This means that most of the nodes already reached their second goal and thus the social component of the movement started to be in effect.

We have chosen this mobility model, because it is a good compromise between synthetic mobility models and recorded traces of real movements. The widely used synthetic mobility models (e.g. the random walk [Ein56], random waypoint [JM96] and random direction [RMSM01] mobility models) produce movements where every node moves independent of all the other nodes. This behavior is very uncommon for social beings like humans. The main advantage of these synthetic mobility models is that they are easy to implement and that they can generate arbitrary many different movement instances. Traces of real movements on the other hand provide realistic movements, but are hard to gain and handle. There are several free movement traces available (e.g. the project CRAWDAD [KH05, KH10] provides an archive). The main problem is that different traces have completely different settings, i.e. they use different hardware, different number of nodes, and different area sizes. Thus, it is hard to perform the same experiments on different movement instances. But using only one movement trace bears the danger to measure phenomenons which are caused by this special movement instance. The chosen "mobility model based on social network theory" provides more realistic movements than the synthetic models and can easily generate different instances. In [MM07] this mobility model was compared to

the random waypoint mobility model and a trace of real movement recorded by Intel (cf. [CHC+05]). They examined the contact duration, i.e. the duration two nodes can communicate nonstop with each other, and the inter-contact time, i.e. the time between two contacts. Their experiments showed that the distributions of the contact duration and the inter-contact time of their mobility model are significantly more similar to the real traces than the one of the random waypoint mobility model.

4.4 Experiments

We conducted three different experiments.

Write/read-ratio with constant write probability
In this experiment we investigate the performance of the described algorithms under different ratios of the number of write requests to the number of read requests. The server and the participating nodes issue their requests randomly and independent of each other. In every step the server issues a write request with probability p_w and each participating node issues a read request with probability p_r. This means that the expected number of write requests is Tp_w and the expected number of read requests is nTp_r, where T is the number of simulated steps. Thus, the ratio of write request to read requests is $\frac{p_w}{np_r}$.

We set p_w to the fixed value of $\frac{1}{20}$ and p_r to values between $\frac{1}{5}$ and $\frac{1}{35}$. Thus, the expected write/read-ratios were between 0.005 and 0.035 if $n = 50$. At the same time the expected number of requests $T(p_w + np_r)$ declined with shrinking p_r. We performed this experiment with different numbers of participating nodes $n \in \{10, 50, 100\}$. Each simulation run consisted of $T = 10,000$ simulation steps and the file size D was always 50.

Write/read-ratio with constant expected number of requests
In this experiment we also investigate the performance of the described algorithms under different ratios of the number of write requests to the number of read requests. But contrary to the prior experiment, we chose the values of p_w and p_r such that the expected number of requests is always 20,000 (2 requests per step). This is the case if $p_w = \alpha\frac{\beta}{\beta+1}$ and $p_r = \frac{\alpha}{n}\frac{1}{\beta+1}$, where α is the desired expected number of requests per step and β is the desired write/read-ratio. Thus, if β rises the expected number of write requests rises while the expected number of read requests falls. We have chosen the value of β between 0.005 and 0.035.

Each simulation run consisted of $T = 10,000$ simulation steps and involved 50 participating nodes. We conducted the experiment for different file sizes $D \in \{10, 30, 50, 70, 100\}$.

Heterogeneous request rates
In this experiment we investigate the performance of the described algorithms when the participating nodes have heterogeneous read probabilities. The simulations involved 50 participating nodes and lasted for 10,000 steps. Each participating node i issued a read request to the file with probability p_i, $1 \leq i \leq 50$, and the server issued write request with probability p_w in every step . We have chosen p_i uniformly at random from the range $[\frac{1}{30}, \frac{1}{10}]$ at the beginning of the input generation. The write probability p_w had a fixed value of $\frac{1}{15}$ which corresponds to the expected read probability of the participating nodes. We investigated the performance of the described algorithms in this scenario under different file sizes $D \in \{5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$.

4.5 Results

DoNothing has to pay for every read request, but has no costs for write requests. Thus, the competitive ratio of DoNothing is high if the write/read-ratio is low and decreases with increasing write/read-ratio. Since DoNothing never replicates, its costs are completely independent from D. The competitive ratio then again decreases with growing D, because the costs of the optimal offline algorithm are increasing.

ReplicateOnFirstRequest on the other hand pays only for the first read request from a participating node and thereafter for every write request. Thus, the competitive ratio of ReplicateOnFirstRequest are low if the write/read-ratio is low and increases with increasing write/read-ratio. The costs of ReplicateOnFirstRequest depends only on D because of the initial replications. If the input sequence is long enough the influence of D can be neglected.

Follow creates a copy on a participating node whenever this participating node issues a request and does not hold a copy and deletes all copies on every write request. This is only a good behavior if there are long sequences of read requests which are not interlaced by write requests. Since in our experiments this is usually not the case, Follow quiet often replicates and deletes copies. As can be seen in Figure 4.2, this results in a much worse performance than Count, DoNothing and ReplicateOnFirstRequest, which rarely create and delete copies. We will therefore concentrate in the following on the comparison of Count with DoNothing and ReplicateOnFirstRequest.

We also calculated the upper bounds for the costs of Follow and Count from

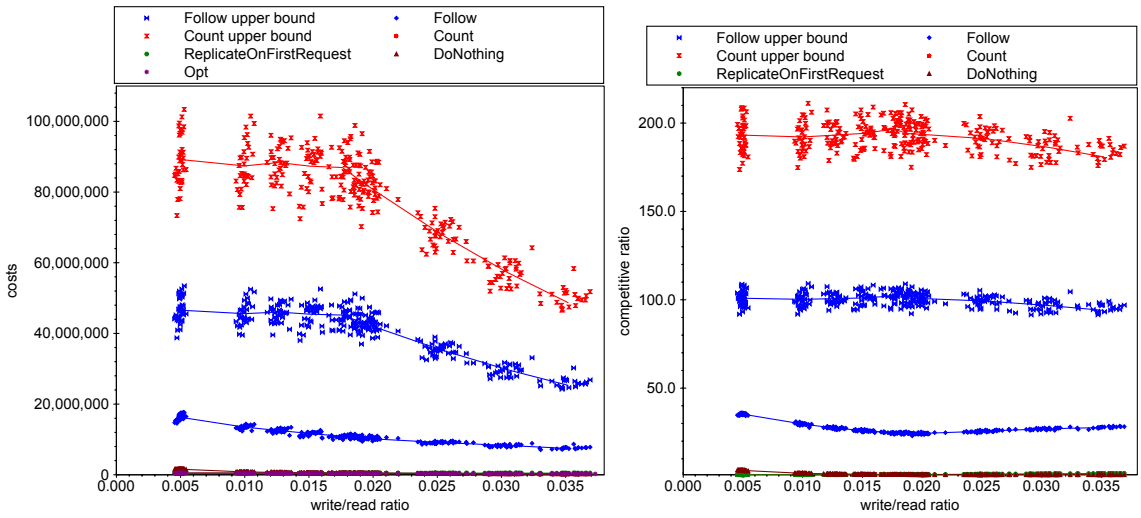


Figure 4.2: The performance of COUNT, FOLLOW and their upper bounds for different write/read-ratios with fixed write probability.

Theorem 3.2 respectively Theorem 3.4:

$$C_{\mathrm{F}}(\sigma) \leq \sum_{t=0}^{|\sigma|} \max\left(D+3,\ 1+\frac{D+1}{p}\bar{\delta}_t\right) \cdot C_{\mathrm{OPT}}(\sigma_t)$$

and

$$C_{\mathrm{C}}(\sigma) \leq \min_{0\leq r\leq 1}\left(\sum_{t=1}^{|\sigma|} \max\left(rD+(3-r),\ 1+\frac{(3-r)D+r}{p}\bar{\delta}_t\right) \cdot C_{\mathrm{OPT}}(\sigma_t)\right).$$

In all performed simulation runs the average value of $\bar{\delta}_t$ was in the range $[0.17, 0.26]$. For values of $\bar{\delta}_t$ in this range combined with the used average stand-by power consumption of $p = 0.01$, the upper bound of COUNT is lowest for $r = 1$. Thus, the effective upper bound for the competitive ratio of COUNT in our simulations is

$$C_{\mathrm{C}}(\sigma) \leq \sum_{t=1}^{|\sigma|} \max\left(D+2,\ 1+\frac{2D+1}{p}\bar{\delta}_t\right) \cdot C_{\mathrm{OPT}}(\sigma_t).$$

In all experiments the upper bounds for the costs of FOLLOW and COUNT were much higher than their actual costs. Figure 4.2 shows a representative example taken from the first experiment. The decrease of the upper bounds of the costs are thereby caused by the decrease of the optimal offline costs (c.f. Figure 4.3). Notice that the scale for the competitive ratio reaches a value of 200 while in all the following diagrams the competitive ratio only ranges up to 2. While the upper bound for FOLLOW was always significantly lower than the one for COUNT,

the actual costs of Follow were always significantly higher than the costs of Count. This suggests that the theoretical worst-case upper bounds are not good performance estimators for realistic inputs.

Write/read-ratio with constant write probability

At first, we will analyse the measured results for 50 participating nodes shown

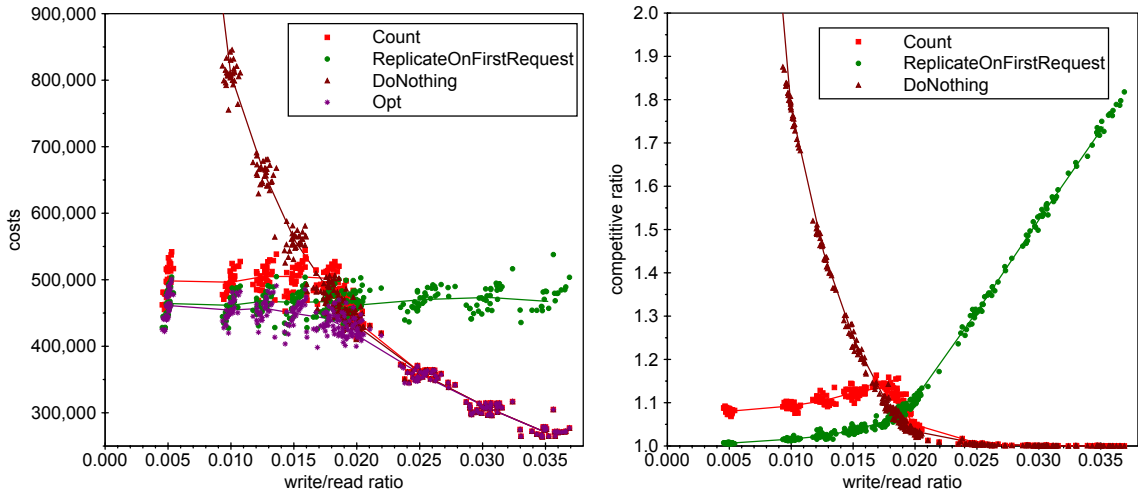


Figure 4.3: The performance of Count, DoNothing and ReplicateOnFirstRequest for different write/read-ratios with fixed write probability.

in Figure 4.3. In this experiment the expected number of write requests is constant. This implies that the costs for ReplicateOnFirstRequest are nearly constant. The expected number of read requests on the other hand sinks with the write/read-ratio. This is the reason for the decrease of the costs of DoNothing with rising write/read-ratio. The optimal offline solution creates, in the case of a write/read ratio of 0.005, copies on every participating node and keeps them. In the range 0.01 to 0.025 it creates and deletes copies as needed. And for write/read-ratios starting above 0.025 the optimal offline solution nearly never creates copies. Thus, ReplicateOnFirstRequest has a nearly optimal competitive ratio for low write/read-ratios and DoNothing has a nearly optimal competitive ratio for high write/read-ratios. But both algorithms have rather bad competitive ratios in the respectively opposite case.

In contrast, Count is relative good over all write/read-ratios. In the case of low write/read-ratios Count is slightly worse than ReplicateOnFirstRequest. This is because Count creates a copy at the earliest after D read request from a participating node, while ReplicateOnFirstRequest creates its copy immediately after the first request. For high write/read-ratios Count behaves basically like DoNothing, i.e. it never creates copies. For write/read-ratios in between Count

creates and deletes copies similar to the optimal offline solution, but shifted in time.

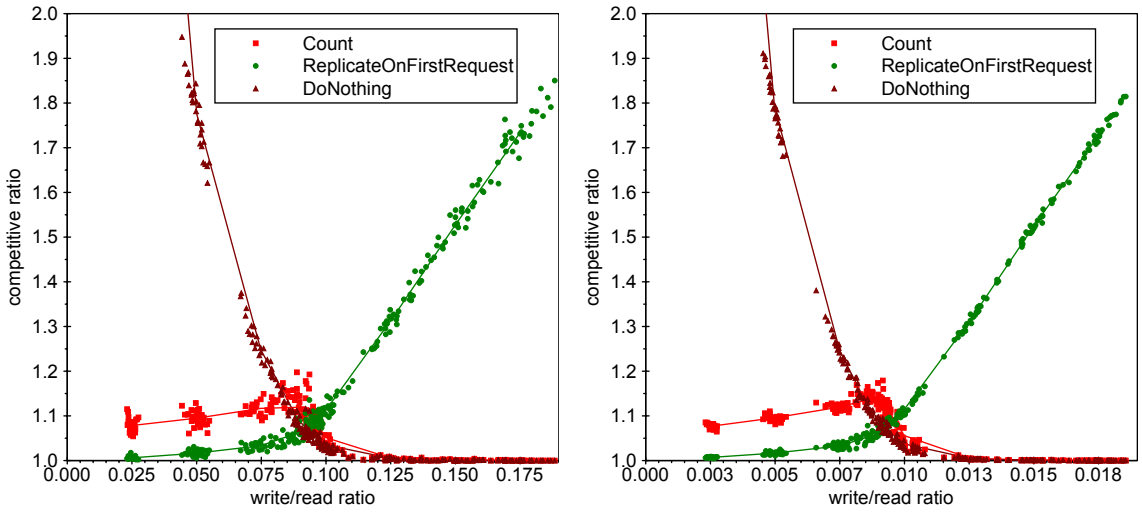


Figure 4.4: The performance of COUNT, DONOTHING and REPLICATEONFIRSTREQUEST for different write/read-ratios with fixed write probability for $n = 10$ (left) and $n = 100$ (right).

The diagrams of the competitive ratio for 10 and 100 participating nodes shown in Figure 4.4 have basically the same shape as the one for 50 participating nodes. The main difference is the scaling of the x-axis. This is due to the fact that the write/read-ratio $\frac{p_w}{np_r}$ depends on the number of participating nodes. The shape of the diagram does not change, because the costs caused by any participating node is defined only by the write requests from the center and its own read requests and is therefore independent of the accesses of the other participating nodes. This implies that the overall competitive ratio is some kind of averaging of the competitive ratios of all participating nodes for a common sequence of write requests. Thus, the overall competitive ratio depends mainly on the expected write/read-ratio of an individual participating node and is therefore independent of n.

Write/read-ratio with constant expected number of requests

At first, we will analyse the measured results for $D = 50$ shown in Figure 4.5. In this experiment the write probability $p_w = \alpha\frac{\beta}{\beta+1}$ takes values in the range $(0.009, 0.070,)$ while the read probabilities $p_r = \frac{\alpha}{n}\frac{1}{\beta+1}$ take values from the range $(0.038, 0.040)$. Thus, the read probabilities are nearly constant and the change of the write/read-ratio is mainly due to the increasing write probability. This leads to nearly constant costs for DONOTHING and linearly rising costs for REPLICATEONFIRSTREQUEST.

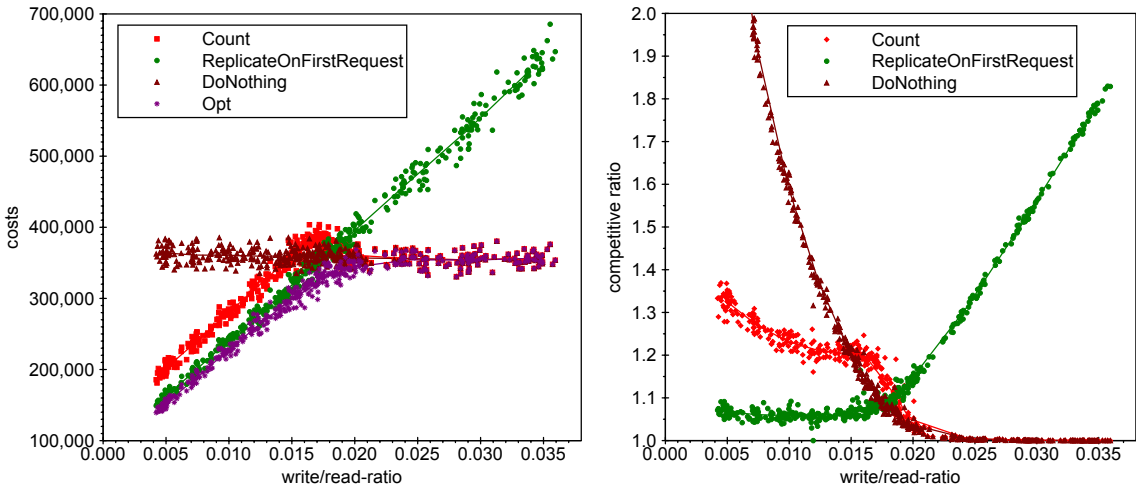


Figure 4.5: The performance of Count, DoNothing and ReplicateOnFirstRequest for different write/read-ratios with constant expected number of requests.

The competitive ratio of DoNothing is again bad for low write/read-ratios and improves while the write/read-ratio rises. ReplicateOnFirstRequest offers the opposite behavior. Its competitive ratio is good for low write/read-ratios and gets worse while the write/read-ratio rises.

The optimal offline solution and Count act basically just like in the first experiment: When the write/read-ratio is low they create many copies and rarely delete them again. When the write/read-ratio is high they nearly never create copies. And in between they create and delete copies if it is sensible. Count is again never better than both DoNothing and ReplicateOnFirstRequest, but offers for all write/read-ratios a reasonable good performance.

In the first experiment the expected overall number of requests highly differs for different write/read-ratios. It declines from about $100,000$ for a write/read-ratio of 0.005 to about $15,000$ for a write/read-ratio of 0.035. Since the expected number of requests in the second experiment has a constant value of $20,000$, we conclude that the competitive ratio depends mainly on the write/read-ratio but hardly on the overall number of requests.

Figure 4.6 shows the diagrams of the competitive ratio for the file sizes 10, 30, 70 and 100. Obviously, the optimal offline costs of a fixed input sequence rises if the file size increases. Furthermore, the number of copies an optimal offline algorithm creates will decrease with increasing D.

Since the costs of DoNothing are completely independent of D, the graph of its competitive ratio only differs because of the changes of the costs of the optimal solution. For growing D, the point where creating no copies is optimal is reached at lower write/read-ratios.

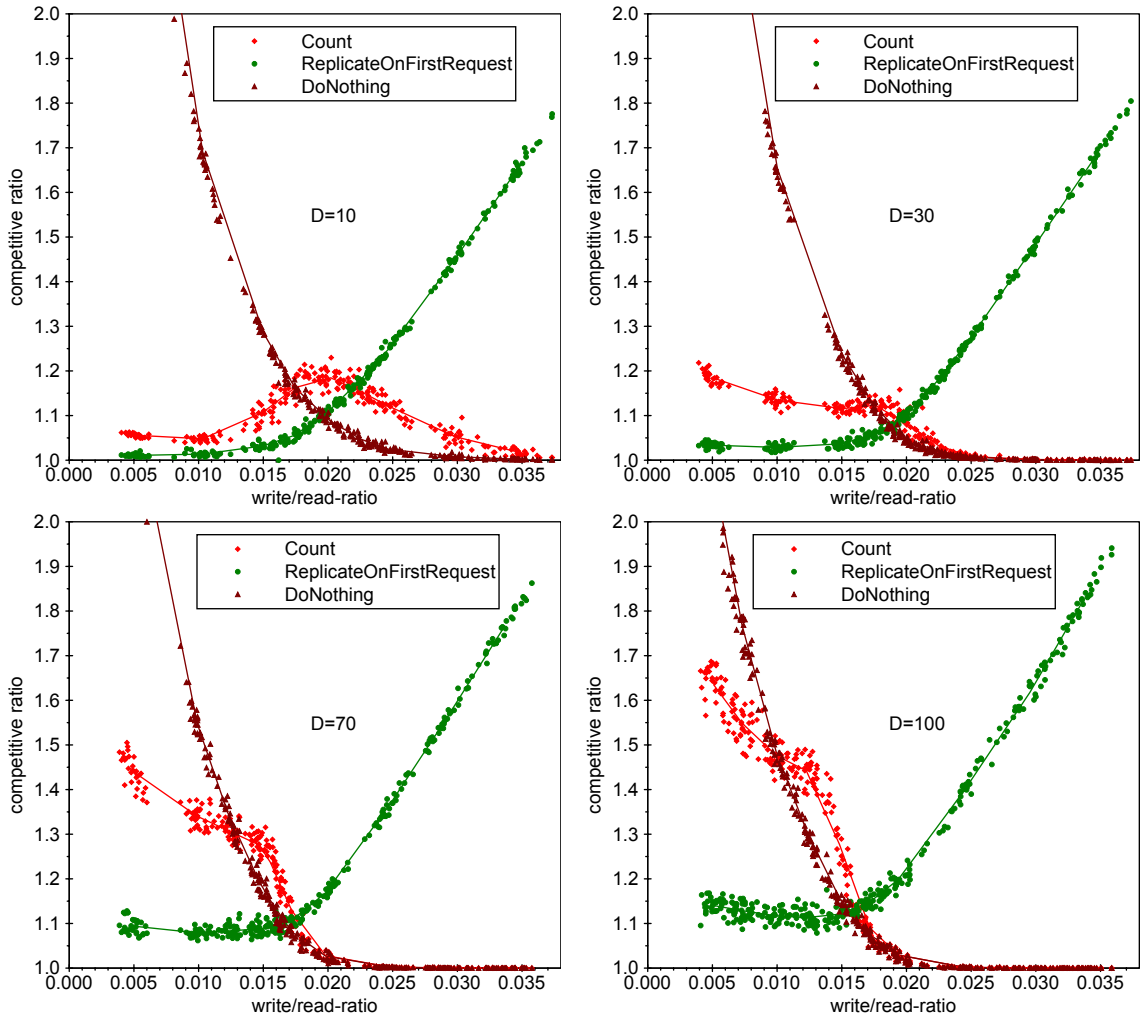


Figure 4.6: The competitive ratios of COUNT, DONOTHING and REPLICATEONFIRSTREQUEST for different write/read-ratios with fixed expected number of requests for $D \in \{10, 30, 70, 100\}$.

As can be seen in the charts, the curves of the competitive ratio of REPLICATEONFIRSTREQUEST is shifted upwards for growing D, i.e. the competitive ratio grows with D. This is because the costs of REPLICATEONFIRSTREQUEST are not independent of D since it replicates to every participating node right after its first read request. If the optimal offline algorithm also creates a copy to fulfill this first read request, it usually does this a few steps sooner than the request in order to save costs. The competitive ratio of REPLICATEONFIRSTREQUEST rises for growing D, because these cost savings grow with D.

The higher the value of D is the more read requests have to be issued by a participating node until COUNT creates a copy on it. The optimal offline algorithm on the other hand mostly creates this copy much earlier than COUNT. This leads

to a rising competitive ratio of Count with growing D for small write/read-ratios. For high write/read-ratios the creation of copies gets less useful with growing D. Both Count and the optimal offline strategy therefore approach the behavior of DoNothing for high write/read-ratios.

Heterogeneous request rates

The points in the charts of Figure 4.7 correspond to the mean value of 3 simula-

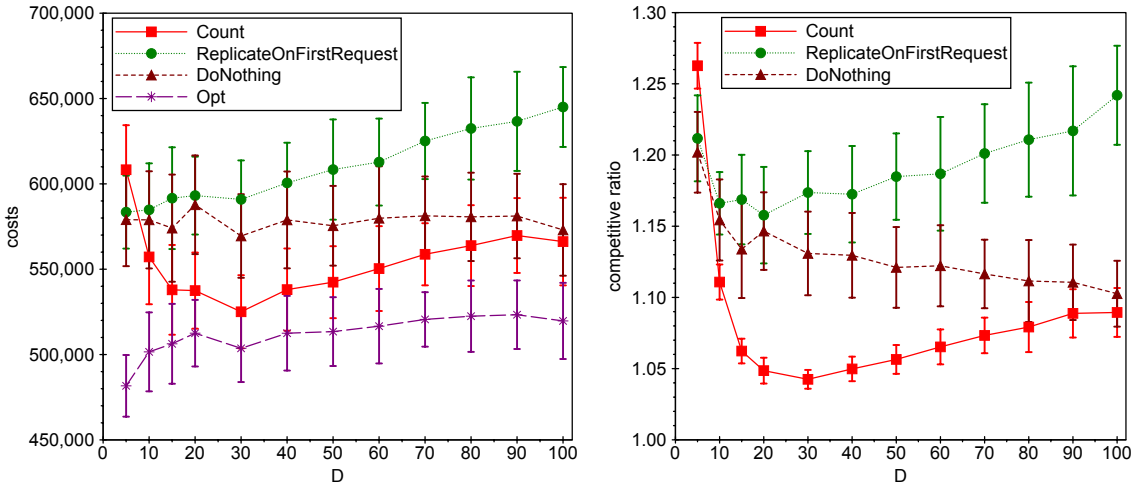


Figure 4.7: The performance of Count, DoNothing and ReplicateOnFirstRequest for different file sizes with heterogeneous request rates.

tion batches (overall 30 measured values). The charts furthermore indicate the standard deviation of the measured values.

In this experiment every participating node acts in the same way as in the first experiment. But here each participating node has its individual read probability and therefore its individual write/read-ratio. The diagrams in Figure 4.7 show that Count clearly outperforms DoNothing and ReplicateOnFirstRequest for values of D greater than or equal to 10. As seen in the analysis of the prior experiments Count is reasonable good for all possible write/read-ratios, but both DoNothing and ReplicateOnFirstRequest are bad in a certain range of write/read-ratios. Since the used range of read-probabilities contains the bad cases of DoNothing and ReplicateOnFirstRequest, this results in a better overall performance of Count.

The competitive ratio of DoNothing slightly decreases with growing D, because the optimal offline solution creates fewer copies for bigger values of D. This is also the reason for the growing competitive ratio of ReplicateOnFirstRequest. The competitive ratio of Count has the highest value for $D = 5$. Thereafter, the competitive ratio falls rapidly until $D = 30$ and then rises slowly with growing D.

Why is the competitive ratio of Count such bad for small values of D? Imagine that Count creates a copy on a participating node and thereafter no read requests are issued from this node until Count deletes this copy. Then Count has payed for the replication, D write requests and the deletion. This means Count has roughly payed the same as if it would have served $2D + 1$ further read requests without the copy. Thus, creating a copy at a wrong point of time can cause sever additional costs. An analysis of the number of created copies showed, that Count created significantly more copies than the optimal offline strategy for $D = 5$. This implies that Count created many unnecessary copies which drives its costs significantly up. For growing D the number of copies created by Count decreases and especially falls under the number of copies created by the optimal offline strategy at $D = 10$. This leads to a better competitive ratio since Count has to pay for fewer erroneous copy creations. The continued fall of the number of copies created by Count implies that the competitive ratio of Count approaches the competitive ratio of DoNothing.

4.6 Conclusion

The simulation based experimental evaluation of Count and Follow showed that their performance is significantly better on realistic inputs than the worst case upper bounds given in Chapter 3 suggest.

In all conducted simulation runs, the parameter $\bar{\delta}_t$ had an average value in the range $[0.17, 0.26]$ and p had an average value of 0.01. Since $\frac{D+2}{D+1}p < \frac{3}{2}p \approx 0.015 < 0.17 \leq \bar{\delta}_t$, the graph in Figure 3.5 suggests that the competitive ratios of Count and Follow are considerably above D. Furthermore, the graph implies that the competitive ratio of Count is higher than the competitive ratio of Follow. But actually, the competitive ratio of Count was in all simulation runs below 3 and therewith below the upper bound for static networks. While the competitive ratio of Follow was always significantly higher than the competitive ratio of Count, it was still significantly below the theoretical upper bound.

The theoretical upper bound for Follow for large enough $\bar{\delta}_t$ is the lowest possible, because Follow handles the lower bound for demand-driven algorithms provided in Subsection 3.1.2.1 in the best way possible for any competitive demand-driven online algorithm. But at the same time, Follow was significantly outperformed by the trivial strategies DoNothing and ReplicateOnFirstRequest in all experiments. This clearly shows that a good performance in the worst case does not imply a good performance on realistic inputs.

Count on the other hand turned out to be a good choice in all experiments. In the experiments with a fixed write/read-ratio, Count provided a reasonable good

performance for all write/read-ratios. In contrast to Count, the trivial strategies DoNothing and ReplicateOnFirstRequest were only good for certain write/read-ratios and quite bad for the other write/read-ratios. While Count was never better than both trivial strategies, it was also never much worse than the better of the trivial strategies. This resulted in a clearly better performance of Count in the experiment with heterogeneous write/read-ratios of the participating nodes.

CHAPTER 5

File Leasing

A major problem in a MANET is that connectivity can not be guaranteed. On the one hand, the movement of the nodes can lead to several connected components which are not connected among each other. The analysis of two real movement traces in [SBF+08] showed that the investigated MANETs consisted almost always of more than one connected component. On the other hand, connectivity can be disturbed by nodes which fail because of hardware or software failures or because they run out of energy. A file allocation system should therefore be able to handle disconnection of the participating nodes.

The model of the file allocation problem in a dynamic star network in the prior chapters assumes that each participating node is always connected to the server. If this assumption is not fulfilled, a file allocation system acting on this assumption can encounter inconsistencies in the copies of the file. If a participating node i which holds a copy gets disconnected, it will not receive updates from the server. Thus, every future read access from i could result in an invalid answer. Although the server would notice a failed update of the disconnected copy, it can not delete this copy until a new connecting path is available. But even when a new path appeared, the server is not automatically aware of it. Thus, the server either has to periodically try to send delete-messages to i or has to tolerate the invalid copy until i contacts him. In either case i could access invalid data arbitrary often.

In this chapter we introduce an alternative file allocation model which can deal better with disconnected copies. As in Section 3.1, we are looking at a file which is primary stored on a server and shared with a subset of participating nodes of a MANET. Instead of creating permanent copies on the participating nodes, the server grants leases of the file. This means that every copy of the file has an expiration date. If the copy of a participating node expires, it either drops the copy or renews the lease. Since the server and the participating node know when a lease expires, dropping the copy does not require a data transfer. The renewal of

the lease on the other hand has to be notified and therefore needs a data transfer. A renewal not only fails if the participating node is disconnected, but also if the copy is not up to date because of a prior failed update. Thus, an invalid copy will remain at most until the lease expires.

We give in Section 5.2 a lower bound of $\Omega(L)$ for the competitive ratio of randomized algorithms against oblivious adversaries. Furthermore, we analyse two simple deterministic algorithms in Section 5.3 which can be combined to a $O(LD)$-competitive algorithm.

5.1 Our model

The *file leasing problem in a dynamic star network* (FADS) is defined as follows: We are given a star network consisting of a center node c (the server) and n peripheral nodes $1, \ldots, n$ (the participating nodes of the MANET). All the peripheral nodes of the star can access an indivisible file which initially is stored on c. During the runtime of the system, an algorithm ALG can arbitrarily place copies of the file on the peripheral nodes. A copy of the file is leased for L steps. When a lease expires, it can either be removed or renewed. The center node always holds a copy. The *replica set* $R^t_{\text{ALG}} \subseteq \{c, 1, \ldots, n\}$ is defined as the set of nodes where algorithm ALG has a copy at the end of step t.

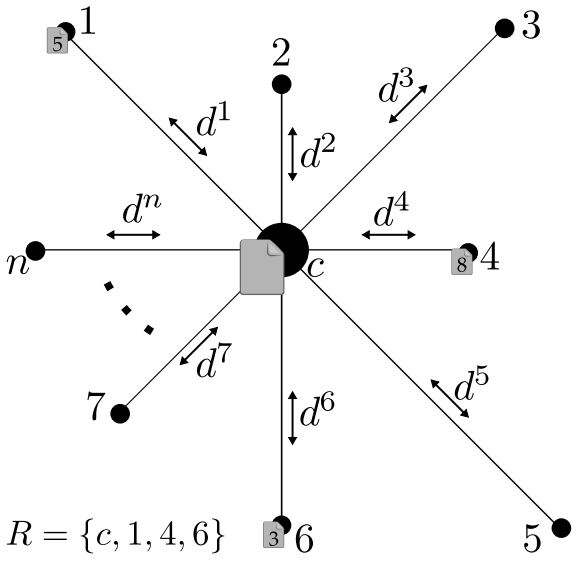


Figure 5.1: The file leasing system is modeled as a star with dynamic edge weights. Each copy has the residual term of its lease attached to it.

The input consists of the initial distances d_0 and a finite sequence σ of pairs (d_t, a_t),

$1 \leq t \leq |\sigma|$, of a vector of distances and an access request. It is presented to the online algorithm in equidistant discrete time steps. The vector $d_t = [d_t^1, \ldots, d_t^n] \in \mathbb{R}_{>0}^n$ defines the distances of the peripheral nodes to the center in step t. The access request $a_t \in \{-, r_i, w_i \mid i \in \{c, 1, \ldots, n\}\}$ either indicates that no access occurs $(-)$ or a read (r) respectively a write (w) access is issued at node i. Every step t is divided into the following three phases, which are executed strictly in the given order: first the distances between the peripheral nodes and c are updated to d_t, then request a_t is served, and finally, the algorithm may create new copies, renew leases and/or delete copies.

We consider the shared file as an indivisible container of a fixed number of data units of uniform size. A single read respectively write access always affects just one unit of the file. While a read request can be fulfilled by a single copy, a write request has to update all the copies. Every transfer of such a data unit costs the corresponding distance. The costs $C_{\text{ALG}}(\sigma_t)$ incurred by ALG in step t are composed of two components: the transfer costs for fulfilling the data access and the transfer costs for creating and deleting copies and for renewing leases. The costs $C_{\text{ALG}}(a_t)$ for serving a data access from $i \in \{c, 1, \ldots, n\}$ are naturally defined by

$$C_{\text{ALG}}^t(-) := 0, \qquad C_{\text{ALG}}^t(r_i) := \begin{cases} 0, & \text{if } i \in R_{\text{ALG}}^{t-1} \\ d_t^i, & \text{otherwise} \end{cases}, \qquad C_{\text{ALG}}^t(w_i) := \sum_{j \in (R_{\text{ALG}}^{t-1} \cup \{i\}) \setminus \{c\}} d_t^j$$

Notice that $C_{\text{ALG}}(r_c)$ is always 0 and therefore a read access from the center node is equivalent to no access $(-)$. Furthermore, $C_{\text{ALG}}(w_i)$ is always at least d_t^i since the center always holds a copy. A replication of the file from the center to peripheral node i costs $D \cdot d_t^i$, where $D \geq 1$ is a fixed constant depending on the size of the file. If the lease of a copy expires, an algorithm decides whether the copy is deleted or renewed. The deletion is free since both the peripheral node and the server know when a lease expires. A renewal costs the current distance between the peripheral node and the server since both have to be aware that the copy persists. If an algorithm decides to delete the copy of a peripheral node i while its still is still valid, both the center and i have to know this. Thus, a communication between these two nodes has to take place which costs d_t^i. We denote by $C_{\text{ALG}}(\sigma)$ the overall costs incurred by algorithm ALG while serving input σ.

In the following we investigate only the case $L \geq D$. This is a reasonable assumption, because otherwise the costs for creating a copy would, in the majority of cases, be higher than the costs this copy could avoid during a single leasing period.

Related work

Different variants of leases are used for caching in distributed file systems [GC89] like Echo [BHJ+93] or MFS [Bur88]. Furthermore, an extended version of leases

has also been proposed for caching in large-scale systems like the World Wide Web [YADL98].

Our file leasing model can be seen as a generalization of the classic ski-rental problem, where a skier has to decide whether to rent a set of skis for 1 per day or to buy it for D. In our model the price for renting respectively buying changes over time, but has a fixed ratio of D, and the lifetime of a bought set of skis is exactly L days. This is basically a combination of the *Bahncard problem* introduced by Fleischer [Fle01] and the *ski rental problem with dynamic prices* introduced by Bienkowski [Bie08].

The Bahncard problem is inspired by the price system of the German railway company "Deutsche Bahn". The input consists of a stream of pairs (t, p) where t is the date of an unavoidable travel and p is the regular price of a ticket. The traveler can buy a discount card (Bahncard) for a fixed price with a fixed period of validity before he buys the next ticket. While the Bahncard is valid, any ticket costs only β times the regular price, where $0 < \beta < 1$. Thus, the main differences to our leasing model are that the costs of two consecutive tickets can change arbitrary in the Bahncard problem and that the possession of a Bahncard does not avoid all the costs of a request. In [Fle01] optimal $(2 - \beta)$-competitive deterministic algorithms are given for the Bahncard problem.

In the ski rental problem with dynamic prices the price of renting respectively buying a set of skis is determined by a lipschitz continuous function. As in our model the ratio of buying to renting is always a fixed constant. The main difference to our model is that a bought set of skis lasts forever.

Reduction lemma

The following lemma shows that regarding the competitive ratio every statement valid for 1-restricted adversaries is also valid for δ-restricted adversaries. We will therefore constrain our investigations to 1-restricted adversaries.

Lemma 5.1. *Let* ALG *be a γ-competitive (randomized) algorithm against δ-restricted adversaries. Then there exists for every $\delta' > 0$ an algorithm which is γ-competitive against δ'-restricted adversaries.*

Proof. Let $\sigma' = (d_t, a_t)_{1 \le t \le |\sigma|}$ be an arbitrary δ'-restricted input sequence. Then the input sequence $\sigma := \left(\frac{\delta}{\delta'} d_t, a_t\right)_{1 \le t \le |\sigma|}$ is obviously δ-restricted. Let ALG$'$ be the online algorithm which simulates on input σ' the behavior of ALG on input σ. Since the costs incurred by any algorithm on any input sequence is a linear combination of the distances, we have

$$C_{\text{ALG}'}(\sigma') = C_{\text{ALG}}(\sigma) \cdot \frac{\delta'}{\delta} \le \gamma \cdot C_{\text{OPT}}(\sigma) \cdot \frac{\delta'}{\delta} = \gamma \cdot C_{\text{OPT}}(\sigma').$$

□

5.2 Lower bound

Here, we give a lower bound for randomized algorithms against oblivious adversaries. In the proof we use the following version of Yao's min-max principle for online algorithms. The proof for this lemma can be found in [CLLR97].

Lemma 5.2 (Yao's min-max principle)**.** *If for every $C \in \mathbb{R}_{>0}$ a probability distribution π over input sequences σ exists such that*

(a) $\mathrm{E}_\pi[C_{\mathrm{Opt}}(\sigma)] \geq C$, *and*

(b) $\mathrm{E}_\pi[C_{\mathrm{Det}}(\sigma)] \geq \gamma \cdot \mathrm{E}_\pi[C_{\mathrm{Opt}}(\sigma)]$ *for every deterministic algorithm* Det,

then no randomized algorithm can be γ'-competitive against oblivious adversaries for any $\gamma' < \gamma$. □

Theorem 5.3. *The competitive ratio of every randomized algorithm for* Flds *against* 1*-restricted oblivious adversaries is at least*

$$\gamma(L, D) := \begin{cases} \frac{(L+1)^2}{4D}, & \text{if } D \leq L < 2D-1 \\ L+1-D, & \text{if } L \geq 2D-1 \end{cases}$$

Proof. We proof the theorem using Lemma 5.2. Let Det be an arbitrary deterministic online algorithm for Flds and $1 \leq i \leq n$ an arbitrary peripheral node.[1] The input sequences of π are concatenations of equal phases τ_T, where $1 \leq T \leq D$. We will choose an adequate value for T at the end of the proof. T depends only on L and D and therefore is constant during every input sequence. We show that for each τ_T holds:

(a) $\mathrm{E}[C_{\mathrm{Opt}}(\tau_T)] = \frac{D}{L+1-T}$, and

(b) $\mathrm{E}[C_{\mathrm{Det}}(\tau_T)] \geq (L+1-T)\frac{T}{D} \cdot \mathrm{E}[C_{\mathrm{Opt}}(\tau_T)]$.

The number of phases m is chosen such that $\mathrm{E}\left[C_{\mathrm{Opt}}(\tau_T^m)\right] = m \cdot \frac{D}{L+1-T} \geq C$.

Every phase τ_T (cf. Figure 5.2) consists of four stages: the clearance-stage, the expansion-stage, the request-stage and the contraction-stage. In the clearance-stage DL read requests are issued from c at a constant distance of 1. In the expansion-stage peripheral node i moves away from c with speed 1 for $L-T$ steps, i.e. $d_t^i = 1+t$ in the t-th step of the expansion-stage. During the expansion-stage c keeps on issuing read requests. In the request-stage peripheral node i issues T read requests with probability $\frac{1}{L+1-T}$, otherwise c issues T read requests. During the request-stage, peripheral node i stays at distance $(L+1-T)$. Finally in the contraction-stage, peripheral node i moves back to distance 1 while c issues read requests.

[1] The described phases can also be executed on all peripheral nodes in parallel.

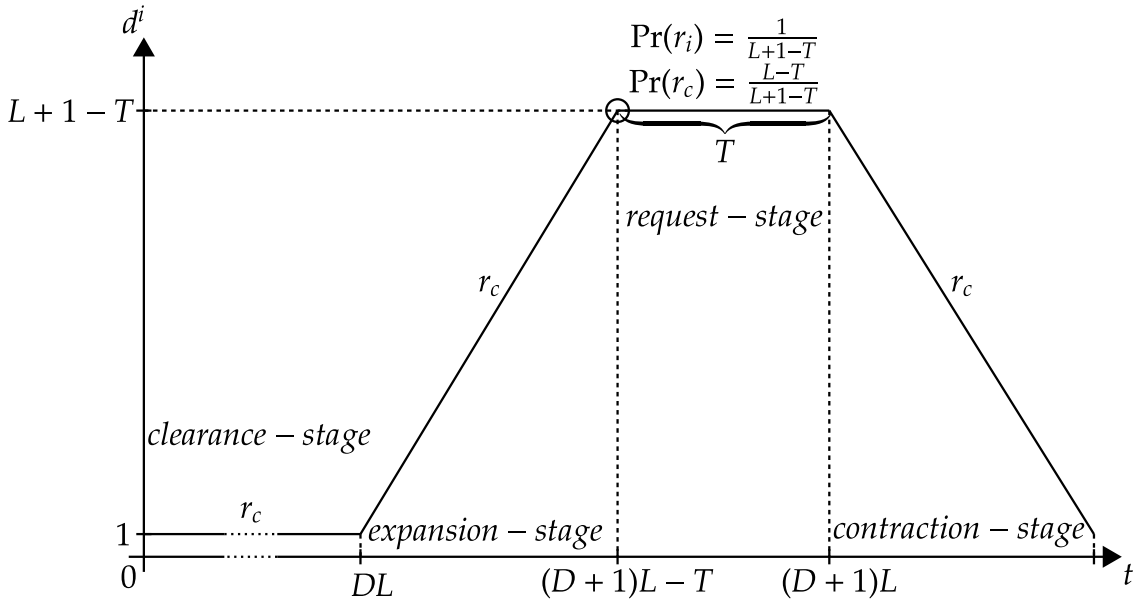


Figure 5.2: A single phase of the lower bound.

An optimal offline algorithm knows which node will issue requests during the request-stage beforehand. Thus, it will not pay anything if c issues requests. If i issues requests, OPT can either serve these requests by creating a copy or by sending data for each of the T requests. The cheapest way to serve read requests from i by a copy is by creating the copy in the last step of the clearance-stage, because a copy created earlier will not last until the end of the request-stage and a copy created later will be more expensive. Thus, OPT pays at most D. If OPT does not create a copy, it incurs costs of $T(L+1-T)$ which is at least $\min(L, D(L+1-D)) \geq D$ since this parabola has no local minimum and $1 \leq T \leq D \leq L$. Together this results in

$$\mathrm{E}\left[C_{\mathrm{OPT}}\right](\tau_T) = \frac{D}{1+L-T}.$$

We distinguish two cases regarding the behavior of DET: either DET has a copy on peripheral node i at the beginning of the request-stage or it does not.

If DET has a copy on i, this copy was either created in the current phase which costs at least D. Or the copy was already present at the beginning of the phase. Then DET had to pay at least D for renewals during the clearance-stage. Thus, in

both cases Det payed at least D in this phase. Together with $T \leq D$, we have

$$\begin{aligned}\mathrm{E}\left[C_{\mathrm{Det}}(\tau_T)\right] &\geq D \\ &= (L+1-T)\,\frac{D}{L+1-T} \\ &= (L+1-T)\,\mathrm{E}\left[C_{\mathrm{Opt}}(\tau_T)\right] \\ &\geq (L+1-T)\,\frac{T}{D}\mathrm{E}\left[C_{\mathrm{Opt}}(\tau_T)\right].\end{aligned}$$

If Det does not hold a copy on i at the beginning of the request-stage, it has to pay for possibly occurring read requests from i. Since $T \leq D$, Det can not reduce its costs by creating a copy anymore. Thus,

$$\begin{aligned}\mathrm{E}\left[C_{\mathrm{Det}}(\tau_T)\right] &\geq T \cdot (L+1-T) \\ &= (L+1-T)\frac{T}{D} \cdot D \\ &> (L+1-T)\frac{T}{D}\mathrm{E}\left[C_{\mathrm{Opt}}(\tau_T)\right].\end{aligned}$$

So, in either case

$$\mathrm{E}\left[C_{\mathrm{Det}}(\tau_T)\right] \geq (L+1-T)\,\frac{T}{D} \cdot \mathrm{E}\left[C_{\mathrm{Opt}}(\tau_T)\right].$$

The theorem follows by choosing $T := \frac{L+1}{2}$ if $L < 2D-1$ and $T := D$ if $L \geq 2D-1$. □

5.3 Algorithms

Here, we analyse the competitive ratios of the two simple online algorithms NeverLease and AlwaysLease.[2] NeverLease (see Algorithm 8) does not create any copies at all. AlwaysLease (see Algorithm 9) creates a copy of the file whenever a peripheral node issues a read request and does not hold a copy. If the lease of a copy on a peripheral node expires, AlwaysLease renews it only if this peripheral node issued a read request in the same step. On a write request, AlwaysLease deletes all the copies of peripheral nodes but a possibly existing copy of the writing node.

Algorithm 8 NeverLease

1: **On** request (d_t, a_t) **do**
2: **end**

[2] We abbreviate NeverLease with NL and AlwaysLease with AL in indices.

Algorithm 9 ALWAYSLEASE

```
1: On request (d_t, a_t) do
2:     Let i be the source of the current request a_t.
3:     if a_t = r_i then
4:         if i ∉ R_AL then
5:             Replicate to i.
6:         else if i ∈ R_AL and the copy of i expires in step t then
7:             Renew the copy of i.
8:         end if
9:     else if a_t = w_i then
10:        for j ∈ R_AL \ {c, i} do
11:            Delete copy on j.
12:        end for
13:    end if
14: end
```

For the analysis of the algorithms we allot the costs incurred by an algorithm ALG as follows to the peripheral nodes: A peripheral node $1 \leq i \leq n$ is charged for all those costs that are caused by it. More precisely, peripheral node i pays for all costs incurred for creating and deleting copies and renewing leases on i. Furthermore, i is charged d_t^i in step t if i issues a read request and does not hold a copy, or if i holds a copy and a write request is issued by any node. We denote by $C^i_{\text{ALG}}(\sigma)$ the costs incurred by i while ALG serves the input sequence σ. Since all the costs incurred by ALG are mapped to exactly one peripheral node, $C_{\text{ALG}}(\sigma) = \sum_{i=1}^{n} C^i_{\text{ALG}}(\sigma)$ holds.

We start with the analysis of NEVERLEASE.

Theorem 5.4. NEVERLEASE *is strictly* $L\frac{L+3}{2}$*-competitive against* 1*-restricted adversaries.*

Proof. Let σ be an arbitrary input sequence and OPT be an optimal offline algorithm for FLDS. We show that for any peripheral node $1 \leq i \leq n$ holds

$$C^i_{\text{NL}}(\sigma) \leq L\frac{L+3}{2} \cdot C^i_{\text{OPT}}(\sigma).$$

This implies

$$C_{\text{NL}}(\sigma) = \sum_{i=1}^{n} C^i_{\text{NL}}(\sigma) \leq \sum_{i=1}^{n} L\frac{L+3}{2} \cdot C^i_{\text{OPT}}(\sigma) = L\frac{L+3}{2} \cdot C_{\text{OPT}}(\sigma).$$

We examine the processing of σ backwards and allot the costs incurred by NEVERLEASE to costs of OPT.

NeverLease incurs costs d_t^i in every step t in which i issues a request. On a write request or if Opt does not hold a copy on i, Opt also has to pay d_t^i to fulfill the request. Then we have

$$C_{\mathrm{NL}}^i(\sigma_t) = d_t^i \leq L\frac{L+3}{2}d_t^i = L\frac{L+3}{2}C_{\mathrm{Opt}}^i(\sigma_t).$$

Otherwise, i issued a read request and Opt has a copy on i. Then Opt has created or renewed this copy in a step t' with $t > t' \geq t - L$. We allot all the costs of NeverLease in the steps after t' until t to Opt's costs for creating respectively renewing this copy. Thus,

$$\begin{aligned} C_{\mathrm{NL}}^i &\leq \sum_{s=t'+1}^{t} d_s^i \\ &\leq \sum_{s=1}^{L} \left(d_{t'}^i + s\right) \\ &= Ld_{t'}^i + \frac{L}{2}(L+1) \\ &\leq \left(L + \frac{L}{2}(L+1)\right) d_{t'}^i \\ &\leq L\frac{L+3}{2}C_{\mathrm{Opt}}^i. \end{aligned}$$

□

The proof of the competitiveness of AlwaysLease uses the same technique as the prior one, but is trickier.

Theorem 5.5. AlwaysLease *is strictly* $((D+5)L + D + 3)$*-competitive against* 1*-restricted adversaries.*

Proof. Let σ be an arbitrary input sequence and Opt be an optimal offline algorithm for Flds. We show that for any $1 \leq i \leq n$

$$C_{\mathrm{AL}}^i(\sigma) \leq ((D+5)L + D + 3)C_{\mathrm{Opt}}^i(\sigma) \tag{5.1}$$

holds. This implies

$$C_{\mathrm{AL}}(\sigma) = \sum_{i=1}^{n} C_{\mathrm{AL}}^i(\sigma) \leq \sum_{i=1}^{n}((D+5)L + D + 3)C_{\mathrm{Opt}}^i(\sigma) = ((D+5)L + D + 3)C_{\mathrm{Opt}}(\sigma).$$

We examine the processing of σ backwards and allot the costs incurred by AlwaysLease to costs of Opt. We thereby divide each step into two parts. In the first part the request is fulfilled by both AlwaysLease and Opt, and AlwaysLease performs actions. In the second part Opt performs actions.

ALWAYSLEASE only incurs costs in step t if i reads the file and it has no copy on i, or if a node $j \neq i$ writes to the file and it has a copy on i, or if i writes to the file. In the last case both ALWAYSLEASE and OPT have to pay d_t^i regardless of their configurations and (5.1) surely holds.

If i reads the file and ALWAYSLEASE has no copy on i, then ALWAYSLEASE pays $(D+1)d_t^i$ for serving the read request and creating a copy. If OPT has no copy on i, it has to pay d_t^i for the request and we are done. Otherwise, OPT has a copy on i and we search the last step t' before t where OPT incurs costs. This can only be due to a write access or due to OPT renewing or creating the copy on i. This means that $t' \geq t - L$ and $C^i_{\text{OPT}} \geq d^i_{t'}$ holds. Since ALWAYSLEASE would have a copy on i in step t, if there were a read request from i since t', we can conclude that only read requests from nodes other than i can occur between t' and t. Thus,

$$\begin{aligned} C^i_{\text{AL}} &\leq d^i_{t'} + (D+1)d^i_t \\ &\leq d^i_{t'} + (D+1)(d^i_{t'} + L) \\ &\leq (D+2+(D+1)L)d_{t'} \\ &< ((D+5)L + D + 3)d^i_{t'} \\ &\leq ((D+5)L + D + 3)C^i_{\text{OPT}}. \end{aligned}$$

If a node $j \neq i$ writes to the file and ALWAYSLEASE has a copy on i, then ALWAYSLEASE pays $2d_t^i$ for serving the write request and deleting the copy. If OPT also has a copy on i, it has to pay d_t^i for the request and we are done. Otherwise, OPT has no copy on i and we search the last step t' before t where OPT incurred costs. This can be because of a request from i or because OPT deleted a copy on i. But it is also possible that OPT does not have a copy because its prior copy expired after t'. We will treat this case below. So, if t' is determined by an access from i or a deletion of OPT, then $C^i_{\text{OPT}} \geq d^i_{t'}$ holds. Since ALWAYSLEASE has a copy on i and no read request from i occurred since t', we have $t' \geq t - L$ and there is no write request from any node between t' and t. Thus,

$$\begin{aligned} C^i_{\text{AL}} &\leq d^i_{t'} + 2d^i_t \\ &\leq d^i_{t'} + 2(d^i_{t'} + L) \\ &\leq (3+2L)d_{t'} \\ &< ((D+5)L + D + 3)d^i_{t'} \\ &\leq ((D+5)L + D + 3)C^i_{\text{OPT}}. \end{aligned}$$

Now for the special case in which OPT has no copy at the write request from j in step t, because its prior copy expired after t'. Let us denote the step when ALWAYSLEASE creates or renews its copy by t'' and the step when OPT loses its copy by t'''. Then, we have $t' < t''' < t$ and $t''' \leq t' + L$ and $t \leq t'' + L$. First note that

$t'' \neq t'$, because this would imply that the access in t' were a read request from i, but this can not cause costs for Opt since Opt has and keeps a copy on i since $t''' - L < t - L \leq t''$. Furthermore, between t''' and t only read requests from nodes other than i can occur, because Opt would incur costs on a request from i and a write request from another node than i would cause AlwaysLease to delete its copy earlier than t.

If AlwaysLease created respectively renewed its copy before t', then $t \leq t'' + L < t' + L$. This means that the costs of Opt at t' could only be caused by a write request from i, because Opt and AlwaysLease have a copy at t' and AlwaysLease still has its copy after t'. Since between t''' and t only read requests from nodes other than i can occur, this results in

$$\begin{aligned} C^i_{\mathrm{AL}} &= d^i_{t'} + 2d^i_t \\ &\leq d^i_{t'} + 2(d^i_{t'} + L) \\ &\leq (3 + 2L)d_{t'} \\ &< ((D+5)L + D + 3)d^i_{t'} \\ &\leq ((D+5)L + D + 3)C^i_{\mathrm{Opt}}. \end{aligned}$$

If AlwaysLease created respectively renewed its copy after t', then we have $t' < t'' \leq t''' < t$, because there are no read request from i after t'''. Together with $t''' \leq t' + L$ and $t \leq t'' + L$, this implies $t'' \leq t' + L$ and $t \leq t' + 2L$. As seen above between t''' and t only read requests from nodes other than i can occur. Between t' and t''' only read request can occur, because Opt has a copy but incurs no cost. Especially, a read request is issued by i in step t'' which causes AlwaysLease to renew or create its copy on i. Since AlwaysLease can create respectively renew only one copy during t' and t''', this results in

$$\begin{aligned} C^i_{\mathrm{AL}} &\leq (D+1)d^i_{t''} + 2d^i_t \\ &\leq (D+1)(d^i_{t'} + L) + 2(d^i_{t'} + 2L) \\ &\leq ((D+5)L + D + 3)d^i_{t'} \\ &\leq ((D+5)L + D + 3)C^i_{\mathrm{Opt}}. \end{aligned}$$

□

Since the parameters L and D are fixed throughout the whole input sequence, we can combine AlwaysLease and NeverLease to a strictly $\min\left((D+5)L + D + 3, L\frac{L+3}{2}\right)$-competitive algorithm.

5.4 Conclusion and open problems

In this chapter we have considered a new model for the file allocation problem in dynamic networks which uses renewable leases with a fixed period of validity. We have shown that this model overcomes the problem of hidden copies, which are moved far away from the server, described in chapter 2 and thereby allows competitive algorithms. This is because the costs of lease renewals do not allow an adversary to move a hidden copy far away from the center of requests for free. But this is in fact just a slight weakening of the main problem of online file allocation. The adversary still can hide its actual configuration from the online algorithm. And so the given lower bound basically depends on the duration the adversary can hide a copy without paying for it.

In our model we defined that every write access is forwarded to all existing copies immediately. But this is actually just one reasonable behavior. The server could also invalidate existing leases either immediately or on the next renewal request. This would give an adversary the power to delete the copies of an online algorithm. Furthermore, the server could defer all the updates of a copy until the next renewal of its lease. This would mean that the size of the answer to a lease renewal would depend on the number of updated data units of the file. While this is surely a reasonable strategy for a practical implementation, it complicates theoretical analysis because of the varying lease renewal answer sizes. Model variants with deferred reactions on write accesses have the disadvantage of temporary inconsistency of the content of the copies. Finally, a vast amount of hybrid strategies to handle write accesses could be implemented. The lower bound given in Section 5.2 holds for all these model variants since it does not contain any write requests.

A possible generalization of the model is to allow an algorithm to assign different durations of validity to leases. In this model the lower bound given in Section 5.2 still holds with the difference that L now stands for the longest allowed duration of validity of a lease. This means especially that if there is no upper bound for the duration of validity of a lease, no competitive online algorithm is possible.

We gave a lower bound of $\Omega(L)$ and two simple $O(DL)$-competitive algorithms. So there is still a big gap between the upper and the lower bounds. We conjecture that the following extension LEASECOUNT of the algorithm COUNT from Subsection 3.1.2.3 is $O(L)$-competitive. LEASECOUNT behaves exactly as COUNT with one exception, whenever a lease expires it deletes the corresponding copy and sets the corresponding counter to 0. The attempt to prove this conjecture with the help of a potential function argument failed because of the following universal problem: The potential function obviously has to depend on the distances of the peripheral

nodes, because otherwise the costs for the creation respectively deletion of a copy could not be amortized by a decrease of the potential. But, it is possible that the adversary has no costs in one (or several) step(s), e.g. when the center node issues a read request. This especially means that the potential may not increase during this step, i.e.

$$\underbrace{C^i_{\text{ALG}}}_{\geq 0} + \Phi(d^i + \Delta d^i) - \Phi(d^i) \leq 0$$

But, if the potential function depends on the current distance of the peripheral node, this means that

$$\forall -\delta \leq \Delta d^i \leq \delta : \quad \Phi(d^i + \Delta d^i) - \Phi(d^i) \leq 0$$

has to hold. Thus, the decisive question is: How can you define a potential function that does not increase for both increasing and decreasing distance? In the model with stand-by costs in Chapter 3 this problem was avoided by the inevitable stand-by costs in every step.

Bibliography

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

[ABF98] Baruch Awerbuch, Yair Bartal, and Amos Fiat. Distributed paging for general networks. *Journal of Algorithms*, 28(1):67–104, 1998.

[ADM04] Ittai Abraham, Danny Dolev, and Dahlia Malkhi. LLS: a locality aware location service for mobile ad hoc networks. In *Proceedings of the Joint Workshop on Foundations of Mobile Computing (DIAL M-POMC)*, pages 75–84, New York, NY, USA, 2004. ACM.

[AL99] Susanne Albers and Stefano Leonardi. Online algorithms. *ACM Computing Surveys*, 31(3es), 1999.

[AP95] Baruch Awerbuch and David Peleg. Online tracking of mobile users. *Journal of the ACM*, 42(5):1021–1058, 1995.

[ASSC02] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[AW98] Susanne Albers and Jeffery Westbrook. Self-organizing data structures. In *Online Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 13–51. Springer, 1998.

[Bar94] Yair Bartal. *Competitive analysis of distributed on-line problems — distributed paging*. PhD thesis, Tel-Aviv University, 1994.

[Bar98a] Yair Bartal. Distributed paging. In *Online Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 1998.

[Bar98b] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 161–168, New York, NY, USA, 1998. ACM.

[BBKadH09] Marcin Bienkowski, Jaroslaw Byrka, Miroslaw Korzeniowski, and Friedhelm Meyer auf der Heide. Optimal algorithms for page migration in dynamic networks. *Journal of Discrete Algorithms*, 7(4):545–569, 2009.

[BDBK+94] Shai Ben-David, Allan Borodin, Richard M. Karp, Gábor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[BDK05] Marcin Bienkowski, Miroslaw Dynia, and Miroslaw Korzeniowski. Improved algorithms for dynamic page migration. In *Proceedings of the 22nd Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 365–376, 2005.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[BFR92] Yair Bartal, Amos Fiat, and Yuval Rabani. Competitive algorithms for distributed data managemen. In *Proceedings of the 24th Symposium on Theory of Computing (STOC)*, pages 39–50. ACM, 1992.

[BFR95] Yair Bartal, Amos Fiat, and Yuval Rabani. Competitive algorithms for distributed data management. *Journal of Computer and System Sciences*, 51(3):341–358, 1995.

[BHJ+93] Andrew D. Birrell, Andy Hisgen, Chuck Jerian, Timothy Mann, and Garret Swart. The Echo distributed file system. Technical Report 111, DEC Systems Research Center, 1993.

[Bie05a] Marcin Bienkowski. Dynamic page migration with stochastic requests. In *Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 270–278, Las Vegas, Nevada, USA, July 2005. ACM Press, New York, NY, USA.

[Bie05b] Marcin Bienkowski. *Page migration in dynamic networks*. PhD thesis, University of Paderborn, 2005.

[Bie08] Marcin Bienkowski. Ski rental problem with dynamic pricing. Technical Report TR032008, Institute of Computer Science, University of Wrocław, 2008.

[BJ05] Marcin Bienkowski and Byrka Jarosław. Bucket game with applications to set multicover and dynamic page migration. In *Proceedings of the 13th Annual European Symposium on Algorithms (ESA)*, volume 3669 of *Lecture Notes in Computer Science*, pages 815–826. Springer Verlag, October 2005.

[BK05] Marcin Bienkowski and Miroslaw Korzeniowski. Dynamic page migration under brownian motion. In *Proceedings of the European Conference in Parallel Processing (Euro-Par)*, pages 962–971, 2005.

[BKM04] Marcin Bienkowski, Miroslaw Korzeniowski, and Friedhelm Meyer auf der Heide. Fighting against two adversaries: page migration in dynamic networks. In *Proceedings of the 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 64–73, 2004.

[BM05] Marcin Bienkowski and Friedhelm Meyer auf der Heide. Page migration in dynamic networks. In *Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 1–14, September 2005. Invited talk.

[BS89] David L. Black and Daniel D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.

[Bur88] Michael Burrows. *Efficient data sharing*. PhD thesis, University of Cambridge, 1988.

[CHC+05] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Pocket switched networks: real-world mobility and its consequences for opportunistic forwarding. Technical Report UCAM-CL-TR-617, Computer Laboratory, University of Cambridge, 2005.

[CLLR97] Marek Chrobak, Lawrence L. Larmore, Carsten Lund, and Nick Reingold. A better lower bound on the competitive ratio of the randomized 2-server problem. *Information Processing Letters*, 63(2):79–83, 1997.

[CM99] M. Scott Corson and Joseph Macker. RFC 2501: Mobile ad hoc networking (MANET): routing protocol performance issues and evaluation considerations, January 1999.

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

[Dij59] Edsger J. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[Ein56] Albert Einstein. *Investigations on the theory of the Brownian movement*. Dover Publications, Inc., 1956.

[FKFP07] Sándor P. Fekete, Alexander Kröller, Stefan Fischer, and Dennis Pfisterer. Shawn: the fast, highly customizable sensor network simulator. In *Proceedings of the 4th International Conference on Networked Sensing Systems (INSS)*, page 299, 2007.

[Fle01] Rudolf Fleischer. On the Bahncard problem. *Theoretical computer science*, 268(1):161–174, 2001.

[FW06] Roland Flury and Roger Wattenhofer. MLS: an efficient location service for mobile ad hoc networks. In *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 226–237, New York, NY, USA, 2006. ACM.

[GC89] Cary C. Gray and David R. Cheriton. Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. *ACM SIGOPS Operating Systems Review*, 23(5):202–210, 1989.

[GS90] Bezalel Gavish and Olivia R. Liu Sheng. Dynamic file migration in distributed computer systems. *Communications of the ACM*, 33(2):177–189, 1990.

[IW91] Makoto Imase and Bernard M. Waxman. Dynamic Steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4:369–384, 1991.

[JM96] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. *Kluwer International Series in Engeneering and Computer Science*, pages 153–179, 1996.

[KH05] David Kotz and Tristan Henderson. CRAWDAD: a community resource for archiving wireless data at dartmouth. *IEEE Pervasive Computing*, 4(4):12–14, 2005.

[KH10] David Kotz and Tristan Henderson. CRAWDAD: a community resource for archiving wireless data at Dartmouth. `http://crawdad.cs.dartmouth.edu/`, July 2010.

[KMRS88] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[KPB+05] Alexander Kröller, Dennis Pfisterer, Carsten Buschmann, Sándor P. Fekete, and Stefan Fischer. Shawn: a new approach to simulating wireless sensor networks. In *Proceedings of the 3rd Symposium on Design, Analysis, and Simulation of Distributed Systems (DASD)*, pages 117–124, 2005.

[LRWY99] Carsten Lund, Nick Reingold, Jeffery Westbrook, and Dicky Yan. Competitive on-line algorithms for distributed data management. *SIAM Journal on Computing*, 28(3):1086–1111, 1999.

[MadHVW97] Bruce M. Maggs, Friedhelm Meyer auf der Heide, Berthold Vöcking, and Matthias Westermann. Exploiting locality for data management in systems of limited bandwidth. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 284–293, October 1997.

[Mah10] Peter Mahlmann. *Peer-to-peer networks based on random graphs*. PhD thesis, University of Paderborn, 2010.

[MM07] Mirco Musolesi and Cecilia Mascolo. Designing mobility models based on social network theory. *ACM SIGMOBILE Mobile Computing and Communications Review*, 11(3):59–70, 2007.

[MM09] Jan Mehler and Friedhelm Meyer auf der Heide. Power-aware online file allocation in mobile ad hoc networks. In *Proceedings of the 21st Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 347–356, New York, NY, USA, 2009. ACM.

[MMS88] Mark S. Manasse, Lyle A. McGeoch, and Daniel D. Sleator. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM symposium on Theory of computing (STOC)*, pages 322–333, New York, NY, USA, 1988. ACM.

[MMVW97] Bruce Maggs, Friedhelm Meyer auf der Heide, Berthold Vöcking, and Matthias Westermann. Exploiting locality for data management in systems of limited bandwidth. *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 284–293, 1997.

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

[PS02] Hans Jürgen Promel and Angelika Steger. *The Steiner tree problem*. Vieweg, 2002.

[RMSM01] Elizabeth M. Royer, P. Michael Melliar-Smith, and Louise E. Moser. An analysis of the optimum node density for ad hoc mobile networks. In *Proceedings of the International Conference on Communications (ICC)*, volume 3, pages 857–861, 2001.

[RS94] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38(6):683–708, 1994.

[SBF+08] Antoine Scherrer, Pierre Borgnat, Eric Fleury, Jean-Loup Guillaume, and Cèline Robardet. Description and simulation of dynamic mobility networks. *Computer Networks*, 52(15):2842–2858, 2008. Complex Computer and Communication Networks.

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

[Wat99] Duncan J. Watts. Networks, dynamics, and the smallworld phenomenon. *American Journal of Sociology*, 105(2):493–527, 1999.

[WMY06] Qin Wang, MarkHempstead, and Woodward Yang. A realistic power consumption model for wireless sensor network devices. In *Proceedings of the 3rd Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 286–295, Sept. 2006.

[WS04] Andrew Y. Wang and Charles G. Sodini. A simple energy model for wireless microsensor transceivers. In *Proceedings of the 47th Annual IEEE Global Telecommunications Conference (GLOBECOM)*, volume 5, pages 3205–3209, 2004.

[WY95] Jeffery Westbrook and Dicky C. K. Yan. The performance of greedy algorithms for the on-line Steiner tree and related problems. *Theory of Computing Systems*, 28(5):451–468, 1995.

[YADL98] Jian Yin, Lorenzo Alvisi, Michael Dahlin, and Calvin Lin. Using leases to support server-driven consistency in large-scale systems. In *Proceedings of the The 18th International Conference on Distributed Computing Systems (ICDCS)*, pages 285–294, Washington, DC, USA, 1998. IEEE Computer Society.