Big Data: Sublinear Algorithms for Distributed Data Streams

Dissertation

In fulfillment of the requirements for the academic degree
Doctor rerum naturalium (Dr. rer. nat.)

MANUEL MALATYALI

Heinz Nixdorf Institute & Computer Science Department
Paderborn University, Germany
Research Group Algorithms and Complexity

This work is part of the project “Distributed Data Streams in Dynamic Environments (DisDaS)” and was supported by the German Research Foundation (DFG) within the Priority Program “Algorithms for Big Data” (SPP 1736).

Reviewer:

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

Abstract

We consider a sensor network comprised of numerous nodes which sense the environment and are equipped with communication capabilities to convey this information to a server. The server evaluates a function (e.g., Maximum, etc.) based on the information at the sensor nodes currently observed. To fulfill this task, the sensor nodes can send their observations to the server, and in turn, the server can send messages directly to a sensor node or broadcast a message to all sensor nodes with unit costs. The objective is to minimize the total communication while the server computes the function.

We apply two different techniques to tackle this situation: First, we consider filter-based protocols and compare protocols against an optimal offline algorithm which knows the whole input in advance and sets filters in an optimal way. Second, we design and analyze protocols in the framework of dynamic algorithms. That is, given observations at a specific time step, we assume that only a fraction of the sensors identify a change in their observation in comparison to the previous time step. We aim at communication protocols which use communication depending on this fraction.

Zusammenfassung

Wir betrachten ein Sensornetzwerk aus zahlreichen Knoten, die die Umgebung beobachten und in der Lage sind, diese Informationen an einen Server zu übermitteln. Der Server evaluiert eine Funktion (z.B. Maximum, etc.) basierend auf den Informationen, die aktuell bei den Sensorknoten vorliegen. Zu diesem Zweck können die Sensorknoten und der Server Nachrichten schicken. Die Sensorknoten können an den Server, der Server wiederum eine Nachrichten direkt an einen Sensorknoten oder an alle Sensorknoten senden. Dabei haben alle oben genannten Nachrichten einheitliche Kosten. Das Ziel ist es, die gesamte Kommunikation zu minimieren, während der Server die gegebene Funktion berechnet.

In diesem Setting wenden wir zwei verschiedene Techniken an: Zunächst betrachten wir filterbasierte Protokolle und vergleichen Protokolle mit einem optimalen Offline-Algorithmus, der die Eingabe im Voraus kennt und die Filter optimal bestimmt. Zweitens entwerfen und analysieren wir Protokolle im Rahmen von dynamischen Algorithmen. Das bedeutet, zwischen zwei Zeitpunkten an denen eine Ausgabe berechnet wird, ändert sich nur für ein Bruchteil der Sensorknoten die beobachte Information. Es werden Kommunikationsprotokolle entwickelt mit einem Kommunikationsaufwand abhängig von der Anzahl der geänderten Sensoren.

CONTENTS

1 A Short Introduction **1**
1.1 Background 4
1.2 Basis of the Thesis 5
1.3 Outline 7

A Monitoring Problems using Filters **9**

2 Introduction to Filter-Based Algorithms for Distributed Streams **11**
2.1 Model Description 12
2.2 Problems Description 13
2.3 Description of Filter-Based Algorithms 14
2.4 Competitive Algorithms 16
2.5 Related Work 17

3 Node Existence & Domain Monitoring **21**
3.1 Introduction & Contribution 22
3.2 Existence – One-Shot Computation 23
3.3 Existence Monitoring 26
3.4 Domain – One-Shot Computation 31

4 Exact & Approximate Top-k Monitoring **33**
4.1 Introduction & Contribution 34
4.2 Preliminaries 35
4.2.1 Top-k-Value Monitoring 35
4.2.2 Exact Top-k-Position Monitoring 37

4.2.3 Discussion on Approx. Top-k-Position Monitoring 40
4.3 Maximum – One-Shot Computation 41
4.4 Top-K – One-Shot Computation 45
4.5 Exact Top-k-Position Monitoring 48
4.6 Lower Bound 51
4.7 Allow the Online Algorithm to Err 52
4.8 Lower Bounds for the Approx. Top-k-Monitoring Problem 56

5 Top-k-Position Monitoring against an Approximate Offline Algorithm 57
5.1 Introduction & Contribution 58
5.2 Lower Bound for Competitive Algorithms 59
5.3 Upper Bounds for Competitive Algorithms 60
5.3.1 The DENSEPROTOCOL 61
5.3.2 The SUBPROTOCOL 64
5.4 Error Augmentation 70

6 Future Research Perspectives 73

B Dynamic Algorithms 81

7 Introduction to Dynamic Algorithms 83
7.1 Model Description 84
7.2 Problems Description 85
7.3 Related Work 85

8 Fully Dynamic Algorithm for the FREQUENCY Problem 87
8.1 Introduction & Results 88
8.2 Frequencies – A One-Shot Computation 89
8.2.1 Constant Factor Approximation of Frequencies 89
8.2.2 Arbitrary Approximation of Frequencies 91
8.3 Maintaining Frequencies over Multiple Time Steps 94

9 A Communication-Efficient Data Structure for Top-k and k-Select Queries 97
9.1 Introduction & Results 98
9.2 Outline of the Data Structure 99
9.3 Initialization of the Data Structure 100
9.4 Update 105
9.5 Weak Select 107

9.6 Strong Approximate k-Select 108
9.6.1 One-Shot Approximate k-Select 113
9.6.2 Top-k 114

10 Fully Dynamic & Filter-Based APPROXIMATE COUNT DISTINCT 117
10.1 Introduction & Contribution 118
10.2 Count Distinct Monitoring – One Shot 119
10.3 Approximate Count Distinct Monitoring 121

11 Future Research Perspectives 123

Chapter 1

A Short Introduction

In the recent years streaming algorithms gained lot of attraction due to cheap storage hardware. The capability and the willingness to store large quantity of data arose. In every aspect of our current life data is being created. The amount of sensors is becoming larger (e.g. smartphones, internet-of-things devices, connected cars, etc.) and thus grows the amount of data generated on the fly. A fraction of this data is sent to a data center and is stored for a later analysis. To be capable of managing and analyzing these amounts of data, *sublinear* algorithms were designed. In the following we shed some light on these algorithms which are executed inside a data center.

A streaming algorithm processes the data one by one in a given order and is not allowed to access passed data[1]. The objective is to compute a function on the data and at the same time minimize the space required to fulfill this task. Since exact solutions often require random access to large parts of the data, it is allowed to introduce an error to the output. Interestingly, for numerous problems the introduced error yields algorithms which only use polylogarithmic memory. In presence of rich literature in this important as well as attracting research area, the term of **sublinear space** algorithms arose, also often referred to as *streaming* algorithms (see [Agg07, Mut05] for a survey).

Furthermore, in huge data centers, data is not only stored on one huge hard disk, but is instead also stored in a distributed manner. Each hard disk is accessed by a processor and the processors are tasked to collectively compute a function on the presence of the entire data and store this result at one dedicated processor, denoted by *server*. To fulfill the task the processors can send messages to the server and the server can send messages directly to the other processors. In such a scenario, the communication chan-

[1] Here we only consider so-called *one pass* algorithms. There are also multiple pass algorithms which can access passed data in the next pass.

nel between the server and the processors represents the bottleneck, since a congested communication channel slows down the overall execution time of the computation of the function. To prevent this, protocols using **sublinear communication** are designed in a distributed setting. Most notably this setting is represented by the distributed monitoring model introduced by Cormode, Muthukrishnan and Yi [CMY08].

We now switch the setting and consider one that includes the generation of the data and its transmission to the data center. Precisely, data is generated by a sensor node (e.g., by observing the local environment) and a sensor node can send a message containing its observation and its unique identifier to a server inside the data center. Furthermore, the server can send a message to a sensor node and is able to send a broadcast message to all sensor nodes at unit cost. We denote this model by the *Distributed Monitoring Model with a Broadcast Channel (DMBC)* and give a formal definition in Section 2.1. We assume discrete time steps, at the beginning of each time step the sensor nodes observe their environment then a communication protocol is allowed to take place. We assume that a message is delivered instantaneously and the function is computed at each time step after the protocol terminates.

The essential differences between the settings within the data center and our setting represented by the DMBC model are that, on the one hand, only problems that relate exclusively to the current point in time are considered. On the other hand, the focus is not exclusively within a data center, but on the communication between mobile sensors and the data center itself. A server in the data center can broadcast messages via antennas or satellites, whereby devices receive messages from the server and send messages directly to the server. We consider the cost of broadcasting from the server to all devices with the same cost as sending a message to the server.[2] For the sake of simplification, we consider communication rounds: i.e., nodes first decide whether to send their input to the server, then the server responds with a broadcast message. In our model, we assume that all sensors can store the current observation. This means that an exact solution can be calculated for all problems as follows: Each sensor sends its observation to the server. In fact, if only a single communication round is allowed, for some problems the overall communication cannot be improved. The difficulty lies in deciding which observations are necessary to determine an output for the current point in time. For illustration the maximum of temperature values measured at the sensor nodes should be calculated. If two communication rounds are allowed, in the first round each of the n sensors sends with a probability of $1/\sqrt{n}$. The server broadcasts the maximum of the received values. Then all nodes with a higher value send a message

[2] Our results can also be applied within a data center. Here the costs of a broadcast are often estimated with logarithmic costs.

to the server. Altogether only $\mathcal{O}(\sqrt{n})$ messages on expectation are sent. That means with a second round of communication an algorithm with linear costs (w.r.t. communication) can be transformed into an algorithm with sublinear costs. In this thesis we will design protocols that often only require a polylogarithmic number of messages. If we compute an output only once we denote this by a *one-shot computation*.

Further Motivation Besides the fact that communication efficient transmission between devices and a data center seems to be a promising setting, there is a further motivation to consider this model. Consider the congested clique model (CC) introduced by Lotker et al. in [LPPSP03] which recently gained a lot of attention. In the CC model there are n nodes in a complete network (a clique). The time is divided into rounds; in each round a node is able to send $\mathcal{O}(\log n)$ bits to each of the $n-1$ neighbor nodes. A protocol is said to terminate if every node decides for the same output. For a given task the number of communication rounds is subject to being minimized. In the CC model it is of particular interest to prove superconstant lower bounds. In fact for several problems, protocols are known which only need constant communication rounds.

If we consider one of these protocols, it is easy to see that this can be mapped to a protocol which uses a constant number of communication rounds and a number of $\mathcal{O}(n)$ messages. The interesting question is whether there are protocols which only need $o(n)$ messages while the number of communication rounds is still constant.

Techniques We tackle the problem at hand by two different strategies: First, we resign to introduce any assumptions to the coarse of the input streams. That is, we assume the presence of an adversary who is able to generate at each point in time the input for the sensor nodes for the next time step. This setting allows to be generically applied to a variety of scenarios. In little detail, we apply techniques from the field of distributed databases known as **filters**, e.g., [MBP06]. Their purpose is to 'filter out' pieces of data which are not necessary for a computation. It turned out that filters can reduce costs significantly especially for data streams that are 'similar' in consecutive points in time. In Section 2.5 we give a detailed overview of the current state of research. In very short, Yi and Zhang [YZ12] were the first to consider a distributed scenario in which a server is tasked to be always informed about the value of one single sensor node, up to an additive error of Δ. Besides others, they designed and analyzed a protocol which uses $\mathcal{O}(\log(\Delta) \cdot \mathrm{OPT})$ messages, where OPT is the number of messages an optimal communication strategy uses. On the basis of this, Lam, Liu, and Ting [LLT10] were the first who applied this mix of filters and competitive analysis to the distributed

monitoring model. They considered the dominance relationship monitoring problem in which at every point in time the server is tasked to output the rank of each single sensor node.

Second, we tackle the problems from the perspective of **dynamic algorithms**. In little detail, we assume that the adversary is updating parts of the input and the server is asked to refresh the output. Clearly, if the adversary updates all observations, the costs for the new computation is the same as for one-shot computations. However, we consider a setting in which the adversary changes a fraction of the sensors. We analyze the number of messages and the time used depending on the amount of nodes that observed new values; however, the protocol itself does not know this fraction. Furthermore, they neither rely on these as an input nor try to guess it.

It is hard to keep up the strands of research previous to this aspect. A current field of research under the keyword of (fully) dynamic algorithms is often understood as adding or deleting one single item to or from the instance and recomputing the desired output. There are tons of examples which range from graph problems (e.g., Shortest Paths [FMS93]) to scheduling related problems (e.g., bin packing [FFG+18]). Here, we consider a more general variant in which a batch of updates is allowed to take place until the next computation takes place. Or in other words, a series of update operations is being called before an output is required. In its manifold details seeing this setting as a data structure reflects this most properly.

1.1 Background

One could get the impression that the research area of streaming algorithms is a recent trend. However, we like to shed some light on the very beginning of this field of research and give credits to the pioneers in this area.

Sequential Streaming Streaming Algorithms dates back to 1980 by Munro and Paterson [MP80] as they considered the scenario in which data is stored on a large magnetic tape. With respect to the data on the tape, the objective is to evaluate a function. Since accessing the data in a pattern driven by the course of an algorithm is very costly, Munro and Paterson identified an interesting class of algorithms which are restricted to access the data in the given order and they called these streaming algorithms. The crucial fact is that this model is essentially different from the model which allows random access (the RAM model). Even in their first work they have shown intriguing insights in the relation between the number of passes over the input and the potential quality of the evaluated function. These early breakthroughs still have an impact even

for modern technologies. In contrast to magnetic tapes, hard disks have an access time that is significantly faster; however, there is still a head which has to move to the place at which the required data lies. An algorithm that directly processes the data which is under this moving head improves the overall performance.

Distributed Streaming Most notable from the model perspective is the continuous distributed monitoring model described in [CMY08]. In contrast to sequential streaming, this model assumes that each of n sensors generates or observes a data stream in a synchronous manner. The goal is to continuously compute a function, i.e., to compute at every point in time the value of the function applied to all n data streams *generated so far*. This computation is done using a central server (also referred to as a coordinator) that is connected to all n sensors. The major objective is to minimize the communication volume, i.e., the overall number of bits transmitted between the sensors and the server.

An important class of problems investigated are threshold computations: Here, it is assumed that the generated streams are observations of events. The corresponding threshold problem is to continuously decide whether the current number of events has reached some given threshold τ or not. Exact characterizations, including matching lower bounds in the deterministic case, are known. Results for more complex functions like the entropy or the p-th frequency are presented in [ABC09, CMY08].

1.2 Basis of the Thesis

The thesis is based on the following publications. The following overview of the published results is considered within the DMBC model (distributed monitoring with a broadcast channel) which we described informally in the introduction and introduce formally in Section 2.1. Note that this model is an extension of the continuous monitoring model introduced by Cormode, Muthukrishnan and Yi in [CMY08].

Online Top-k-Position Monitoring of Distributed Data Streams

In the first paper, the server has to know the k nodes currently observing the largest values, for a given k between 1 and n. This problem is denoted as the Top-k-Position Monitoring problem. We design and analyze an algorithm that solves this problem while bounding the total amount of messages exchanged between the nodes and the server.

Our algorithm employs the idea of using filters which, intuitively speaking, leads to few messages to be sent if the new input is "similar" to the previous ones. The algorithm uses a number of messages that is by a factor of $\mathcal{O}((\log \Delta + k) \cdot \log n)$ larger than that of an offline algorithm that sets filters in an optimal way, where Δ is upper bounded by the largest value observed by any node.

- Mäcker, M., and Meyer auf der Heide. Online Top-k-Position Monitoring of Distributed Data Streams. In: *29th International Parallel and Distributed Processing Symposium (IPDPS, 2015)*, [MMM15].

On Competitive Algorithms for Approximations of Top-k-Position Monitoring of Distributed Streams

The server is supposed to keep track of an approximation of the set of nodes currently observing the k largest values. Such an approximate set is exact except for some imprecision in an ε-neighborhood of the k-th largest value. This approximation of the Top-k-Position Monitoring Problem is of interest in cases where marginal changes in observed values (e.g., due to noise) can be ignored so that monitoring an approximation is sufficient and can reduce communication.

This paper extends the results from [MMM15], where we have developed a filter-based online algorithm for the (exact) Top-k-Position Monitoring Problem. There we have presented a competitive analysis of our algorithm against an offline adversary that also is restricted to filter-based algorithms. Our new algorithms as well as their analyses use new methods. We analyze their competitiveness against adversaries that use both exact and approximate filter-based algorithms, and observe severe differences between the respective powers of these adversaries.

- Mäcker, M., and Meyer auf der Heide. On Competitive Algorithms for Approximations of Top-k-Position Monitoring of Distributed Streams. In: *30th International Parallel and Distributed Processing Symposium (IPDPS, 2016)*, [MMM16].

Monitoring of Domain-Related Problems in Distributed Data Streams

We consider monitoring problems related to the domain D_t to be the set of values observed by at least one node at time t. We provide randomized algorithms for monitoring D_t, (approximations of) the size $|D_t|$ and the frequencies of all members of D_t. Besides worst-case bounds, we also obtain improved results when inputs are parameterized according to the similarity of observations between consecutive time steps. This

parameterization allows to exclude inputs with rapid and heavy changes, which usually leads to the worst-case bounds but might be rather artificial in certain scenarios.

- Bemmann, Biermeier, Bürmann, Kemper, Knollmann, Knorr, Kothe, Mäcker, M., Meyer auf der Heide, Riechers, Schaefer, Sundermeier. Monitoring of Domain Related Problems in Distributed Data Streams. In: Structural Information and Communication Complexity (SIROCCO 2017) [BBB+17].

A Dynamic Distributed Data Structure for Top-k and k-Select Queries

Our goal is to compute the k currently lowest observed values (Top-k) or a value with rank in $[(1-\varepsilon)k;(1+\varepsilon)k]$ with a probability at least $1-\delta$ (approximate k-Select). We consider different variants of this problem:

1. Protocols which answer Top-k and k-Select queries as one-shot versions. These protocols are memoryless in the sense that they gather all information at the time of the request.
2. A dynamic data structure which at any point in time can answer a request for a rank k with a data item d with a rank 'close' to k.

We describe how to combine the two parts to receive a protocol answering the stated queries over multiple time steps. We present a communication-efficient dynamic data structure that supports these queries under updates of the data items arriving at the sensor nodes. We show that the bounds on the expected number of messages and the expected communication rounds decrease significantly.

- Biermeier, Feldkord, M., and Meyer auf der Heide. A Communication-Efficient Distributed Data Structure for Top-k and k-Select Queries. In: *15th International Workshop on Approximation and Online Algorithms (WAOA, 2017)*, [BFMM17].
- Feldkord, M., and Meyer auf der Heide. A Dynamic Distributed Data Structure for Top-k and k-Select Queries. In: *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, [FMM18].

1.3 Outline

The thesis is structured in two parts: In Part A we tackle the problems of using filters and apply competitive analysis. In Part B we design and analyze dynamic algorithms.

We start Part A with a technical introduction in Chapter 2. We propose the model in Section 2.1, describe the considered problems in Section 2.2 and apply filters specifically to the problems in Section 2.3. We then describe the considered adversary in Section 2.4 and present relevant related results in Section 2.5. In Chapter 3 we consider the problem of monitoring the existence and the domain and use this toy problem to illustrate first simple insights and apply a filter technique. In Chapter 4, protocols are designed which monitor the Top-k. In Chapter 5 the optimal offline algorithm is allowed to introduce an error. We sketch future research perspectives of filter-based algorithms in Chapter 6.

Part B considers dynamic algorithms and introduces the needed basis in Chapter 7. We consider the Frequency Problem as a toy problem in Chapter 8. In Chapter 9 we consider the Top-k and the k-Select problems under the aspect of dynamic algorithms. Chapter 10 combines both techniques of using filters and the aspect of dynamic algorithms to the problem of approximating the distinct count. Finally, the future of this area of research is sketched in Chapter 11.

PART A:

MONITORING PROBLEMS USING FILTERS

Chapter 2

Introduction to Filter-Based Algorithms for Distributed Streams

In the following, we consider the problems of monitoring the Existence, the Domain, the Top-k, and Approximate Top-k in a distributed sensor network. We introduce the Distributed Monitoring with a Broadcast Channel (DMBC) Model, which deals with distributed data stream and a single server which is asked to evaluate the given function on the basis of the observations in the current time step.

In this chapter we introduce the notation and techniques used. We design and evaluate our algorithms using competitive analysis: i.e., we compare the costs of our algorithm (which is not aware of the future updates of the data stream) against an optimal offline algorithm which knows the whole instance in advance.

The problems we look at do not admit bounded competitiveness without further restrictions on the optimal offline algorithm. Therefore, we use *filters* and restrict the optimal offline algorithm to be *filter-based*, i.e., to set filters in an optimal way. Intuitively speaking, a filter defines, for each sensor node, a set of observations at which the output need not change and thus, the sensor node does not need to send a message to the server. The competitiveness gives insight about the number of filters the online algorithm has to set in comparison to the offline algorithm, and thus sets a focus on the importance of knowing the future in advance or in other words measures the price of not knowing the future.

2.1 Model Description

In our setting there are n distributed nodes identified by unique identifiers (IDs) from the set $\{1, \ldots, n\}$, each receiving a continuous data stream $(v_i^1, v_i^2, v_i^3 \ldots)$, connected to a single server. Also, at time t, a node i observes $v_i^t \in \mathbb{N}$ and does not know any $v_i^{t'}$, $t' > t$. We omit the index t if it is clear from the context.

Following the model in [CMY08], we allow that between any two consecutive time steps, a *communication protocol* exchanging messages between the server and the nodes may take place. The communication protocol is allowed to use an amount of rounds which is polylogarithmic in n and $\max_{1 \leq i \leq n}(v_i^t)$. The nodes can send messages to the server, but not to each other. They are able to store a constant number of integers, compare two integers and perform Bernoulli trials with success probability $2^i/n$ for $i \in \{0, \ldots, \log n\}$. The server can, on the one hand, communicate to one device, and has, on the other hand, a broadcast channel to send one message received by all nodes at the same time. All the communication methods described above incur unit communication cost per message, the delivery is instantaneous, and we allow a message at time t to have a size at most logarithmic in n and $\max_{1 \leq i \leq n}(v_i^t)$.

Notation of Time Recall that a time step t defines a point in time at which the sensor nodes obtain a new piece of input (v_i^t for node i at time t). Between two consecutive time steps t and $t + 1$ a communication protocol takes place. The protocol consists of multiple (communication) rounds: A sensor nodes performs local computations and may decide to send a message to the server. The server collects all messages, performs local computations and (may) decide on a message to broadcast to all sensor nodes.

Since all nodes are synchronized, the server is able to identify the situation that no sensor decided to send a message and, on the other hand, the sensor nodes can identify that the server did not decide to send a message. Furthermore, the server is able to collect and read all messages given by the sensor node, even if every node has decided to send a message, i.e., the capacity of the communication channel in one round is not restricted.

If the communication protocol of each node for the current time step has terminated, the server decides on the output of the function at time t and the sensor network proceeds from time step t to the next time step $t + 1$.

2.2 Problems Description

In this section we consider four problems, specifically Existence Monitoring, Domain Monitoring, Top-k-Position Monitoring, and Approximate Top-k-Position Monitoring.

Definition 2.2.1 (Existence Monitoring)**.** *Let $v_i^t \in \{0,1\}$ hold for all i and t. At each time step t the server has to output the logical disjunction; i.e., the server outputs 1 if and only if there exists a sensor node i with $v_i^t = 1$.*

We generalize this problem to the Domain Monitoring problem. That is, at any point in time, the server needs to know the *domain*, the set of values observed at a particular time step. More formally:

Definition 2.2.2 (Domain Monitoring)**.** *Let $v_i^t \in \{1,\dots,\Delta\}$ hold for each i and t. At each time step t the server has to output set $D_t = \{v_1,\dots,v_n\} \subseteq \{1,\dots,\Delta\}$.*

Furthermore, we consider a problem which takes the ordering of the values into account. The *Top-k-Position Monitoring* problem asks the server to output the IDs of the nodes holding the k largest values.

Definition 2.2.3 (Top-k-Position Monitoring)**.** *At each time t the server has to output the value of a function $\mathcal{F}$ mapping from the current values $v_1^t,\dots,v_n^t$ to the set $\{s_1,\dots,s_k\}$, $k \leq n$, of nodes currently observing the k largest values among the (multi-)set $\{v_1^t,\dots,v_n^t\}$.*

We refer to this set of nodes as top-$k \coloneqq \{i \mid \exists j \in \{1,\dots,k\} : v_i^t = \pi(j,t)\}$, where $\pi(j,t)$ denotes the j-th largest value at time t. Explicitly, the values at times $t' < t$ do not matter for this output. To ease the presentation, we assume that all v_i are pairwise distinct at every time t. However, the main results remain valid if we drop this assumption.

We relax the definition stated above and study the following approximate variant of the problem in which the output is exact except for nodes in a small neighborhood around the k-th largest value.

Definition 2.2.4 (Approximate Top-k-Position Monitoring)**.** *Let $\pi(k,t)$ denote the node which observes the k-th largest value at time t and denote by top-k the nodes observing the k largest values. Given an error $0 < \varepsilon < 1$, for a time t we denote by $E(t) \coloneqq (\frac{1}{1-\varepsilon} v^t_{\pi(k,t)}, \infty]$ the range of values that are clearly larger than the k-th largest value and by $A(t) \coloneqq [(1-\varepsilon)v^t_{\pi(k,t)}, \frac{1}{1-\varepsilon} v^t_{\pi(k,t)}]$ the* ε-neighborhood *around the k-th largest value. Furthermore, we denote by $\mathcal{K}(t) \coloneqq \{i : v_i^t \in A(t)\}$ the nodes*

in the ε-neighborhood around the k-th largest value. Then, at any time t, the server is supposed to know the nodes $\mathcal{F}(t) = \mathcal{F}_E(t) \cup \mathcal{F}_A(t) = \{i_1, \ldots, i_k\}$ according to the following properties:

1. *$\mathcal{F}_E(t) = \{i : v_i^t \in E(t)\}$ and*
2. *$\mathcal{F}_A(t) \subseteq \mathcal{K}(t) = \{i : v_i^t \in A(t)\}$, such that $|\mathcal{F}_A(t)| = k - |\mathcal{F}_E(t)|$ holds.*

Denote by Δ the maximal value observed by some node (which may not be known beforehand). We use $\mathcal{F}_1 = \mathcal{F}(t)$ if t is clear from the context, $\mathcal{F}_2 = \{1, \ldots, n\} \setminus \mathcal{F}(t)$, and call $\mathcal{F}^*$ the *output* of an optimal offline algorithm. Note that the optimal algorithm is optimal with respect to the number of filters used during a phase and may output a set of nodes which reflects an approximate Top-k. If the k-th and the $(k+1)$-st largest value differ by more than $\varepsilon\, v^t_{\pi(k,t)}$, $\mathcal{F}(t)$ coincides with the set in the (exact) Top-k-Position Monitoring problem and hence, $\mathcal{F}(t)$ is unique. We denote by $\sigma(t) \coloneqq |\mathcal{K}(t)|$ the number of nodes at time t which are in the ε-neighborhood of the k-th largest value and $\sigma \coloneqq \max_t \sigma(t)$. Note that $|\mathcal{K}(t)| = 1$ implies that $\mathcal{F}(t)$ is unique. Furthermore for solving the exact Top-k-Position Monitoring problem we assume that the values are distinct (otherwise we break ties by using the nodes' identifier in case the same value is observed by several nodes).

2.3 Description of Filter-Based Algorithms

A set of filters is a collection of intervals, one assigned to each node such that, as long as the observed values at each node are within the given interval, the value of the function $\mathcal{F}$ does not change. For the problem at hand, this general idea of filters translates to the following definition.

Definition 2.3.1. *For a fixed time t, a* set of filters *is defined by an n-tuple of intervals $(F_1^t, \ldots, F_n^t)$, $F_i \subseteq \mathbb{N} \cup \{-\infty, \infty\}$ with $v_i^t \in F_i^t$, such that as long as the value of node i only changes within its interval, i.e., it holds $v_i^{t'} \in F_i^{t'} = F_i^t$ for $t' \geq t$, the value of the function $\mathcal{F}$ need not change. We use $F_i^t = [\ell_i^t, u_i^t]$ to denote the lower and upper bound of a filter interval, respectively.*

In our model, we assume that nodes are assigned such filters by the server. If a node *violates* its filter, i.e., the currently observed value is not contained in its filter, the node may report the violation and its current value to the server. The server then computes a new set of filters and sends them to the affected nodes. To this end, the server is allowed to assign "invalid" filters; i.e., there are affected nodes that directly

observe a filter violation. However, for such an algorithm to be correct, we demand that the intervals assigned to the nodes at the end of the protocol at time t and thus, before observations at time $t+1$, constitute a (valid) set of filters. We call such an algorithm *filter-based*. Note that the fact that we allow invalid filters simplifies the presentation of the algorithms in the following. However, using a constant overhead the protocols can be changed such that only (valid) filters are sent to the nodes.

Note that the easiest way of defining a set of filters is to assign the value currently observed by a node as its interval. In this case the usage of filters does not lead to any benefit, so in general we are looking for filters that are as large as possible to minimize the count of filter changes which is directly related to the number of exchanged messages.

In order to serve its purpose that the value of the function $\mathcal{F}$ does not change as long as the observed values are within their filters, we define in the following a filter as large as possible.

Definition 2.3.2 (Existence Filter)**.** *Denote by r a sensor node which is the representative with $v_r^t = 1$. Define for the representative r, the filter $F_r \coloneqq [1,1]$ and for each remaining node i, the filter $F_i \coloneqq [0,1]$. In case no node i observed the value $v_i = 1$, each node i defines its filter to be $F_i \coloneqq [0,0]$.*

The filters for the domain are a generalization of the Existence Filters:

Definition 2.3.3 (Domain Filter)**.** *Let R_t be a sequence $(j_1, \ldots, j_\Delta)$ of nodes such that for all observed values $v \in D_t$ a representative i is determined with $j_v = i$ and $v_i^t = v$. Define for each node i which is a representative ($j_v = i$), the filter $F_i \coloneqq [v,v]$. For each value $v \notin D_t$ which is not observed, no representative is given, i.e., $j_v = \text{nil}$. The filters of nodes i which are no representatives are defined as $F_i \coloneqq [-\infty, \infty]$.*

Now we consider the filters for the Top-k problem. Observe that each pair of filters (F_i, F_j) of nodes $i \in$ top-k and $j \notin$ top-k must be disjoint except for a (possible) single common point at their boundaries. Formally, we can state this observation as shown in the lemma below.

Lemma 2.3.4 (Top-k Filter)**.** *For a fixed time t, an n-tuple of intervals forms a set of filters for the Top-k-Problem if and only if*

1. *for all $i \in \mathcal{F}(t)$ it holds $v_i \in F_i = [\ell_i, u_i]$, and*
2. *for all $j \notin \mathcal{F}(t)$ it holds $v_j \in F_j = [\ell_j, u_j]$ and $u_j \leq \ell_i$, for all $i \in \mathcal{F}(t)$.*

The filter for the Approximate-Top-k Problem is similar to the Top-k filters up to an potential overlap of size $\varepsilon \cdot v_{k+1}$.

Lemma 2.3.5 (Approximate-Top-k Filters)**.** *For a fixed time t, an n-tuple of intervals forms a set of filters for the Approximate-Top-k Problem if and only if*

1. *for all $i \in \mathcal{F}(t)$ it holds $v_i \in F_i = [\ell_i, u_i]$, and*
2. *for all $j \notin \mathcal{F}(t)$ it holds $v_j \in F_j = [\ell_j, u_j]$ and $(1-\varepsilon)u_j \leq \ell_i$, for all $i \in \mathcal{F}(t)$.*

If a node observes a value inside the filter at time t and observes at time $t+1$ a value that is larger than the upper bound of its (previously defined) filter, we say the node *violates its filter from below*. A violation *from above* is defined analogously. If such a violation occurs, the node may report it and its current value to the server.

2.4 Competitive Algorithms

To analyze the quality of our online algorithms, we use analysis based on competitiveness and compare the communication volume induced by the algorithm to that of an appropriately defined offline algorithm. In our model, an offline algorithm knows all the input streams in advance and without further restrictions can trivially solve the Top-k-Monitoring Problem without any communication. In order to still reap meaningful results and assess the quality of our algorithm, we assume the optimal offline algorithm OPT to use filters assigned by the server to the nodes. Hence, to lower bound the cost induced by OPT, we will essentially count the number of filter updates over time.

We call an online algorithm ALG *c-competitive* if for every instance (i) its communication volume is by a factor of at most c larger than the communication volume of OPT, in case ALG is deterministic and (ii) the expected communication volume is by a factor of at most c larger than the communication volume of OPT, in case ALG is randomized.

Adversary We assume that the instance (for comparison between the online and offline algorithm) is given by an adversary. There are three different kinds of adversaries underlying for this comparison:

The *oblivious adversary* generates the whole instance in advance only with respect to the algorithm's code. In detail, the adversary can respond to deterministic choices by the algorithm but does not know the outcome of random experiments.

The *weak adaptive adversary* generates the instance piecewise, that is, only a current time step. In detail, the adversary has no access to the random process used by the algorithm but still foresees deterministic choices. In our case the weak adaptive

adversary can see the (deterministic) choices of filters applied to the sensor nodes but not the random sample chosen among the sensor nodes.

The *strong adaptive adversary* can access not only the algorithm itself, but also the outcome of random experiments. Informally speaking, against this adversary randomization does not give any benefit upon a deterministic algorithm.

In our work we will make use of the oblivious and the weak adaptive adversary and, if not stated otherwise, we compare the online algorithm against a weak adaptive adversary. Since we do not compare against a strong adaptive adversary we will simply use the terms oblivious and adaptive adversaries.

2.5 Related Work

The Continuous Monitoring Model is introduced in [CMY08] by Cormode, Muthukrishnan, and Yi with an emphasis on n nodes generating or observing *distributed* data streams and a designated coordinator. In this model the coordinator is asked to continuously compute a function, i.e., to compute a new output with respect to all observations made up to that point. The objective is to aim at minimizing the total communication between the nodes and the coordinator. We enhance the continuous monitoring model (as proposed by Cormode, Muthukrishnan, and Yi in [CMY08]) by a broadcast channel. Note that in comparison to our model, we introduce a dynamic data structure which computes a function only if there is a query for it. However, there is still a continuous aspect: to compute an estimate of k after every time step.

A general approach to reduce the communication when monitoring distributed data streams is proposed in [ZCPT09]. Zhang et al. use the notion of *filters*, which are also an integral part of our algorithm presented in this paper. They consider a problem called continuous skyline maintenance in which a coordinator is supposed to continuously maintain the skyline of dynamic objects. As they aim at minimizing the communication overhead between the coordinator and the objects, they use a filter method that helps avoid the transmission of updates from the objects to the coordinator in case these updates cannot influence the skyline maintained by the coordinator. More precisely, the dynamic objects are points of a d-dimensional space and filters are hyper-rectangles assigned by the coordinator to the objects in such a way that as long as these points are within the assigned hyper-rectangle, updates need not be communicated.

In their work [YZ12], Yi and Zhang were the first to study streaming algorithms with respect to their competitiveness. In their model there is one node and one coordinator and the goal is to keep the coordinator informed about the current value of a

multivalued function $f : \mathbb{Z}^+ \to \mathbb{Z}^d$ that is observed by the node and changes its value over time, while minimizing the number of messages. Among others, for $d = 1$, Yi and Zhang present an algorithm that is $\mathcal{O}(\log \delta)$-competitive if the last value received by the coordinator might deviate by δ from the current value of f. For arbitrary d, a competitiveness of $\mathcal{O}(d^2 \log(d \cdot \delta))$ is shown.

Following the idea of studying competitive algorithms for monitoring streams and the notion of filters, Lam et al. [LLT10] present an algorithm for online dominance tracking of distributed streams. In this problem a coordinator always has to be informed about the dominance relationship between n distributed nodes, each observing an online stream of d-dimensional values. Here, the dominance relation of two nodes b_1 and b_2 currently observing $(p_1, \ldots, p_d)$ and $(q_1, \ldots, q_d)$, respectively, both from $\{1, \ldots, U\}^d$, is defined as b_1 dominates b_2 if $p_i < q_i$ for some $1 \leq i \leq d$ and $p_j \leq q_j$ for all other $1 \leq j \leq d$. Their algorithm is based on the idea of assigning filters to the nodes and they show that a mid-point strategy, which basically sets filters to be the mid-point between neighboring nodes, is $\mathcal{O}(d \log U)$-competitive with respect to the number of messages sent in comparison to an offline algorithm that sets filters optimally.

In [DEI06], Davis, Edmonds and Impagliazzo consider the following resource allocation problem: n nodes observe streams of required shares of a given resource and let Δ denote the size of this resource. The server has to assign, to each node and in each time step, a share of the resource that is as least as large as the required share. (Their results assume that the sum of the shares required in one step is at most Δ). They assume real-valued required shares and show that there is no online algorithm with bounded competitiveness. Then, they present an online algorithm that assumes resource augmentation – the online algorithm may use a larger resource – and prove its competitiveness. The algorithm is based on the idea of "steal from the rich": A node that needs a larger share repeatedly steals small parts from nodes chosen randomly, proportionally to their current shares.

In [GK12], Giannakopoulos and Koutsoupias also applied the approach of competitive analysis, however, with a completely different fashion: They consider a single server which is given an input stream, processes it one by one in the given order and is allowed to store a fixed number of items of the data stream. The task is to continuously output the most frequent item and compare the quality of their output against an adversary which also is allowed to store the same number of data items but knows the whole input sequence in advance. They show that the maximum over all time steps does not lead to bounded competitiveness, but taking all time steps into account and averaging over all outputs leads to bounded competitiveness of $\mathcal{O}(\sqrt{m})$, where m denotes the

length of the data stream.

Further Related Work An interesting area of problems within this model are threshold functions: The coordinator has to decide whether the function value (based on all observations) has reached a given threshold τ. For well-structured functions (e.g. count-distinct or the sum-problem) asymptotically optimal bounds are known [Cor13, CMY11]. Functions which do not provide such structures (e.g. the entropy [ABC09]) turn out to require much more communication volume.

A related problem is considered in [BO03]. In their work, Babcock and Olston consider a variant of the distributed top-k monitoring problem: There is a set of objects $\{O_1, \ldots, O_n\}$ given, in which each object has a numeric value. The stream of data items updates these numeric values (of the given objects). In case each object is associated with exactly one node, their problem is to monitor the k largest values. Babcock and Olston have shown by an empirical evaluation that the amount of communication is an order of magnitude lower than that of a naive approach.

A model related to our (sub-)problem of finding the k-th largest values, and exploiting a broadcast channel is investigated by the shout-echo model [MG85, RSS86]: A communication round is defined as a broadcast by a single node, which is replied by all remaining nodes. The objective is to minimize the number of communication rounds, which differs from ours.

CHAPTER 3

NODE EXISTENCE & DOMAIN MONITORING

In this chapter, we tackle the Existence Problem and the Domain Problem and design one-shot and monitoring algorithms. On the one hand, these problems are toy problems to illustrate first simple ideas other protocols are based on. On the other hand, we introduce notations and show how to analyze the quality of an algorithm using competitive analysis where we compare our algorithm against an optimal filter-based offline algorithm.

The Existence Problem is a fundamental problem which occurs as a subtask in several different problems: Simply think of the question whether there is a node which observed a filter violation or not (recall that as long as each node observes values within their filter; i.e., within a range of values, the output need not change). If there is no filter violation, no communication needs to take place. This means, for each protocol it is of particular interest to efficiently identify filter violations.

In this chapter we design a protocol which solves the task to decide whether there exists a node i (which observed the value $v_i = 1$) or not with probability 1. The number of sent messages is based on a random process and we show an upper bound on the expected number of messages of $\mathcal{O}(1)$ (for one single execution). However, if there does not exist such a node, there is no communication at all. We will exploit this fact later in this section with respect to monitoring the existence; i.e., we decide the Existence Problem over multiple time steps.

3.1 Introduction & Contribution

Problem	Communication	Time
Existence	$\mathcal{O}(1)$	$\log n + 1$
Existence Monitoring	$\mathcal{O}(T)$	
	$\mathcal{O}(\log n)$-comp.	
Domain	$\mathcal{O}(\lvert D\rvert)$	$\log n + 1$
Domain Monitoring	$\mathcal{O}(T\ \lvert D\rvert)$	
	$\mathcal{O}\left(\log\left(\frac{n}{d_{min}}\right)\right)$-comp.	

We start the presentation of the competitive algorithms with a toy problem called EXISTENCE. Recall its definition (Section 2.2): Assume all nodes observe only binary values, i.e., $\forall i, t : v_i^t \in \{0, 1\}$. The server is asked to decide the *logical disjunction* for one fixed time step t, i.e., output $\mathcal{F}(t) = 1$, if and only if there **exists** a node i which observed the value $v_i^t = 1$. We extend this problem to the so-called DOMAIN problem (cf. Section 2.2): Assume all nodes observe (integral) values from $\{1, \ldots, \Delta\}$. The server is asked to output the **domain**, $D_t = \{v_1^t, \ldots, v_n^t\} \subseteq \{1, \ldots, \Delta\}$, i.e., each value which is observed by at least one sensor node.

A Short Note Although the EXISTENCE and DOMAIN problems share a common structure we use these two problems to present two different approaches to tackle its single shot variants which can be exchanged easily. The EXISTENCEPROTOCOL (presented in Algorithm 1) technically reflects the idea to match the probability of sending a message with the unknown size of nodes that observed the value 1. On the other hand, the DOMAINPROTOCOL for computing the DOMAIN uses a probability distribution of sending a message based on a geometric experiment.

The results for monitoring the Domain are based on the same approach as the Existence Monitoring Problem and the results directly follow; we omit its presentation.

Chapter Basis Parts of the model, analyses, and results for the one-shot computation presented in the remainder of this chapter are based on the following publications:

- Mäcker, M., and Meyer auf der Heide. On Competitive Algorithms for Approximations of Top-k-Position Monitoring of Distributed Streams. In: *30th International Parallel and Distributed Processing Symposium (IPDPS, 2016)*, [MMM16].
- Bemmann, Biermeier, Bürmann, Kemper, Knollmann, Knorr, Kothe, Mäcker, M., Meyer auf der Heide, Riechers, Schaefer, Sundermeier. Monitoring of Domain Related Problems in Distributed Data Streams. In: *Structural Information and Communication Complexity - 24th International Colloquium (SIROCCO 2017)*, [BBB+17].

3.2 Existence – One-Shot Computation

Consider the EXISTENCE Problem for a single time step. That is, we omit the index t and consider n nodes observing values $v_1, \ldots, v_n$ from the universe $\{1, \ldots, \Delta\}$. We start with a protocol which only uses a constant number of messages in expectation and uses a logarithmic number of rounds (recall that the model introduces the constraint to use at most a polylogarithmic number of rounds).

Observe that in our model one message is sufficient to decide the problem assuming the nodes have a unique identifier in $\{1, \ldots, n\}$ and the protocol uses n rounds: Consider a protocol in which node i sends in round $r = i$ a broadcast message if and only if no node has sent a message before. After n rounds the server can compute EXISTENCE. However, this protocol uses n communication rounds, which violates the constraint of using only a polylogarithmic number of rounds between each time step.

In contrast to such an approach where each node has its dedicated time slot to send a message, we on the one hand randomly map several nodes to the same time slot, but on the other hand show that the expected number of messages sent by the protocol is only constant.

> The simple idea of the EXISTENCEPROTOCOL is to synchronously let each node flip a coin with an initial success probability of $1/n$ and iteratively double the success probability until it reaches 1 or some node has sent a message and the protocol terminated.

Note that the EXISTENCEPROTOCOL is a Las Vegas algorithm; i.e., the algorithm is always correct and the number of messages needed is based on a random process.

Algorithm 1 EXISTENCEPROTOCOL

- Initially each node i defines $active_i = true$ iff $v_i = 1$ holds.
- In each round $r = 0, 1, \ldots, \log n$ the active nodes send messages uniformly and independently at random with probability $p_r := 2^r/n$.
- As soon as at least one message was sent or the γ-th round ends, the protocol is terminated and the server can decide EXISTENCE.
- If the last round $\gamma = \log n$ is reached, all active nodes i with $v_i = 1$ send a message with probability 1.

Analysis. We prove that it is sufficient to use a constant amount of messages on expectation and observe that the number of communication rounds is logarithmic in n.

To analyze the number of communication rounds simply observe that Algorithm 1 defines rounds $r = 0, \ldots, \log n$ and thus, uses at most $\log n+1$ communication rounds.

Observation 3.2.1. *The algorithm* EXISTENCEPROTOCOL *needs up to* $\log n + 1$ *number of communication rounds.*

Now we consider the number of messages used in this process:

Theorem 3.2.2. *The algorithm* EXISTENCEPROTOCOL *uses* $\mathcal{O}(1)$ *messages on expectation to solve the problem* EXISTENCE.

Proof. We analyze the above protocol and show that the bound on the expected number of messages is fulfilled. Let X be the random variable for the number of messages used by the protocol and b be the number of nodes i with $v_i = 1$. Note that the expected number of messages sent in round r is $b \cdot p_r$ and the probability that no node has sent a message before is $\prod_{k=0}^{r-1} (1 - p_k)^b$.

Observing that the function $f(r) = b \cdot p_r \left(1 - p_{r-1}\right)^b$ has only one extreme point and $0 \leq f(r) < 2$ for $r \in [0, \log n]$, it is easy to verify that the series can be upper bounded by simple integration:

$$
\begin{aligned}
\mathbb{E}[X] &\leq \frac{b}{n} + \sum_{r=1}^{\log(n)} \frac{b2^r}{n} \prod_{k=0}^{r-1} \left(1 - \frac{2^k}{n}\right)^b \\
&\leq 1 + \sum_{r=1}^{\log(n)} \frac{b2^r}{n} \left(1 - \frac{2^{r-1}}{n}\right)^b \\
&\leq 1 + \int_0^{\log(n)} \frac{b2^r}{n} \left(1 - \frac{2^{r-1}}{n}\right)^b dr + 2 \\
&\leq 3 + \left[\frac{b}{(b+1)n\ln(2)}(2^r - 2n)\left(1 - \frac{2^{r-1}}{n}\right)^b\right]_0^{\log n} \\
&\leq 3 + \frac{1}{n\ln(2)} \cdot \left((2^{\log n} - 2n)\left(1 - \frac{2^{\log n - 1}}{n}\right)^b + 2n\left(1 - \frac{2^{0-1}}{n}\right)^b\right) \\
&\leq 3 + \frac{1}{n\ln(2)} \left[(n - 2n)\left(1 - \frac{1}{2}\right)^b + 2n\left(1 - \frac{1}{2n}\right)^b\right]
\end{aligned}
$$

$$\leq 3 + \frac{1}{n\ln(2)}\left[(-n)\frac{1}{2^b} + 2n\left(1-\frac{1}{2n}\right)^b\right]$$
$$\leq 3 + \frac{1}{\ln(2)}\left(2\left(1-\frac{1}{2n}\right)^b - \frac{1}{2^b}\right)$$
$$\leq 3 + \frac{1}{\ln(2)}\left(2-\frac{1}{2^b}\right) \leq 3 + \frac{2}{\ln(2)} \leq 6 \ .$$

□

Applications. This protocol can be used for a variety of subtasks, e.g. validating that all nodes are within their filters, identifying that there is some filter violation or whether there are nodes that have a higher value than a certain threshold.

Corollary 3.2.3. *Given a time t. There is an algorithm which decides whether there are nodes which observed a filter violation using $\mathcal{O}(1)$ messages on expectation.*

Proof. For the distributed nodes to report filter violations we use an approach based on the EXISTENCEPROTOCOL to reduce the number of messages sent in case several nodes observe filter violations at the same time. The nodes apply the EXISTENCEPROTOCOL as follows: Each node that is still within its filter applies the protocol using a 0 as its value and each node that observes a filter violation uses a 1. Note that by this approach the server is definitely informed if there is some filter violation and otherwise no communication takes place. □

Observation 3.2.4. *Given a time step t. In case no node observed a filter violation no communication takes place.*

3.3 Existence Monitoring

Now we consider the EXISTENCEMONITORING problem to decide EXISTENCE at any point in time t. One can simply think of a protocol which applies the one-shot protocol (cf. Algorithm 1) at each time step. Consider T time steps, then this algorithm uses $\mathcal{O}(T)$ messages in expectation.

Interestingly, this upper bound is (asymptotically) tight, since it is complemented by a simple lower bound construction in which the output alternates between 0 and 1. Although this analysis is asymptotically tight, to compute a new solution from scratch in each time step does not seem to reflect the essence of the problem properly. Consider an instance which is (somehow) 'similar' between consecutive time steps: Let t be a time step and node i with $v_i^t = 1$ has sent a message during the execution of the EXISTENCEPROTOCOL. Assume in time step $t+1$ node i again observed $v_i^{t+1} = 1$. We exploit this fact by the applying the following simple idea:

> Consider a node i which observed at time t the value $v_i^t = 1$ and has sent a message to the server. Node i witnesses the output $\mathcal{F}(t)$ to be 1. We denote this node i to be the *representative*. As long as i observes $v_{t'} = 1$ for consecutive time steps $t' > t$, no message is sent.

An Improved Algorithm - Short Discussion Now we have an idea which reflects the intuition of exploiting similarities between consecutive time steps. However, a classical (worst-case) analysis still does not show any improvement. (In the worst-case, exactly those nodes i observe the value $v_i = 0$ at time $t+1$ after sending a message at time t. Even a restriction on the number of nodes that are allowed to change its value is not sufficient to reduce the upper bound.)

Before we apply competitive analysis to the (much more detailed) algorithm we like to develop a deeper understanding of the character of the problem and how this improves the communication bounds. Since if even only one node at a time changes its value, in other words an instance which is highly similar to the previous time step, does not improve the worst-case bound, consider the following:

We denote by s the *stability* the number of consecutive time steps in which a node observed the value $v = 1$. More formally, if the value changed from time step $t-1$ to time step t, then the value stays the same during the next s time steps, i.e., $v_t \neq v_{t-1}$, then $\forall t' \leq t+s : v_{t'} = v_t$. We can simply observe that an analysis based on such a

parameter leads to a simple improved upper bound of $\mathcal{O}(T/s)$ msg. in expectation.

On the basis of this observation we can precisely define the task for the algorithm: At a time t, find the node which observes the value 1 for the longest period of time. An algorithm which *knows the future in advance* can solve this task in an *optimal* way. In contrast to the absolute number of messages used, we express the number of messages used in relation to the costs of an *offline* algorithm (competitive analysis).

Formally, we express our bounds with respect to the number of *candidates* which could be chosen as the representative and the number of changes of these in comparison to an optimal offline algorithm which has the minimal number of such changes. Observe that if the adversary is strongly adaptive, i.e., it can choose the instance on the basis of the choice of the representative, the algorithm yields the same $\mathcal{O}(T)$ result, however expressed differently (as $\mathcal{O}(n)$, where n denotes the number of candidates). In case the adversary is oblivious, i.e., it generates the instance in advance, we show that the algorithm needs a logarithmic number of representatives more, compared to an optimal offline algorithm. Note that the competitiveness against an adaptive adversary is $\mathcal{O}(n)$.

Protocol Description Due to these observations we next formalize a parameter describing this behavior and provide a competitive analysis. To this end, we consider the number of differences in the sequences of nodes rep_{t-1} and rep_t and call this difference the *number of changes of representatives*. Let OPT denote the minimum possible number of changes of representatives (over all considered time steps T). The formal description of our algorithm is given in Algorithm 2. Roughly speaking, the algorithm defines phases, where a phase is defined as a maximal time interval during which there exists one node observing value $v = 1$ throughout the entire interval. Whenever a node being a representative changes its observation, it informs the server so that a new representative can be chosen (from those observing $v = 1$ throughout the entire phase, which is indicated by $active_i = true$). If no new representative is found this way, a new phase starts: all nodes observing $v = 1$ become active and one of these nodes is sampled uniformly at random.

Additionally, if no node observed the value $v = 1$ at time t, no representative is defined, but also no communication took place. Note that each node only stores two bits of information: one bit to indicate whether the node is active and one bit to indicate that node i is the representative.

Now we apply competitive analysis and compare the algorithm against a restricted offline algorithm, i.e., an offline algorithm that has to set filters. For the problem at hand this means the offline algorithm has to broadcast the representative rep_t of the

current time step t.

Algorithm 2 EXISTENCEMONITORING

Initialize with $t \coloneqq 0$, $\mathcal{F}_{t-1} \coloneqq 0$, $rep_{t-1} \coloneqq null$, $p_{t-1} \coloneqq 0$.
Each node i defines $active_i \coloneqq true$ if and only if $v_i^t = 1$.
(Rule for any $t' \geq t$: Each node with $v_i^{t'} = 0$ deactivates, i.e., defines $active_i \coloneqq false$.
If node i was the representative $rep_{t'-1} = i$, i sends a message to the server.)
repeat
  Server defines $t \coloneqq t + 1$, $\mathcal{F}_t \coloneqq \mathcal{F}_{t-1}$, $rep_t \coloneqq rep_{t-1}$, $p_t \coloneqq p_{t-1}$
  Each sensor node i observes a value v_i^t.
  if $\mathcal{F}_t = 0$ **then**
    Each node i defines $active_i = true$ if and only if $v_i^t = 1$.
    Server redefines rep_t by applying the EXISTENCEPROTOCOL to active nodes.
    if $rep_t \neq null$ **then** Server redefines output and phase $\mathcal{F}_t \coloneqq 1$, $p_t \coloneqq p_{t-1} + 1$.
  else
    if rep_t sends a message **then**
      Define rep_t by applying EXISTENCEPROTOCOL to all active nodes.
      if $rep_t = null$ **then**
        Server redefines output and phase $\mathcal{F}_t \coloneqq 0$, $p_t \coloneqq p_{t-1} + 1$
        Each node i defines $active_i = true$ if and only if $v_i = 1$.
        Server defines rep_t by applying the EXISTENCEPROTOCOL to active nodes.
        if $rep_t \neq null$ **then** Server redefines $\mathcal{F}_t \coloneqq 1$

Analysis In the following we show an upper bound on the competitiveness of the EXISTENCEMONITORING protocol as given in Algorithm 2. We shortly discuss some insights gained by the algorithm and its proof afterwards.

Theorem 3.3.1 (Competitiveness of EXISTENCEMONITORING)**.** EXISTENCEMONITORING *as described in Algorithm 2 is* $\mathcal{O}(\log n)$*-competitive against a filter-based offline algorithm with respect to an oblivious adversary.*

Proof. We consider one phase of the EXISTENCEMONITORING algorithm and show (i) that the offline algorithm has to send a message at least once and (ii) that the algorithm only sends $\mathcal{O}(\log n)$ messages on expectation. Let $\mathcal{P} = \{t_1, \ldots, t_m\}$ denote such a phase. For the sake of contradiction, assume to the contrary that the offline algorithm did not send any message during $\mathcal{P}$, i.e., defined a (valid) filter throughout $\mathcal{P}$.

[Case 1] First, consider the case that during the phase, no node observed a value $v = 1$, i.e., $\forall t \in \mathcal{P}, i \in \{1, \ldots, n\} : v_i^t = 0$. Note that both the offline and on-

line algorithms only send a broadcast message to inform each node that there is no representative. During the phase $\mathcal{P}$ no node sends further messages as given by the EXISTENCEMONITORING protocol since each node i defines $active_i$ to be false if it observes $v_i = 0$.

[Case 2] Now consider the case that during the phase the output is $\mathcal{F} = 1$.

To formally argue that the optimal offline algorithm has to communicate, we introduce some notation: Let $\mathcal{A}_{t_1,t'} \coloneqq \{i \mid \forall t_1 \leq t \leq t' : v_i^t = 1\}$ denote the set of nodes that observe the value $v_i = 1$ at each point in time t with $t_1 \leq t \leq t'$. The set $\mathcal{A}_{t_1,t_1}$ contains all nodes that observe $v = 1$ at time step t_1. Furthermore, $\mathcal{A}_{t_1,t'+1}$ can be defined inductively, i.e., $\mathcal{A}_{t_1,t+1} = \mathcal{A}_{t_1,t} \cap \{i \mid v_i^{t+1} = 1\}$. That is, from time step t to a consecutive time step $t+1$ the set $\mathcal{A}_{t_1,t+1}$ is a subset of $\mathcal{A}_{t_1,t}$. The phase $\mathcal{P}$ ends if and only if no representative can be found within the active nodes; i.e., if $\mathcal{A}_{t_1,t}$ is empty and this implies that there is no node which observed the same value in the time period $\{t_1, \ldots, t\}$. This fact leads to a contradiction to the assumption that the offline algorithm did not communicate as intended.

Next we analyze the expected cost of the EXISTENCEMONITORING protocol as given in Algorithm 2 during the considered phase $\mathcal{P}$. Let w.l.o.g. $\mathcal{A}_{t_1} \coloneqq \mathcal{A}_{t_1,t_1} = \{1, 2, \ldots, k\}$. With respect to the fixed phase, only nodes in $\mathcal{A}_{t_1}$ can communicate and the communication is bounded by the number of changes of the representative for $v = 1$ during the phase. Let t'_i be the first time after t_1 at which node i does not observe v. Let the nodes be sorted such that $i < j$ implies $t'_i \geq t'_j$. Let $r_1, \ldots, r_m$ be the nodes Algorithm 2 chooses as representatives in the considered phase. We show that $\mathbb{E}[m] = \mathcal{O}(\log k)$ holds. To this end, partition the set of time steps t'_i into groups G_i. Intuitively, G_i represents the time steps in which the nodes continuously observe value $v = 1$ since time t_1 and the size of the initial set of nodes that observed v is halved i times. Formally, G_i contains all time steps $t_{\ell_{i-1}+1}, \ldots, t_{\ell_i}$ (where $\ell_{-1} \coloneqq 0$ for convenience) such that ℓ_i is the largest integer fulfilling $|\mathcal{A}_{t_1,t'_{\ell_i}}| \in (k/2^{i+1}, k/2^i]$.

Let S_i be the number of changes of representatives in time steps belonging to G_i. We have $\mathbb{E}[m] = \sum_{i=0}^{\log k} \mathbb{E}[S_i]$. Consider a fixed S_i. Let $\mathcal{E}_j$ be the event that the j-th representative chosen in time steps belonging to G_i is the first one with an index in $\left\{1, \ldots, \lfloor \frac{k}{2^{i+1}} \rfloor\right\}$. Observe that as soon as this happens, the respective representative will be the last one chosen in a time step belonging to group G_i.

Now, since the algorithm chooses a new representative uniformly at random from the index set $\left\{1, \ldots, \lfloor \frac{k}{2^i} \rfloor\right\}$, the probability that it chooses a representative from the set $\left\{1, \ldots, \lfloor \frac{k}{2^{i+1}} \rfloor\right\}$ is at least $1/2$ except for the first representative of v, where it might be slightly smaller due to rounding errors. The event $\mathcal{E}_j$ occurs only if each of the first $j-1$ representatives were *not* chosen from this set, i.e., $\Pr[\mathcal{E}_j] \leq \left(\frac{1}{2}\right)^{j-2}$. Hence,

$\mathbb{E}[S_i] = \sum_j \mathbb{E}[S_i|\mathcal{E}_j] \cdot \Pr[\mathcal{E}_j] \leq \sum_j j \cdot (\frac{1}{2})^{j-2} = \sum_j \frac{j}{2^{j-2}} = \mathcal{O}(1).$ □

A Short Note. Now we have shown that the EXISTENCE problem can be monitored and its complexity is by a factor of $\mathcal{O}(\log n)$ larger in comparison to a filter-based offline algorithm with respect to a predefined instance.

Revisiting the analysis and especially the worst-case instance with respect to competitive analysis one can observe that this instance is highly 'similar' between consecutive time steps: i.e., only one node i changes its value from $v_i^t = 1$ to $v_i^{t+1} = 0$. Furthermore one considered phase $\mathcal{P}$ consists of n time steps which means that there is one node i which is from the beginning (at time step t_1) chosen by the offline algorithm as a representative which observed $v = 1$ throughout the whole phase $\mathcal{P}$.

3.4 Domain – One-Shot Computation

We extend the EXISTENCE problem from the previous section as follows: Identify each value v which is observed by at least one node; we denote this task by the DOMAIN problem. Recall the formal definition of Domain Monitoring (cf. Definition 2.2.2) to output at each time t the set $D_t = \{v_1, \ldots, v_n\} \subseteq \{1, \ldots, \Delta\}$; this one shot computation considers only a fixed time step t.

To solve this task, one might simply verify that the following simple change in the EXISTENCEPROTOCOL (cf. Algorithm 1) outputs the function correctly: Let each node i still be active as long as no node i' with $v_{i'} = v_i$ has sent a message before. As a consequence, for each value v which is observed by at least one node, there is a message sent to the server.

However, we present here a different approach to solve this task which is based on the following idea:

> Each node draws a random variable from a geometric distribution.
> For each value v: Those nodes with the (potentially same) largest random value inform the server about their value.

The formalized protocol directly translates the idea in a very nice way. The protocol is almost self describing, however a node can broadcast a message by sending a message to the server, which in turn sends a broadcast message to each sensor node.

Algorithm 3 DOMAINPROTOCOL

1. Each node i draws h_i from a geometric distribution with probability $p \coloneqq 1/2$.
2. As long as no node i' with $v_{i'} = v_i$ broadcasted before, node i broadcasts its value in round $r_i \coloneqq \max(\log n - h_i, 0)$.
3. The server outputs the union of all broadcasted values.

Analysis. Similar to the analysis of the EXISTENCEPROTOCOL, we shortly consider the number of rounds and the number of messages sent.

To bound the number of communication rounds, note that Line 2 of the protocol introduces rounds between 0 and $\log n$. Recall that the server is capable of processing up to n messages in a round, which directly implies:

Observation 3.4.1. *The* DOMAINPROTOCOL *as presented in Algorithm 3 uses at most* $\log n + 1$ *communication rounds.*

Now we consider the number of messages sent by the DOMAINPROTOCOL:

Theorem 3.4.2. *Applied for a fixed time* t*, the* DOMAINPROTOCOL *uses at most* $\mathcal{O}(|D_t|)$ *messages in expectation.*

Proof. Let X_i^v be a binary random variable indicating that node $i \in N_t^v$ sends a message to the server, and $X^v \coloneqq \sum_i X_i^v$. According to the algorithm node i sends a message if and only if its height h_i matches round $r_i = \max(\log n - h_i, 0)$ and no other sensor i' has sent its value before, i.e., $\forall i' : r_{i'} \geq r_i$.

$$\begin{aligned}\Pr\left[X_i^v = 1\right] &= \Pr\left[\exists r \in \{1, \ldots, \log n\} : h_i = r \wedge \forall i' \in N_t^v \setminus \{i\} : h_{i'} \leq r\right] \\ &\leq \sum_{r=1}^{\log n} \Pr[h_i = r] \cdot Pr[\forall i' \in N_t^v \setminus \{i\} : h_{i'} \leq r] \\ &\leq \sum_{r=1}^{\log n} \frac{1}{2^r} \left(1 - \frac{1}{2^r}\right)^{n^v - 1}.\end{aligned}$$

Since the random variables are binary, we can simply upper bound $\mathbb{E}[X]$ as follows:

$$\mathbb{E}[X] \leq n^v \cdot \sum_{r=1}^{\log n} \frac{1}{2^r} \left(1 - \frac{1}{2^r}\right)^{n^v - 1}.$$

Observing that $f(r) = n^v \cdot \frac{1}{2^r} \left(1 - \frac{1}{2^r}\right)^{n^v - 1}$ has only one extreme point and $f(r) \leq 2$ for all $r \in [0, \log(n)]$, we use the integral test for convergence to obtain

$$\begin{aligned}\mathbb{E}[X] &\leq n^v \cdot \sum_{r=1}^{\log n} \frac{1}{2^r} \left(1 - \frac{1}{2^r}\right)^{n^v - 1} \leq n^v \int_0^{\log n} \frac{1}{2^r} \left(1 - \frac{1}{2^r}\right)^{n^v - 1} \mathrm{dr} + 2 \\ &\leq \left[\frac{1}{\ln(2)} \left(1 - \frac{1}{2^r}\right)^{n^v}\right]_0^{\log n} + 2 \leq \frac{1}{\ln(2)} + 2 < 4.\end{aligned}$$

We apply the same argumentation independently for each value $v \in D_t$, concluding the proof. □

CHAPTER 4

EXACT & APPROXIMATE TOP-k MONITORING

In this chapter we consider problems regarding the k largest values at the current time step t and its monitoring variants for multiple time steps. We design and analyze Las Vegas algorithms which solve the problem with probability 1 and the number of messages used is on the basis of a random process.

Now after we have first insights handling monitoring problems (i.e., Existence and Domain) we extend the techniques used for one-shot and monitoring computations. Monitoring the Top-k(-Positions) yields a property which we can exploit to reduce communication, especially in comparison to monitoring the Sum: Only the subset (of size k) is needed to evaluate the output, whereas monitoring the Sum makes it necessary to transmit each update of any value (at least in the worst case).

In more detail, we consider the Top-k-**Position** Monitoring instead of monitoring the **values**. Monitoring the positions (node IDs) is a generalization of monitoring the values: The Server can simply probe each node (in the top k) by its ID and identify the top-k values using k messages.

Furthermore, comparing both by its output, we observe that the top-k value problem has to update the output whenever some value within the top-k is updated. However, if we only monitor the (set of) nodes observing the top-k values, the output is more 'resilient' against small updates. Especially if only the top-k values change, but not the set itself.

In this chapter we will further generalize the problem to monitor an approximation of the top-k positions. That is, if only small changes occur within the k-th largest values, e.g. the nodes observing the k-th and $k+1$-st largest values 'oscillate', large parts of these updates have to be transmitted when considering the exact version.

4.1 Introduction & Contribution

Problem	On., Off. Error	Competitiveness
Top-k-Position Monitoring	0, 0	$\mathcal{O}(k + \log n + \log \Delta)$
	0, 0	$\Omega(k + \log n + \log \Delta)$
Approx. Top-k-Pos. Mon.	ε, 0	$\mathcal{O}(k + \log n + \log\log \Delta + \log \frac{1}{\varepsilon})$
	ε, 0	$\Omega(k + \log n + \log \frac{1}{\varepsilon})$

The goal of this chapter is (among others) to design and analyze algorithms which monitor the Top-k-Positions exactly and with an allowed error of $\varepsilon > 0$ (described formally in Definition 2.2.4). We will show that the one-shot computation needs $\mathcal{O}(k + \log n)$, which is asymptotically tight (since we provide lower bounds), and show that monitoring needs $\mathcal{O}(\log \Delta)$ or $\mathcal{O}(\log\log \Delta + \log \frac{1}{\varepsilon})$ additional messages to monitor the Top-k in case the exact or approximated variant is considered.

Chapter Basis. Parts of the model, analysis, and results for the one-shot computation presented in the remainder of this chapter are based on the following publications:

- Mäcker, M., and Meyer auf der Heide. Online Top-k-Position Monitoring of Distributed Data Streams. In: *29th International Parallel and Distributed Processing Symposium (IPDPS, 2015)*, [MMM15].
- Mäcker, M., and Meyer auf der Heide. On Competitive Algorithms for Approximations of Top-k-Position Monitoring of Distributed Streams. In: *30th International Parallel and Distributed Processing Symposium (IPDPS, 2016)*, [MMM16].
- Biermeier, Feldkord, M., and Meyer auf der Heide. A Communication-Efficient Distributed Data Structure for Top-k and k-Select Queries. In: *15th International Workshop on Approximation and Online Algorithms (WAOA, 2017)*, [BFMM17].
- Feldkord, M., and Meyer auf der Heide. A Dynamic Distributed Data Structure for Top-k and k-Select Queries. In: *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, [FMM18].

Outline of the Chapter. We subdivide this chapter into four parts. First, we present the ideas for the protocols and discuss small variations of the problems. Second, we design a protocol which solves the Top-k problem for a single time step. Then, we present a protocol which monitors the Top-k and is competitive against an offline algorithm which is filter-based. We complement this protocol by lower bounds which are asymptotically tight. Finally, we allow the online algorithm to solve an approximation variant of the problem and compare the performance against an offline algorithm which is filter-based and still solves the exact problem.

4.2 Preliminaries

4.2.1 Top-k-Value Monitoring

The reader might ask why we consider the Top-k-*Position* Monitoring problem, which is the task to identify the IDs of the sensor nodes instead of considering the *values* within the Top-k. To answer this question we shortly consider the Top-k-Value Monitoring problem.

For the purpose of illustrating the effects we face with the monitoring aspect of the problem, we now assume that there is a protocol which we apply to identify the k largest values using $\mathcal{O}(\mathcal{C}_{\text{Top-}k})$ messages. Furthermore, we assume that all observed values are pairwise different.

Definition 4.2.1 (Top-k-Value Monitoring)**.** *At each time t let $v_1^t, \ldots, v_n^t$ be the values observed by the sensor nodes. Furthermore, denote by $s_1^t, \ldots, s_n^t$ be the* sorted *version of the observed values $v_1^t, \ldots v_n^t$ at time t, with $s_i^t \geq s_j^t$, iff $i < j$.*
The server has to output $s_1^t, \ldots, s_k^t$, $k \leq n$ at any point in time t.

Now, observe that without the restriction of filter-based algorithms (especially for the optimal offline-algorithm) there is no online algorithm with bounded competitiveness.

Observation 4.2.2. *Each online algorithm has an unbounded competitiveness against an optimal offline algorithm.*

For this simply consider an instance in which two sensor nodes observe two values in an alternating fashion. The optimal offline algorithm has to output the same (maximum) value all the time in comparison to the online algorithm which has to communicate at least one value at each time step to be correct.

Since this approach does not lead to any insight in the problem, we consider filter-based algorithms instead. Recall that a filter is defined as an interval of values which, as long as each node stays within its filter, the output need not change (and thus, no communication needs to take place). Applied to the Top-k-Value Monitoring problem a filter translates to the following definition:

Definition 4.2.3 (Top-k-Value Filter)**.** *For a fixed time t, an n-tuple of intervals forms a set of filters for the Top-k-Value Problem if and only if*

1. *for all $i \in \mathcal{F}(t)$ it holds $v_i \in F_i = [v_i, v_i]$, and*
2. *for all $j \notin \mathcal{F}(t)$ it holds $v_j \in F_j = [-\infty, s_k]$, where s_k denotes the k-th largest value.*

Protocol Description. Now the online algorithm boils down to the following very simple strategy: Identify the k largest values using the protocol given as assumed (using $\mathcal{O}(\mathcal{C}_{\text{Top-}k})$ messages). Then, it is sufficient to broadcast the k-th largest value v, such that all nodes i with $v_i \geq v$ define their filter to $F_i := [v_i, v_i]$ and the remaining nodes i with $v_i < v$ to $F_i := [-\infty, v]$.

Simply observe whenever a node within the k largest values observed a different value, a filter violation occurs such that the node sends a message to the server. To prevent that each node sends a message to the server, each node that observed a filter violation applies the protocol to identify the k-largest values. Consider the second case, a node observed a value which was not part of the output: The server is informed that there are new values within the Top-k.

An Analysis Sketch. We shortly argue on the competitiveness, however without giving a formal proof. It is easy to verify that the optimal offline algorithm can define the nodes filter using only one message, i.e., broadcasting the k-th largest value. Furthermore, observe that the offline algorithm needs to define new filters if one node within the Top-k observed a value which differs from the previous one, or a node which was not within the Top-k in the previous time step observes now a value within the Top-k leading to a filter violation.

On the other hand, the online algorithm needs until the next filter violation at most $\mathcal{O}(\mathcal{C}_{\text{Top-}k})$ messages. The given protocol is executed at most two times: once to initially identify the k largest values, and afterwards at most once if a filter violation occurs.

This short argumentation directly leads to a competitiveness of $\mathcal{O}(\mathcal{C}_{\text{Top-}k})$. To compare this result with the Top-k-Position Monitoring problem, we just apply the complexity of the protocol which we design and analyze in a later section. This protocol uses $\mathcal{O}(k + \log n)$ messages in expectation.

Proposition 4.2.4. *There is an online algorithm which monitors the Top-k-Values and is $\mathcal{O}(k + \log n)$-competitive against a filter-based offline algorithm.*

4.2.2 Exact Top-k-Position Monitoring

We now have some insight in monitoring the Top-k-Values. However, we have seen that each change of a value within the Top-k directly implies a message to the server and an update of the respective filter. We aim for a problem which still identifies the k largest values efficiently, but allows protocols which not necessarily transmit the whole update stream of the k largest values.

To this end consider the problem of (only) monitor the IDs of the nodes that observed the k largest values. As long as the set of nodes does not change it is easy to probe these nodes and identify the k largest values. We understand the Top-k-Position Monitoring as a generalization of the Top-k-Value Monitoring problem since it potentially reduces the amount of communication. However, this does not directly implies that the competitiveness is reduced. In fact we will show the opposite: Additionally to the costs of a one-shot computation, the (filter-based) offline algorithm can set filters better compared to the online algorithm.

Proposition 4.2.5. *Let each sensor node observe values from* $1, \ldots, \Delta$. *There is an online algorithm which monitors the Top-k-Positions and is* $\mathcal{O}(k + \log n + \log \Delta)$ *competitive compared to a filter-based offline algorithm.*

To prove this statement, we start with a protocol for the one-shot computation of the k largest values and the respective sensor nodes. We start by presenting a protocol which identifies the Maximum value and the ID in Section 4.3 using $\mathcal{O}(\log n)$ messages in expectation and with high probability. This can be used to identify the Top-k using $\mathcal{O}(k \cdot \log n)$ messages in expectation and with high probability. This result might be of independent interest.

Afterwards, we present a protocol which uses $\mathcal{O}(k + \log n)$ messages in expectation to identify the Top-k in Section 4.4. Using the EXISTENCEPROTOCOL and the algorithm for the one-shot problem, a protocol is presented which proves the claimed competitiveness of $\mathcal{O}(k + \log n + \log \Delta)$ in Section 4.5. Finally, we show a matching lower bound for restricted class of online algorithms in Section 4.6.

High-level Discussion: Maximum & Top-*k*. We now shortly discuss the ideas of the protocols and start with the MAXIMUM problem. The protocol presented in the next section is technically based on the EXISTENCEPROTOCOL.

> Let each node synchronously flip a coin with an initial success probability of $1/n$ and iteratively double the success probability. On a successful coin flip a message is sent to the server. A node does not take part in this process if a larger value is observed.

Note that the EXISTENCEPROTOCOL terminated as soon as there was at least one message. Here, we only let each nodes stop the process that observed a larger value. Furthermore it is easy to see that this process identifies the largest value and the respective node with probability 1 if executed to the last round.

To identify the Top-k, we might apply the MAXIMUMPROTOCOL multiple times to identify the k largest values – however, observe that a lot of communication is (not necessarily) used. Especially those nodes that send a message to identify the set of nodes that are 'near' to the maximum might send a message multiple times. The idea is to prevent this situation – one might think of 'going back' a few steps without starting the whole process from its beginning. This translates to a well known idea which we simulate here in a distributed fashion and without preprocessing:

> We build a simple search-tree-like structure based on a height the nodes drawn from a geometric distribution. Afterwards, a simple strategy comparable to an in-order tree walk is applied.

Note that this protocol always identifies the Top-k and technically this allows us to only consider one single random experiment, i.e., the expected number of nodes to identify the $k + 1$st largest value.

Monitoring of Top-k-Positions Using Filters One solution for this problem is to use the TOP-K-PROTOCOL at each time step. However, if the Top-k does not change this might be an unnecessary large amount of communication. We apply filters to identify changes to the input which need not be sent to the server.

Observe that for this problem it is sufficient to sent only a single value v which divides the Top-k from the remaining nodes, i.e., a value which is between the k-th and the $(k+1)$-st largest value. Based on this observation, the task for the online algorithm is to decide where to set the value v which divides the Top-k and the remaining sensor nodes from each other. Since no information about the future is known, and the adversary has no restriction in the process of generating the values that the sensor nodes observe in future time steps, the strategy is as follows:

> Initially identify the $(k+1)$-st largest values and the respective sensor nodes. As long as the Top-k-Positions do not change, define the filters in the middle between the k-th and the $(k+1)$-st largest value.

This strategy yields additional $\mathcal{O}(\log \Delta)$ messages in expectation (by applying the EXISTENCEPROTOCOL for identifying a filter violation). Note that this strategy only leads to a competitiveness of $\mathcal{O}(k + \log n + \log \Delta)$ if the observed values are integer values (with Δ the size of the universe). Furthermore, since the adversary is strongly adaptive, it is easy to see that every online algorithm needs at least $\Omega(\log \Delta)$ messages.

Note that this strategy successfully reduces communication each time the values at a new time step are within its filters or it is possible to find new filters; i.e., the Top-k set did not change. Intuitively speaking, filters can be seen as a certificate that witness the correctness of the output. If filters are violated, i.e., the certificate does not hold, the protocol tries to find a new certificate without computing the output new from scratch. In case this was successful, communication is reduced. As a consequence, if we only consider instances in which at each time step at least one node from the Top-k changes every time this strategy fails.

However, we will present two approaches to also tackle these situations: First, if a node switches its position with a node 'close' to the k-th largest value, we relax the problem to only monitor approximated Top-k-Positions which we discuss in the next section. On the other hand, if generally just a few nodes are change overall, the Top-k might change a lot, but the random structure to reduce the candidates significantly might be potentially reused. This is a completely different approach which we denote by *Maintaining Algorithms* and discuss in the second part of this thesis (cf. Chapter 7).

4.2.3 Discussion on Approx. Top-k-Position Monitoring

Now we shortly discuss on the motivation to consider this problem, the ideas of the protocols and aim for a deeper technical understanding where the bounds analyzed afterwards stem from.

Recall that monitoring the Approximate Top-k-Positions allows (only) the online algorithm to choose nodes as an output which are 'close' to the k-th largest value. As a direct consequence, filters are allowed to overlap. Intuitively speaking, the goal is to bring down the term of $\mathcal{O}(\log \Delta)$ in the competitiveness down to an $\mathcal{O}(\log 1/\varepsilon)$ by applying standard techniques from approximation algorithms. And in fact, if we introduce an additive error (say M), we claim that the competitiveness of $\mathcal{O}(k+\log n+\log \Delta)$ is reduced to $\mathcal{O}(k + \log n + \log(\Delta - M))$. (Where we again compare against an offline algorithm which solves the exact problem.)

However, with respect to a multiplicative error, we face the following situation: If filter violations are often observed from above, the allowed error shrinks smaller and smaller. To conclude, allowing the online algorithm to err, introduces the need to identify 'very roughly' at which point the offline algorithm defines its filters and apply the approximation strategy afterwards. Intuitively speaking, to identify this 'rough' value (more formally, the optimal value is approximated up to constant factors) the online algorithm needs additional $\mathcal{O}(\log \log \Delta)$ messages in expectation.

Proposition 4.2.6. *Let each sensor node observe values from* $1, \ldots, \Delta$. *There is a online algorithm which monitors the Approximate Top-k-Positions with a competitiveness of* $\mathcal{O}(k + \log n + \log \log \Delta + \log 1/\varepsilon)$ *compared to a filter-based offline algorithm which monitors the Top-k-Positions.*

We will show a very simple lower bound on the competitiveness of $\Omega(k + \log n + \log 1/\varepsilon)$. This does not match the upper bound up to the term of $\mathcal{O}(\log \log \Delta)$ which is introduced by this additional step of identifying a constant factor approximation of the result. The exact result for this situation, i.e., the competitiveness of an online algorithm which monitors the Approximated Top-k-Positions, in comparison to an offline algorithm which monitors the (Exact) Top-k-Positions remains open.

Next Steps We now start with the technical presentation of the protocols and their analysis. At first the MAXIMUMPROTOCOL is considered.

4.3 Maximum – One-Shot Computation

In this section we present and analyze a protocol to determine the maximum value currently observed by a set of nodes. This protocol is asked to solve the one-shot version of the Maximum Problem; i.e., we assume that the protocol is applied at a *fixed* time t. Hence, the values of the nodes do not change during a single execution.

Protocol Description. In order to determine the maximum value currently hold by n nodes the algorithm works as follows (cf. Algorithm 4): The algorithm proceeds in $\log N$ rounds, for a given $N > n$. At the beginning, i.e., before round $r = 1$, all nodes are set to be active. In each round r, each active node decides independently of the other nodes to send its value to the coordinator with a probability of $2^r/N$. After this, the coordinator broadcasts the largest value observed so far (in case it changed) and nodes having a value not larger than this maximum are deactivated, i.e., they no longer take part in the algorithm. The next round continues with all nodes still being active.

Algorithm 4 MAXIMUMPROTOCOL(N) $\triangleright N > n$

1. Each node i initially defines $active_i \coloneqq$ true and $max_0 \coloneqq -\infty$.
2. **For** round $r \coloneqq 1$ to $\log N$ **do**
3. **If** node i is active and $max_r > v_i$ **then**
4. $active_i \coloneqq$ false.
5. **If** node i is active and $max_r < v_i$ **then**
6. Node i flips a coin with success probability $p = \frac{2^r}{N}$
7. **If** coin flip is successful **then**
8. Node i sends (i, v_i) to the server.
9. $active_i \coloneqq$ false.
10. **If** $max_r > max_{r-1}$ **then**
11. Server broadcasts maximum max_r of all seen values.

Note that Algorithm 4 constitutes a Las Vegas algorithm that always computes the correct solution in $\log N$ steps, but the number of messages exchanged is described by a random variable which we denote by M. Furthermore, there may be rounds without communication; each node however is able to determine the current round r independently and without communication based on their synchronous clocks.

Analysis. To analyze the expectation of the random variable M, we first analyze the probability that a fixed node sends a message to the coordinator during one execution of MAXIMUMPROTOCOL. This is used to upper bound the expected number of messages. Afterwards, we show that this (asymptotic) bound holds with high probability. The bounds are shown to be asymptotically tight as complemented in Section 4.6.

Lemma 4.3.1. *Let M_i be a binary random variable indicating whether the sensor node $i \in \{1, \ldots, n\}$ sends a message during a run of Algorithm 4. Then,*

$$\Pr[M_i = 1 \big| r \in R_2] \leq \sum_{r \in R_2} \frac{2^r}{N} \cdot \left(1 - \frac{2^{r-1}}{N}\right)^i .$$

Proof. Let $S_{i,j}$ be a binary random variable indicating whether node i's coin flip is *successful* in round r. Let $A_{i,j}$ be a binary random variable indicating that node i is *active* in round j. A node sends a message in round j if and only if it is active in round j and its coin flip results in a success. Furthermore it deactivates itself after sending a message, so it is not possible that one node sends two or more messages. Hence,

$$\begin{aligned}
\Pr[M_i = 1 \big| r \in R_2] &= \sum_{r \in R_2} \Pr[S_{i,r} \textbf{ and } A_{i,r} = 1] \\
&= \sum_{r \in R_2} \Pr[S_{i,r} \big| A_{i,r} = 1] \cdot \Pr[A_{i,r} = 1] \\
&= \sum_{r \in R_2} \frac{2^r}{N} \cdot \Pr[A_{i,r} = 1] \ .
\end{aligned}$$

Observe that a node i is active at the beginning of round r if and only if it is active at the beginning of round $r-1$ and no node $i' \leq i$ sends a message in round $r-1$. Hence, for $r = 1$ we obtain $\Pr[A_{i,1} = 1] = 1$ and for $r > 1$:

$$\Pr[A_{i,r} = 1] = \Pr[\text{no } i' \leq i \text{ sends in round } r-1 \big| A_{i,r-1} = 1] \cdot \Pr[A_{i,r-1} = 1] \ .$$

In a fixed round r, the decisions of active nodes whether to send a message or not are independent. Additionally, $A_{i,r} = 1$ implies $A_{i',r} = 1 \ \forall 1 \leq i' \leq i$. Hence,

$$\Pr[\text{no } i' \leq i \text{ sends in round } r-1 \big| A_{i,r-1} = 1] \leq \left(1 - \frac{2^{r-1}}{N}\right)^i$$

$$\Pr[A_{i,r} = 1] \leq \prod_{r'=1}^{r-1} \left(1 - \frac{2^{r'}}{N}\right)^i \leq \left(1 - \frac{2^{r-1}}{N}\right)^i \ ,$$

By an inductive argument. Therefore the probability for $M_i = 1$ is bounded as claimed by the lemma. □

This result about the probability that a node sends a message can be used to determine an upper bound on the expected communication volume.

Theorem 4.3.2 (Upper Bound of MAXIMUMPROTOCOL)**.** *The expected number of messages sent in Algorithm 4 is at most* $4 \cdot \log n + 6$ *using* $\log N$ *number of rounds.*

Proof. [Communication] We upper bound the expected number of messages during the execution of Algorithm 4 by upper bounding the number of messages during rounds R_1 defined by the set of rounds $\{0, \ldots, \log N - \log n - 1\}$ and $R_2 = \{\log N - \log n, \ldots, \log N\}$ independently of each other.

$$\begin{aligned}
\mathbb{E}[M \big| r \in R_1] &\leq \sum_{i \in [n]} \mathbb{E}[M_i \big| r \in R_1] \leq \sum_{i \in [n]} \Pr[M_i \big| r \in R_1] \\
&\leq \sum_{i \in [n]} \sum_{r \in R_1} \Pr[S_{i,r} = 1 \wedge A_{i,r} = 1] \\
&\leq \sum_{i \in [n]} \sum_{r \in R_1} \frac{2^r}{N} \cdot 1 \leq \frac{n}{N} \sum_{r=1}^{\log N - \log n - 1} 2^r \leq \frac{n}{N}(2^{\log N - \log n}) = 1
\end{aligned}$$

By Lemma 4.3.1, we have $\mathbb{E}[M_i \big| r \in R_2] \leq \sum_{r \in R_2} \frac{2^r}{N} \cdot \left(1 - \frac{2^{r-1}}{N}\right)^i$. Hence,

$$\mathbb{E}[M \big| r \in R_2] \leq \sum_{i \in [n]} \sum_{r \in R_2} \frac{2^r}{N} \left(1 - \frac{2^{r-1}}{N}\right)^i \tag{4.1}$$

$$\leq \frac{1}{N} \sum_{r \in R_2} 2^r \sum_{i \in [n]} \left(1 - \frac{2^{r-1}}{N}\right)^i \tag{4.2}$$

$$\leq \frac{1}{N} \sum_{r \in R_2} 2^r \frac{1}{1 - \left(1 - \frac{2^{r-1}}{N}\right)} \tag{4.3}$$

$$= \frac{1}{N} \sum_{r \in R_2} 2^r \frac{1}{\frac{2^{r-1}}{N}} = \sum_{r \in R_2} 2 = 2 \cdot \log n + 2 \ , \tag{4.4}$$

proving the expected number of messages, where in Equation (4.3) we replaced the terms of the geometric series.

Since the expected number of messages sent by the sensor nodes is upper bounded by $2\log n + 3$ for all rounds $r \in R_1 \cup R_2$ and the server only sends a message if and only if a sensor has sent a value, the total number of messages is upper bounded by $4\log n + 6$ as claimed.

[Time] The number of rounds directly follow from Step 2, which defines to use a number of $\log N$ rounds. Since in each iteration at most one message may be sent by the server and at most one message is potentially sent by the node, the total amount of rounds follows as stated above. □

Theorem 4.3.3. *The number of messages sent in Algorithm 4 is at most $\mathcal{O}(\log n)$ with high probability, i.e., for any fixed $c > 1$ it is $\mathcal{O}(\log n)$ with probability at least $1 - \frac{1}{n^c}$.*

Proof. Note that although the random variables M_i are not independent, a variable M_i only depends on those M_j with $j < i$. Observe that the event of a node i to send a message, only decreases the probability of sending a message of another node. It is known that Chernoff bounds for the upper tail of a distribution can be applied to variables satisfying such a kind of negative correlation [SF13]. More precisely, we can apply such a Chernoff bound if for all $I \subseteq \{1, \ldots, n\}$ it holds

$$\Pr[\forall i \in I : M_i = 1] \leq \prod_{i \in I} \Pr[M_i = 1] \ .$$

Without loss of generality assume that $I = \{i_1, \ldots, i_\ell\}$ and $v_{i_1} \geq v_{i_2} \geq \ldots \geq v_{i_\ell}$. Then we have

$$\begin{aligned}
\Pr[\forall i \in I : M_i = 1] &= \Pr[M_{i_1} = 1 \wedge \ldots \wedge M_{i_\ell} = 1] \\
&= \Pr[M_{i_1} = 1 \big| M_{i_2} = 1 \wedge \ldots \wedge M_{i_\ell} = 1] \\
&\qquad \cdot \Pr[M_{i_2} = 1 \wedge \ldots \wedge M_{i_\ell} = 1] \\
&\leq \Pr[M_{i_1} = 1] \cdot \Pr[M_{i_2} = 1 \big| M_{i_3} = 1 \wedge \ldots \wedge M_{i_\ell} = 1] \\
&\qquad \cdot \Pr[M_{i_3} = 1 \wedge \ldots \wedge M_{i_\ell} = 1] \\
&\leq \Pr[M_{i_1} = 1] \cdot \Pr[M_{i_2} = 1] \cdot \ldots \\
&\qquad \vdots \\
&\leq \prod_{i \in I} \Pr[M_i = 1]
\end{aligned}$$

and thus, the M_i's are negatively correlated and we obtain the claimed result by applying a standard Chernoff bound. □

4.4 Top-K – One-Shot Computation

In this section we present an algorithm which identifies all k largest values currently observed by the sensor nodes. In this section, we denote by n the number of nodes that participate in the algorithm which is unknown to the algorithm and N a given upper bound for n. Note that by applying the MAXIMUMPROTOCOL k times, the problem can be solved using $\mathcal{O}(k \cdot \log(n))$ messages in expectation and $\mathcal{O}(k \cdot \log(N))$ rounds.

Here, we design and analyze an algorithm which improves both bounds and allows a trade-off between the number of messages and the number of rounds used. We denote by ϕ the trade-off parameter and will show upper bounds of $k + \frac{1-\phi}{\phi} \cdot \log_{1/\phi}(n) + \frac{1}{\phi}$ for the number of messages and $\mathcal{O}(\phi \cdot k + \log_{1/\phi}(N))$ for the number of rounds.

Protocol Description We apply the idea of identifying the k largest values in a distributed (non binary) search tree as discussed in Section 4.2. The protocol is given a parameter ϕ, where $\phi \in \left(0, \frac{1}{2}\right]$ holds. Furthermore there is a parameter h_{max} to define the maximal height of the distributed search tree. If only an upper bound of n (denoted by N) is known we apply the parameter $h_{max} \coloneqq \log_{1/\phi} N$ to the protocol.

Algorithm 5 TOP-KPROTOCOL (ϕ, h_{max})

Initialization()

1. Each node i draws a random variable h_i, i.i.d. from a geometric distribution with $p \coloneqq 1 - \phi$
2. Server defines $\ell \coloneqq -\infty$, $u \coloneqq \infty$, $h \coloneqq h_{max}$, $S \coloneqq \emptyset$
3. **Call** Top-k-Rec(ℓ, u, h)
4. Raise an **error**, if $|S| < k$

Top-k-Rec(ℓ, u, h)

1. **If** $h = 0$ **then**
2. **if** $|S| = k$ **then** return S,
3. **else** end recursion
4. Server probes sensor nodes i with $\ell < v_i < u$ and $h_i \geq h$
   Let $r_1 < \ldots < r_j$ be the responses
5. **Call** Top-k-Rec$(\ell, r_1, h-1)$
6. $S \leftarrow S \cup r_1$
7. **For** $i = 1$ to $j - 1$ do
8. **Call** Top-k-Rec$(r_i, r_{i+1}, h-1)$
9. $S \leftarrow S \cup r_{i+1}$
10. **Call** Top-k-Rec$(r_j, u, h-1)$

The algorithm starts by drawing a random variable h_i from a geometric distribution, i.e., $\Pr[h_i = h] = \phi^{h-1}(1-\phi)$. For the monitoring problem we will apply $\phi \coloneqq \frac{1}{2}$ (as defined in Corollary 4.4.5) which results in $\Pr[h_i = h] = 2^{-h}$. Observe that a smaller choice of the failure-probability ϕ results in smaller random heights h_i, but a larger expected number of siblings. To succeed an inorder treewalk the server identifies the siblings of a node with respect to the current path of the protocol by broadcasting values ℓ, u and h to identify all nodes i with values $\ell < v_i < u$ and a height of $h_i \geq h$.

Analysis In the following we show that the expected number of messages used by the TOP-KPROTOCOL is upper bounded by $k + \frac{1-\phi}{\phi} \cdot \log_{1/\phi}(n) + \frac{1}{\phi}$ in Theorem 4.4.3. Afterwards, an upper bound of $\mathcal{O}(\phi \cdot k + h_{max})$ on the number of communication rounds is presented in Lemma 4.4.4. Defining $\phi := \frac{1}{2}$ the bound on the communication translates to $k + \log(n) + 2$ in Corollary 4.4.5 and $\mathcal{O}(k + \log n)$ number of rounds. The communication bound is asymptotically tight as complemented by a simple lower bound of $\Omega(k + \log n)$ in Section 4.6.

We show an upper bound on the communication used by the TOP-KPROTOCOL analyzing the expected value of a mixed distribution: Intuitively speaking, consider the path from the root to the maximum in a non-binary searchtree. For each node i on the path consider the number of siblings j with a larger value, i.e., $v_j > v_i$. To bound the expected number of such siblings j, we first consider on a fixed height h the number tries G_h until the first node j' has drawn a height $h_{j'} > h$ (for each height h this results in the geometric-sequence, Definition 4.4.1). On the basis of G_h, we consider the number of nodes that have drawn precisely the height $h_{j'} = h$ (for each height h, the geocoin experiment Definition 4.4.2).

Note that this analysis turns out to be very simple since independence can be exploited in a restricted way and leads to a proper analysis with respect to small constants.

Definition 4.4.1. *We call a sequence $G = (G_1, \ldots, G_m)$ of m random experiments a* geometric-sequence *if each G_h is chosen from a geometric distribution with $p_h^{geo} := \phi^h$. We denote its $size(G) := \sum_h G_h$ and say it* covers *all nodes, if $size(G) \geq n$.*

For the analysis, we choose a fixed length of $m := \log_{1/\phi}(n)$ and modify G to $G' = (G_1, \ldots, G_{m-1}, n)$ such that G' covers all nodes with probability 1.

On the basis of a given geometric-sequence, we define a sequence describing the number of messages send by the nodes on a given height. We take the number of nodes G_j as a basis for a Bernoulli experiment where the success probability is the probability that a node sends a message on height h_j. This is $\Pr[h = h_j | h \leq h_j] = \frac{\phi^{h-1}(1-\phi)}{1-\phi^h}$.

Definition 4.4.2. *We denote a* geocoin experiment *by $C = (C_1, \ldots, C_m)$ a sequence of random variables C_h which are drawn from the binomial distribution $Binom(n = G_h, p_h^{bin} = \frac{\phi^{h-1}(1-\phi)}{1-\phi^h})$, i.e., C_h out of G_h successful coin tosses and each coin toss is successful with probability p_h^{bin}.*

Theorem 4.4.3. *Let $n > k$ and $h_{max} \geq \log_{1/\phi}(n)$ hold. The* TOP-KPROTOCOL *uses at most $k + \frac{1-\phi}{\phi} \log_{1/\phi}(n) + \frac{1}{\phi}$ messages in expectation.*

Proof. The probability to send a message of a node v within the Top-k is 1. It remains to show that the overhead is bounded by $\frac{1-\phi}{\phi} \log_{1/\phi}(n) + \frac{1}{\phi}$.

The number of messages sent by Algorithm 5 (excluding the k nodes observing the k largest values) is upper bounded by a geocoin experiment C.
Let $\mathcal{H} \coloneqq \log_{1/\phi}(n)$. For $h < \mathcal{H}$ we use that the geometric distribution is memory-less and hence

$$\mathbb{E}[C_h] = (1 - p_h^{geo}) \cdot (p_h^{bin} + \mathbb{E}[C_h]) = (1 - \phi^h) \cdot \left(\frac{\phi^{h-1}(1-\phi)}{1-\phi^h} + \mathbb{E}[C_i] \right) \ .$$

This can simply be rewritten as $\mathbb{E}[C_h] = \frac{1-\phi}{\phi}$.

For $h \geq \mathcal{H} = \log_{1/\phi}(n)$ we bound the number of messages by the total number of nodes with height at least $\mathcal{H}$. These can be described as the expectation of a Bernoulli experiment with n nodes and success probability $\phi^{\mathcal{H}-1}$ and hence we can bound $\mathbb{E}[C_{\geq \mathcal{H}}] \leq \phi^{\mathcal{H}-1} \cdot n = \frac{1}{\phi}$.

In total, we get

$$\sum_h \mathbb{E}[C_h] = \left(\sum_{h=1}^{\mathcal{H}-1} \mathbb{E}[C_i] \right) + \mathbb{E}[C_{\geq \mathcal{H}}] \leq \frac{1-\phi}{\phi} \log_{1/\phi}(n) + \frac{1}{\phi} \ ,$$

concluding the proof. □

Lemma 4.4.4. *The* TOP-KPROTOCOL *needs* $\mathcal{O}(\phi \cdot k + h_{max})$ *communication rounds in expectation.*

Proof. We structure the proof in two steps: First, we analyse the number of rounds used to determine the maximum, and second, the number of communication rounds used to determine the Top-k.

Observe that the algorithm uses a linear amount of steps (linear in h_{max}), until it reaches $h = 1$, after which the maximum is found. Afterwards, in each step the algorithm recursively probes for nodes successively smaller than the currently largest values that are added to the output set S. Note that by the analysis in Theorem 4.4.3, the number of nodes that send a message in expectation in each round is $(1-\phi)/\phi$ (for $h < \log_{1/\phi}(n)$). Thus, in each communication round there are $\Omega(\frac{1}{\phi})$ nodes in expectation that send a message, such that after an expected number of $\mathcal{O}(\phi \cdot k)$ rounds the TOP-KPROTOCOL terminates. □

Corollary 4.4.5. *For* $N = n$, $\phi \coloneqq \frac{1}{2}$, *and* $h_{max} \coloneqq \log(n)$, *the* TOP-KPROTOCOL *uses an amount of* $k + \log(n) + 2$ *number of messages in expectation and* $\mathcal{O}(k + \log(n))$ *communication rounds.*

4.5 Exact Top-k-Position Monitoring

In this section we describe our algorithm for monitoring the IDs of the nodes holding the k largest values. Assuming that we have distributed protocols to solve the following problems: Compute the k largest values and the respective nodes which observed these values (Section 4.4) using $k + \log n$ messages in expectation. Second, identify a filter violation only using $\mathcal{O}(1)$ messages in expectation (Section 3.2).

Protocol Description. Recall that the key idea is to use filters to prevent that nodes communicate changes to the coordinator that are not essential for the computation of the Top-k. In case filters are violated, the EXISTENCEPROTOCOL (cf. Algorithm 1) is applied to identify this. The strategy to update the filters is as follows:

Initially, filters are set to be $[-\infty, mid]$ or $[mid, \infty]$, respectively, where mid is determined by the coordinator. The coordinator applies the TOP-KPROTOCOL and identifies the k-th and $(k+1)$-st largest value. On the basis of these values mid is defined as the midpoint of v_k and v_{k+1}. Now consider a time t at which nodes violate their filters. (Note that the coordinator is informed that *some* filter violation has happened.) The server determines new intervals and broadcasts these to the nodes. This might incur new filter violations. Either the protocol finds new intervals (within the same phase) which do not lead to filter violations, or a new phase begins.

Algorithm 6 TOP-k-POSITION MONITORING

[New phase p at time step t_0]

1. Determine $k+1$ largest values and the respective nodes using the TOP-KPROTOCOL.
2. Server defines $M := [v_{k+1}, v_k]$ and mid the midpoint of M.
3. Server broadcasts mid. Each node i with $v_i > mid$ defines its filter $F_i := [mid, \infty]$ and each i with $v_i \leq mid$ defines $F_i := [-\infty, mid]$, respectively.
4. Each node i defines a boolean flag $top_i = 1$, iff $v_i > mid$ holds.

[Same phase p, filter violation]

1. **if** filter violation from *below* (i.e., $top_i = 0$) with value v_i
2. **then** Server redefines $M := M \cap [v_i, \infty]$
3. **else** Server redefines $M := M \cap [-\infty, v_i]$
4. **If** $M = \emptyset$ **then** Server starts a new phase.
5. **else** Server broadcasts mid, the midpoint of (updated) M.
6. Each node i updates its value according the rules above.

[Same phase p, no filter violation] (No communication takes place)

Analysis. In order to prove the competitiveness of TOP-k-POSITION MONITORING as presented in Algorithm 6, we first define the minimum value and the maximum value with respect to a set of nodes and a period of time steps in Definition 4.5.1. Afterwards, we show that there has to be value v which separates the Top-k and the remaining nodes during a period of time at which no communication does not need to take place in Lemma 4.5.2. On the basis of this observation we show the main result, the upper bound on the competitiveness of $\mathcal{O}(k + \log n + \log \Delta)$ as given in Theorem 4.5.3. As a short remark we consider the number of rounds and show that the protocol does not violate the demanded bound, i.e., to only use a polylogarithmic number of rounds in expectation, in Observation 4.5.4.

Definition 4.5.1. *Let t_0, t be given time steps with $t \geq t_0$. We denote the Top-k during $[t_0, t]$ by $\mathcal{F}_1 := \mathcal{F}(t')$ with $t' \in [t_0, t]$ assuming $\mathcal{F}(t')$ is constant during $[t_0, t]$. (Let $\mathcal{F}_2 := [n] \setminus \mathcal{F}_1$ respectively.) We denote the maximum over all values observed by nodes $i \in \mathcal{F}_2$ during $[t_0, t]$ by $\text{MAX}_{\mathcal{F}_2}(t_0, t) = \max_{t_0 \leq t' \leq t} \max_{i \in \{k+1,\dots,n\}}(v_i^{t'})$. Consequently, let $\text{MIN}_{\mathcal{F}_1}(t_0, t)$ be the minimum over all values observed by nodes $i \in \mathcal{F}_1$. We omit the parameters, if they are clear from the context.*

Lemma 4.5.2. *Consider time steps $[t_0, t]$ and some arbitrary time step $t' \in [t_0, t]$. If OPT uses the same set of filters throughout $[t_0, t]$, the minimum over all nodes $i \in \mathcal{F}(t')$ is greater or equal the maximum over all nodes $i \notin \mathcal{F}(t')$, i.e., $\text{MIN}_{\mathcal{F}_1} \geq \text{MAX}_{\mathcal{F}_2}$.*

Proof. Proof by contradiction. Assume OPT uses the same set of filters throughout $[t_0, t]$, but $\text{MAX}_{\mathcal{F}_2} > \text{MIN}_{\mathcal{F}_1}$ holds. Then there are two nodes, $i \in \mathcal{F}_1$ and $j \in \mathcal{F}_2$, and two times $t_1, t_2 \in [t_0, t]$, such that $v_i^{t_1} = \text{MIN}_{\mathcal{F}_1}$ and $v_j^{t_2} = \text{MAX}_{\mathcal{F}_2}$. Due to the definition of a set of filters and the fact that OPT has not communicated during $[t_0, t]$, OPT must have set the filter for node $i \in \mathcal{F}_1$ to $[s_1, \infty]$, $s_1 \leq v_i^{t_1}$, and for node $j \in \mathcal{F}_2$ to $[-\infty, s_2]$, $s_2 \geq v_j^{t_2}$. This is a contradiction to the definition of a set of filters (applied for this problem in Lemma 2.3.4). □

As direct implications of the previous simple observations a filter-based algorithm only needs to define two different intervals of values. One interval to let all nodes that are part of the output, i.e., are in the Top-k, and one interval for the remaining nodes. The second observation that can be made is that no change in the output can occur without violating the filter. Another consequence will be used to argue that there is no further choice of a filter during a (consecutive) interval of time, i.e., if $\text{MAX}_{\mathcal{F}_2} > \text{MIN}_{\mathcal{F}_1}$. The considerations above lead to the following theorem.

Theorem 4.5.3. *Let* $\Delta := max_t(v_k^t - v_{k+1}^t)$. *Then the* TOP-$k$-POSITION MONITORING *as presented in Algorithm 6 is* $\mathcal{O}(k + \log n + \log \Delta)$*-competitive against a centralized filter-based offline algorithm.*

Proof. We split the time into intervals on the basis of the time steps $t_1, t_2, \ldots$ at which OPT communicates. Consider the case that for a given time step t_i a time step t_{i+1} exists at which OPT communicates. Fix an arbitrary interval $I := [t_i, t_{i+1})$ for $i \geq 1$. Since OPT does not communicate during $I' := (t_i, t_{i+1})$, the Top-k does not change and there is a set $\mathcal{V}^*$ of values which separate $\mathcal{F}_1$ and $\mathcal{F}_2$ during I. At the end of I, the protocol has chosen one value $mid \in \mathcal{V}^*$ to define the nodes filter (otherwise a filter violation would have occurred and the algorithm would have updated the filters). This implies that the algorithm computes the output during this time interval only once, i.e., the TOP-KPROTOCOL is called only once. And since the phase restarts at most once, at most $\mathcal{O}(\log \Delta)$ filter violations have to be processed. Each uses $\mathcal{O}(1)$ messages in expectation to be identified and thus, the number of messages as claimed follows.

Now consider the second case in which OPT did not communicate after time step t_i. Since OPT did not communicate, the Top-k does not change at any time $t' \geq t_i$. Furthermore, and on the basis of Lemma 4.5.2, there is a set of values $\mathcal{V}$ which separate $\mathcal{F}_1$ and $\mathcal{F}_2$: For each $v \in \mathcal{V}$ it holds $\text{MIN}_{\mathcal{F}_1} \geq v \geq \text{MAX}_{\mathcal{F}_2}$. Such a value v is chosen by the algorithm as a value mid after at most $\log \Delta$ handled filter violations. □

The previous observations show that only two intervals are sufficient to define the filters for the nodes. In detail it is sufficient for an algorithm to broadcast only the value mid to define the interval $[-\infty, mid]$ for each node $i \notin \mathcal{F}(t)$ and $[mid, \infty]$ for those nodes $i \in \mathcal{F}(t)$. That is, an optimal filter-based offline algorithm only needs to send *one* message to define the filters.

Observation 4.5.4. *The* TOP-k-POSITION MONITORING *protocol uses an amount of* $\mathcal{O}(k + \log(\Delta + n))$ *number of rounds per time step.*

Proof. First, recall that one execution of the protocol EXISTENCEPROTOCOL needs precisely $\log n$ number of rounds. This protocol is applied to reliably identify a filter violation. However, since the algorithm is only informed about *some* filter violation the updated filters may incur new filter violations. However, after $\mathcal{O}(\log \Delta)$ filter updates, a new phase starts, using $\mathcal{O}(k + \log n)$ number of rounds in expectation. □

Note that a worst-case instance (which maximizes the competitive ratio) would present only one filter violation at a time-step. For such an instance the algorithm needs $\mathcal{O}(\frac{k + \log n \cdot \log \Delta}{\log \Delta})$ (amortized) number of rounds per time-step.

4.6 Lower Bound

We show that the given bounds of computing the Maximum or the Top-k are optimal (up to constant factors). We show that if the observed values at a time t are pairwise distinct, then any comparison-based randomized algorithm needs a number of messages that is at least logarithmic, i.e., $\mathbb{E}[X] = \Omega(\log n)$. By following Yao's minimax principle, it is sufficient to show that, given a probability distribution on the inputs, any deterministic algorithm has to send $\Omega(\log n)$ messages in expectation.

Theorem 4.6.1 (Lower Bound)**.** *Every comparison-based randomized algorithm requires $\Omega(\log n)$ messages on expectation to compute the maximum in our model.*

Proof. Let the inputs be distributed according to a distribution P in such a way that each instance where each of the numbers $\{1, \ldots, n\}$ is assigned to exactly one node v_i is chosen with probability $(1/n!)$. Consider any deterministic algorithm A that calculates the maximum in our model. Since we are looking for a lower bound on the number of messages, we see that A can basically not do better than having a fixed sequence $(s_1, \ldots, s_n)$ of nodes that it probes consecutively in this ordering, skipping nodes that have values smaller than the maximum value observed so far.

By the following simple argument, we can show the desired result: Assume for the moment that A probes *all* nodes, i.e., A receives a randomly chosen permutation of the values $\{1, \ldots, n\}$. The course of the algorithm can be (partly) described by gradually constructing a binary search tree of the observed values. Now consider the actual course of the algorithm: A skips nodes which cannot deliver new information about the maximum value. Then the course of the algorithm is nothing else but the path in the binary search tree from the root to the node holding the maximum value. It is known (e.g. [SF13]) that in expectation this path has a length of $\Theta(\log n)$, proving the theorem. □

We extend the lower bound to the number k of values that are to be determined. Simply observe that k constitutes the size of the output (in terms of machine words) and since each sensor node observes only one value at a time step, also k messages have to be sent.

Corollary 4.6.2. *$\Omega(k + \log n)$ messages in expectation to determine the k largest values in our model.*

4.7 Allow the Online Algorithm to Err

In this section, we analyze the competitiveness of the online algorithm which is asked to compute an relaxation of the Top-k-Positions and compare this against an offline algorithm which solves the exact problem. The main result in this subsection is an upper bound on the competitiveness of $\mathcal{O}(k + \log n + \log\log\Delta + \log\frac{1}{\varepsilon})$. Note that this bound is tight up to an additive $\log\log\Delta$ term as simply shown in Section 4.8.

Short Overview. Consider the approach from the past section: The protocol uses an interval to choose a value with the purpose to a) separate the Top-k from the remaining nodes and b) define filters. This interval is denoted by $M = [\ell, u]$ and has the invariant that OPT had to choose a value mid^* within this interval, i.e., $mid^* \subseteq M$. If for a phase p this M is empty, OPT had to communicate at least once.

Within this framework, the strategy of the previous section was to always choose the midpoint of M to define Filters. This is one of three strategies of algorithm APRXTOP-K, which additionally uses the following strategies: First, if ℓ and u are ε-close, then a filter is chosen, which *overlaps* this gap. And second, the midpoint-strategy is extended by a preceding strategy which chooses the midpoint on a *logarithmic scale*. We describe the approach below in (much) more details.

Algorithm Description. We propose an algorithm started at t that computes the output set $\mathcal{F}_1 \coloneqq \mathcal{F}(t)$ using the TOP-KPROTOCOL and for all consecutive times tries to find Filters that preserve this output. If no Filter can be found, a new phase starts.

The time steps $t_0, t_1, \ldots$ at which the algorithm is executed are split into phases $p_0, p_1, \ldots$. Each phase is split into at most three *subphases*. A subphase is defined as the time steps of a phase p at which the same rule can be executed by the protocol.

The algorithm tries to find a value m which partitions $\mathcal{F}_1$ from $\mathcal{F}_2$, such that for all nodes $i \in \mathcal{F}_1$ it holds $v_i \geq m$ and for all nodes $i \in \mathcal{F}_2$ it holds $v_i \leq m$. Thus, only two different filters that are basically defined by such one value. Whenever a filter violation is reported, this value is recalculated and used to set filters properly.

The approach proceeds in rounds. In the first round we define an initial interval M_0. In the r-th round, on the basis of interval M_r, we compute a value m that is broadcasted and is used to set the filters to $[0, m]$ and $[m, \infty]$. As soon as node i reports a filter violation observing the value v_i, the coordinator redefines the interval $M_{r+1} \coloneqq M_r \cap [-\infty, v_i]$ if the violation is from above and $M_{r+1} \coloneqq M_r \cap [v_i, \infty]$ otherwise. For the sake of readability, and since at a round r there is no need to keep the information about round $r' < r$, we omit the parameter to identify its round.

Algorithm 7 APRXTOP-K

	Rule R1	Rule R2	Rule R3
Condition	$\log \ell + 2 \leq \log u$	$\log \ell + 2 > \log u$ $\wedge\, \ell < (1-\varepsilon)u$	$\ell \geq (1-\varepsilon)u$ $\wedge\, \ell \leq u$
Def. m	midpoint of $M = \left[\lceil \log \ell \rceil, \lfloor \log u \rfloor\right]$	midpoint of $M = [\ell, u]$	/
Def. Filters	$F_1 \coloneqq [2^m, \infty]$ $F_2 \coloneqq [-\infty, 2^m]$	$F_1 \coloneqq [m, \infty]$, $F_2 \coloneqq [-\infty, m]$	$F_1 \coloneqq [\ell, \infty]$, $F_2 \coloneqq [-\infty, u]$

[Initialize at time t, new phase p]

1. Compute the nodes holding the $(k+1)$ largest values.
2. Define $\mathcal{F}(t)$, $\ell \coloneqq v_{k+1}^t$, $u \coloneqq v_k^t$ and $M \coloneqq [\ell, u]$.
3. **If** the condition for Ri holds
4. **then** apply rule Ri (i.e., define m and broadcast new filters F_1, F_2).
5. **else** (i.e., no condition holds, $\ell > u$) Start a new phase $p+1$, call Initialize.

[Filter-violation by node i at time t, in phase p]

1. **If** the filter violation is reported from below
2. **then** $\ell \coloneqq v_i^t$ **else** $u \coloneqq v_i^t$.
3. **If** the condition for Ri holds
4. **then** apply rule Ri (i.e., define m and broadcast new filters F_1, F_2).
5. **else** (i.e., no condition holds, $\ell > u$) Start a new phase $p+1$, call Initialize.

Note that in one phase it is not necessary that all subphases have to be proceeded. Due to this fact, the algorithm applies a *rule* on the basis of the current situation (i.e., on the basis of u and ℓ). If no rule can be applied, it holds $\ell > u$ and a new phase starts. We now consider the subphases in more detail:

R1 The first subphase is applied if $\log \ell + 2 > \log u$ holds, i.e., ℓ and u differ more than a (predefined) constant factor. A midpoint-strategy is applied on a logarithmic scale, i.e., the midpoint of $\lceil \log \ell \rceil$ and $\lfloor \log u \rfloor$ is chosen as the value m. To separate $\mathcal{F}_1$ from $\mathcal{F}_2$ the value $v \coloneqq 2^m$ is broadcasted to the nodes. The subphase terminates if $\log \ell + 2 \leq \log u$ holds, i.e., $u = \Theta(\ell)$.

R2 During the second subphase it holds $\log \ell + 2 \leq \log u$ and $\ell \leq (1-\varepsilon)u$. Now the midpoint strategy is applied (in the same way as in the previous section for monitoring the exact Top-k-Positions). However, the strategy is interrupted at the time step at which ℓ and u differ by a factor of ε factor.

R3 The final subphase is started, in case $\ell \geq (1-\varepsilon)u$ holds. The filters are defined as $F_2 \coloneqq [-\infty, u]$ and $F_1 \coloneqq [\ell, \infty]$, i.e., they overlap in at most εu values. Note that this overlap meets the definition of filters for the Approximate Top-k-Position Monitoring problem. The current phase stops with the next filter violation.

Analysis We analyze the protocol as given in Algorithm 7 in the following steps: First, we shortly analyze the correctness of the algorithm; i.e., we show that the output is correct at each time step and that the defined filters are correct in Observation 4.7.1. Second, we upper bound the number of messages used in each phase p in Lemma 4.7.3. Third, we show our main result: an upper bound on the competitiveness, where we compare the protocol against an adversary who solves the exact problem in Theorem 4.7.4. Finally, (as a side-remark) we consider the number of rounds and show that this can be upper bounded by a polylogarithmic term in Observation 4.7.5.

We start with the first step and observe its correctness. Simply observe that for a fixed phase p if rule R1 or R2 are applied filters overlap only in a single value, i.e., constitute Top-k Filters, which are more restricted than the Approximate Top-k Filters. Thus, in these time steps the output is even a correct output for monitoring the Top-k exactly. If rule R3 is applied the filters overlap at most by εu which meets the definition of an Approximate Top-k Filter and implies the correctness of the output.

Observation 4.7.1. *Consider a time step t_0 and a phase p which starts at t_0 and ends at time t' which is defined by* APRXTOP-K. *The protocol computes and monitors the Top-k approximately for a given $\varepsilon > 0$ during p.*

Now we argue that the number of messages during a phase p is upper bounded by $\mathcal{O}(k + \log n + \log\log\Delta + \log\frac{1}{\varepsilon})$ in expectation. To this end, we upper bound the number of messages in each subphase (i.e., each time step of a phase p in which the same rule is applied).

Consider the first subphase (all time steps of a fixed phase p in which rule R1 is applied):

Lemma 4.7.2. *Let t be a given time step, assume $\mathcal{F}(t)$ the output determined by the protocol. Let p_1 the interval of (consecutive) time steps in which the rule R1 holds and the output does not change. The protocol uses at most $\mathcal{O}(\log\log\Delta)$ messages in expectation (during p_1).*

Proof. Consider an arbitrary, fixed phase p_1 consisting of consecutive time steps $p_1 = \{t_1, \ldots, t'\}$ in which rule p_1 always holds. Note that the server defines (and broadcasts) the value m to be the midpoint of the interval $\lceil\log\ell\rceil$ and $\lfloor\log u\rfloor$. In other words, a midpoint-strategy on a logarithmic scale is applied. To analyze the amount of messages needed and express it in terms of Δ, observe that in the worst case $u = \Delta$ and $\ell = 0$ holds. Now observe that after $\mathcal{O}(\log\log\Delta)$ filter violations (no matter from below or from above) the condition of rule R1 does not hold any longer. Each filter violation

uses at most $\mathcal{O}(1)$ messages in expectation by applying the EXISTENCEPROTOCOL as presented in Algorithm 1. □

By the same arguments we can show that the second subphase needs $\mathcal{O}(\log 1/\varepsilon)$ and the third subphase needs $\mathcal{O}(1)$ messages in expectation. We conclude:

Lemma 4.7.3. *Consider a phase p. The algorithm uses a number of messages upper bounded by $\mathcal{O}(k + \log n + \log\log\Delta + \log\frac{1}{\varepsilon})$ in expectation throughout p.*

Proof. To argue on the number of messages, observe that the initialization can be executed using $\mathcal{O}(k + \log n)$ number of messages in expectation by applying the Top-k protocol as given in Algorithm 5. At the time the condition of the rules are checked these steps can be performed without any additional communication since the server keeps track of u and ℓ. Each subphase is executed at most once and thus, each number of messages from the subphases is added to the overall costs per phase p at most once. If no rule can be applied, no further subphase is called and no further message is being sent. A new phase p' is started concluding the analysis for the subphase p. □

We have shown the correctness of the online algorithm and the number of messages used during a phase. Now we can combine these results to the result on the competitiveness:

Theorem 4.7.4. *The algorithm* APRXTOP-K *has a competitiveness which is upper bounded by $\mathcal{O}(k + \log n + \log\log\Delta + \log\frac{1}{\varepsilon})$ allowing an error of $\varepsilon > 0$ compared to an optimal offline algorithm that solves the exact Top-k-Position Monitoring problem.*

Proof. Fix a phase $p = \{t, \ldots, t'\}$. The number of messages of APRXTOP-K follow from Lemma 4.7.3.

Now we argue that OPT had to communicate at least once in the interval $[t, t' + 1]$ during which APRXTOP-K was applied.

If OPT communicated, the bound on the competitiveness directly follows. Now assume that OPT did not communicate in the interval $[t, t' + 1]$. We claim that the interval $L = [\ell, u]$ which is maintained during APRXTOP-K always satisfies the invariant $L^* \subseteq L$. If this claim is true, we directly obtain a contradiction to the fact that OPT did not communicate because of the following reasons. On the one hand, because OPT has to monitor the exact Top-k-Positions, OPT chooses the same set of nodes $\mathcal{F}^* = \mathcal{F}_1 = \mathcal{F}(t)$ which was chosen by the online algorithm. On the other hand, at the time $t' + 1$ the algorithm APRXTOP-K starts a new phase, and thus, $u < \ell$ holds. Thus, the interval L is empty and since $L^* \subseteq L$ holds, it follows that L^* is empty and hence, OPT must have communicated.

We now prove the claim. Recall that APRXTOP-K starts a new phase with ℓ and u defining the interval L such that $L^* \subseteq L$ holds by definition. To show that $L^* \subseteq L$ holds during the entire interval $[t, t'+1]$, it suffices to argue that each change of u or ℓ also has to hold for L^*:

Consider the cases in which filter violations are observed and hence the interval L is modified: If a filter violation from below happened at a time t'', there is a node $i \in \mathcal{F}_2$ with a value $v_i^{t''} = \ell^{t''} > \ell^{t''-1}$ and thus, $\ell^* > v_i^{t''}$ holds. (Analogously for filter violation from above.) This case distinction leads to the result, that L^* has to be a subset of $L = [\ell, u]$. □

Finally, and as a side remark we consider the number of communication rounds the algorithm needs per time step. Recall that the protocol is allowed to use a polylogarithmic number of rounds at each time step which can be shown easily (assuming k is polylogarithmic): The Top-k protocol and the EXISTENCEPROTOCOL only use polylogarithmic number of rounds and are applied only a polylogarithmic number of rounds.

Observation 4.7.5. *The protocol* APRXTOP-K *as presented in Algorithm 7 uses a polylogarthmic number of rounds per time step.*

4.8 Lower Bounds for the Approx. Top-k-Monitoring Problem

In this section we present a simple lower bound to show the tightness of the result in the previous section. We show that the result is tight up to an additive term of $\mathcal{O}(\log \log n)$.

Theorem 4.8.1. *Let $\mathcal{A}$ be a comparison-based online algorithm which solves the approximate Top-k problem. Then $\mathcal{A}$ has a competitiveness against an offline algorithm of at least $\Omega(k + \log n + \log \frac{1}{\varepsilon})$.*

Proof. Note that the output has to be computed by any algorithm $\mathcal{A}$ and thus, the same argument holds as in Corollary 4.6.2 proving a lower bound of $\Omega(k + \log n)$.

It remains to show a lower bound of $\Omega(\log 1/\varepsilon)$. For this simply consider two sensor nodes with observed values $v_1 := \Delta$ and $v_2 := \Delta/2$. The instance is generated such that the sensor node with the smallest difference to the filter border (as defined by algorithm $\mathcal{A}$) moves towards the boarder and yields a filter violation. □

Chapter 5

Top-k-Position Monitoring against an Approximate Offline Algorithm

In this chapter, we study a variant in which the optimal offline algorithm is allowed to introduce an error; i.e., we monitor the Top-k-Positions approximately. It turns out that it is much more challenging for online than for offline algorithms to cope with or exploit the allowed error in the output.

This fact is formalized in a lower bound of $\Omega(n)$ (for constant k), which is much larger than previous upper bounds of $\mathcal{O}(k + \log n + \log \Delta)$ for the exact problem. However, we also propose two online algorithms that are competitive against offline algorithms. One online algorithm, is allowed to make use of the same error ε as the offline algorithm, which results in a competitiveness of $\mathcal{O}(n^2 \log \Delta)$ (assuming reasonable relations between n and Δ to simplify the bounds). Furthermore, an augmented version which allows the online algorithm an error of 2ε compared to ε the offline algorithm uses with the result of $\mathcal{O}(n)$-competitiveness (again with reasonable assumptions on n and ε and Δ).

5.1 Introduction & Contribution

Problem	Err. On.	Err. Off.	Competitiveness
Approx. Top-k Position	ε_{ON}	ε_{OFF}	$\Omega(n/k)$
	ε	ε	$\mathcal{O}(n^2 \log \Delta)$
	2ε	ε	$\mathcal{O}(n)$

In this chapter we concentrate on the problem of monitoring Approximate Top-k-Positions. We start with a very simple construction to show a lower bound of $\mathcal{O}(\frac{n}{k})$ on the competitiveness of any (potentially randomized) online algorithm and arbitrary allowed errors $\varepsilon_{ON}, \varepsilon_{OFF} \geq \frac{1}{\Delta}$. We show that there exists an online algorithm which admits bounded competitiveness of $\mathcal{O}(n^2 \log \Delta)$. Furthermore, if the online algorithm is allowed to use twice the error of the offline algorithm, the competitiveness is only $\mathcal{O}(n)$ which is asymptotically tight for the cases in which $k = \mathcal{O}(1)$ and $\Delta = \mathcal{O}(2^n)$.

A Short High-level Discussion of the Problem. The inclined reader might ask why this problem is essentially different in comparison to the previous section although the problem is almost the same.

In fact, the central question of 'where to set filters' is still present. However, observe that after the set of nodes that observed the k largest values, the output did not change until the considered phase ended (recall a phase is defined as the interval of time steps in which the optimal offline algorithm did not communicate). That is, during a phase the output did not change, only a valid set of filters was tasked to find. One might think of this filter as a certificate which witnesses the correctness of the output.

The problem we tackle in this section raises the following tasks: (1) decide on an output and (2) define filters. Note that if a subset is chosen, it is not clear where to set filters: i.e., a midpoint-strategy might show where the optimal offline algorithm might have set the filters. However, this particular set of node is not necessarily part of the output. These two aspects are orthogonal and thus, yield a competitiveness much larger than the results in previous sections.

Chapter Basis. Parts of the model, analysis, and results in the remainder of this chapter are based on the following publication:

- Mäcker, M., and Meyer auf der Heide. On Competitive Algorithms for Approximations of Top-k-Position Monitoring of Distributed Streams. In: *30th International Parallel and Distributed Processing Symposium (IPDPS, 2016)*, [MMM16].

5.2 Lower Bound for Competitive Algorithms

We show a lower bound on the competitiveness proving any online algorithm has to communicate at least $(\sigma - k)$ times in contrast to an offline algorithm which only uses $k + 1$ messages. Recall that the adversary generates the data streams and can see the filters communicated by the server. Note that as long as the online and the offline algorithm are allowed to make use of an error $\varepsilon \in (0, 1)$ the lower bound holds, even if the errors are different.

Theorem 5.2.1. *Any filter-based online algorithm which solves the Approximate-Top-k-Position Monitoring problem and is allowed to make use of an error of $\varepsilon \in (0, 1)$ has a competitiveness of $\Omega\left(\sigma/k\right)$ compared to an optimal offline algorithm which is allowed to use a (potentially different) error of $\varepsilon' \in (0, 1)$.*

Proof. Consider an instance in which the observed values of $\sigma \in [k + 1, n]$ nodes are equal to some value y_0 (the remaining $n - \sigma$ nodes observe smaller values) at time $t = 0$ and the following adversary: In time step $r = 0, 1, \ldots, n - k$, the adversary decides to change the value of one node i with $v_i^r = y_0$ to be $v_i^{r+1} = y_1 < (1 - \varepsilon) \cdot y_0$ such that a filter violation occurs. Observe that such a value y_1 exists if $\varepsilon < 1$ holds and a node i always exists since otherwise the filters assigned by the online algorithm cannot be feasible. Hence, the number of messages sent by the online algorithm until time step $n-k$ is at least $n-k$. In contrast, the offline algorithm knows the $n-k$ nodes whose values change over time and hence, can set the filters such that no filter violation happens. The offline algorithm sets two different filters: One filter $F_1 = [y_0, \infty]$ for those k nodes which have a value of y_0 at time step $n - k$ using k messages and one filter $F_2 = [0, y_0]$ for the remaining $n - k$ nodes using one broadcast message. By essentially repeating these ideas, the input stream can be extended to an arbitrary length, obtaining the lower bound as stated. □

A Short Remark. The lower bound presented above can be enlarged such that the (asymptotic) competitiveness is $\Omega(\sigma)$ since the optimal offline algorithm needs only to communicate a constant amount of filters for each phase (as described above).

5.3 Upper Bounds for Competitive Algorithms

Now we propose an algorithm DENSEPROTOCOL and analyze the competitiveness against an optimal offline algorithm in the setting that both algorithms are allowed to use an error of ε.

A Simple Assumption. The algorithm DENSEPROTOCOL is started a time t. For sake of simplicity we assume that the k-th and the $(k+1)$-st node observe the same value z, that is $z \coloneqq v^t_{\pi(k,t)} = v^t_{\pi(k+1,t)}$. However, if this does not hold we can define the filters to be $F_1 = [v^t_{\pi(k+1,t)}, \infty]$ and $F_2 = [0, v^t_{\pi(k,t)}]$ until a filter violation is observed at some time t' using $\mathcal{O}(k + \log n)$ messages in expectation. If the filter violation occurred from below define $z \coloneqq v^t_{\pi(k,t)}$ and if a filter violation from above is observed define $z \coloneqq v^t_{\pi(k+1,t)}$.

Technical Idea. The high-level idea of this protocol is similar to the APRXTOP-K. That is, we compute a guess L on the lower endpoint of the filter of the output $\mathcal{F}^*$ of OPT (assuming OPT did not communicate during $[t, t']$) for which the invariant $\ell^* \in L^* \subseteq L_r$ holds. The goal of DENSEPROTOCOL is to halve the interval L while maintaining $\ell^* \in L$ until $L = \emptyset$ and thus show that no value exists which could be used by OPT.

To this end, the algorithm partitions the nodes into three sets. Intuitively speaking, the first set which we call V_1 contains those nodes which have to be part of the optimal output, V_3 those nodes that cannot be part of any optimal output and V_2 the remaining nodes. The sets change over time as follows. Initially V_1^t contains those nodes that observes a value $v_i^t > \frac{1}{1-\varepsilon} z$. Since the algorithm may discover at a time $t' > t$ that some node i has to be moved to $V_1^{t'+1}$ which also contains all nodes from previous rounds, i.e., $V_1^{t'} \subseteq V_1^{t'+1}$. On the other hand, V_3^t initially contains the nodes which observed a value $v_i^t < (1-\varepsilon)z$. Here also the algorithm may discover at a time $t' > t$ that some node i has to be moved to $V_3^{t'+1}$ which (similar to V_1) contains nodes from previous rounds. At the time t the set V_2^t simply contains the remaining nodes $\{1, \ldots, n\} \setminus (V_1^t \cup V_3^t)$ and its cardinality will only decrease over time.

In the following we make use of sets S_1 and S_2 to indicate that nodes in V_2 may be moved to V_1 or V_3 depending on the values observed by the remaining nodes in V_2. Nodes in S_1 observed a value larger than z but still not that large to decide to move it to V_1 and similarly nodes in S_2 observed smaller values than z but not that small to move it to V_3.

5.3.1 The DENSEPROTOCOL

Next we propose the algorithm DENSEPROTOCOL in which we make use of an algorithm SUBPROTOCOL for the scenario in which some node i exists; that is, in S_1 and in S_2. At a time at which the SUBPROTOCOL terminates it outputs that ℓ^* has to be in the lower half of L or in the upper half of L thus, the interval L is halved (which initiates the next round) or moves one node from V_2 to V_1 or V_3. Intuitively speaking SUBPROTOCOL is designed such that, if OPT did not communicate during $[t_0, t]$, where t_0 is the time the DENSEPROTOCOL is started and t is the current time step, the movement of one node $i \in V_2$ to V_1 or V_3 implies that i has necessarily to be part of $\mathcal{F}^*$ or not. For now we assume the algorithm SUBPROTOCOL to work correctly as a black box using $SUB(n, |L|)$ number of messages.

Note that in case L contains one value and is halved, the interval L is redefined to be empty. In case the algorithm observes multiple nodes reporting a filter violation the server processes one violation at a time in an arbitrary order. Since the server may define new filters after processing a violation one of the multiple filter violations may be not relevant any longer, thus the server simply ignores it.

Algorithm 8 DENSEPROTOCOL

1. Define $z \coloneqq v^t_{\pi(k,t)} = v^t_{\pi(k+1,t)}$ and the following sets:
   $V_1 \coloneqq \{i \in \{1, \dots, n\} \mid v^t_i > \frac{1}{1-\varepsilon} z\}$,
   $V_3 \coloneqq \{i \in \{1, \dots, n\} \mid v^t_i < (1-\varepsilon)z\}$,
   $V_2 \coloneqq \{1, \dots, n\} \setminus (V_1 \cup V_3)$.
   Define an interval $L \coloneqq [(1-\varepsilon)z, z]$ and define sets S_1, S_2 of nodes which are initially empty and use S to denote $S_1 \cup S_2$.

2. The following **rules** are applied for each time step t:
   Let ℓ be the midpoint of L and $u \coloneqq \frac{1}{1-\varepsilon}\ell$
   For a node i the filter is defined as follows:
   If $i \in V_1$, $F_i \coloneqq [\ell, \infty]$;
   If $i \in V_2 \cap S_1$, $F_i \coloneqq [\ell, \frac{1}{1-\varepsilon} z]$.
   if $i \in V_2 \setminus S$, $F_i \coloneqq [\ell, u]$;
   If $i \in V_2 \cap S_2$, $F_i \coloneqq [(1-\varepsilon)z, u]$.
   if $i \in V_3$, $F_i \coloneqq [0, u]$.
   The output $\mathcal{F}(t)$ is defined as $V_1 \cup (S_1 \setminus S_2)$ and $k - |V_1 \cup (S_1 \setminus S_2)|$ many nodes from $V_2 \setminus S_2$.

DENSEPROTOCOL (part 2)

3. Wait until time t, at which some node i reports a filter violation:
   a. **If** $i \in V_1$, **then** redefine L to be the lower half of L and define $S_2 \coloneqq \emptyset$.
   b. **If** $i \in (V_2 \setminus S)$ violates its filter from below **then**
      b.1. **If** the server observed strictly more than k nodes with larger values than u_r **then** redefine L to be the upper half of L and define $S_1 \coloneqq \emptyset$.
      b.2. **else** add i to S_1 and update i's filter.
   c. **If** $i \in S_1 \setminus S_2$ violates its filter **then**
      c.1. **If** i violates its filter from below **then** move i from S_1 and V_2 to V_1 and update i's filter.
      c.2. **else** add i to S_2 and call SUBPROTOCOL.
   d. **If** the server observed k nodes with values $v_i > u$ and $n - k$ nodes with values $v_i < \ell$ **then** call APRXTOP-K
   e. **If** L was redefined at this time step t **then**
      e.1. **if** L is empty **then** end the protocol,
      e.2. **else** Apply rules from Step 2 (update u, ℓ, and all filters). Goto Step 3.

   — And their symmetric cases —

   a'. **If** $i \in V_3$ **then** redefine L to be the upper half of L and define $S_1 \coloneqq \emptyset$.
   b'. **If** $i \in (V_2 \setminus S)$ violates its filter from above **then**
      b'.1. **If** the server observed strictly more than $n - k$ nodes with smaller values than ℓ **then** redefine L to the lower half of L and define $S_2 \coloneqq \emptyset$.
      b'.2. **else** add i to S_2.
   c'. **If** $i \in S_2 \setminus S_1$ violates its filter **then**
      c'.1. **If** i violates its filter from above
      then delete i from S_2, delete i from V_2, and add i to V_3.
      c'.2. **else** add i to S_1 and call SUBPROTOCOL.

Analysis. We analyze the correctness of the protocol in the following lemma and the number of messages used in Lemma 5.3.2. We prove that OPT communicated at least once in Lemma 5.3.6. Note that we assume (for the moment) the SUBPROTOCOL to be correct in order to prove the correctness of DENSEPROTOCOL.

Lemma 5.3.1. *The protocol* DENSEPROTOCOL *computes a correct output $\mathcal{F}(t)$ at any time t.*

Proof. By definition the output consists of nodes from V_1, S_1 and (arbitrary) nodes from $V_2 \setminus S_2$ (cf. step 2.). Observe that by definition of the filters of the nodes in these subsets, the minimum of all lower endpoints of the filters is ℓ following the rules in step 2. Also observe that the maximum of all upper endpoints of the filters of the remaining nodes is u. Since by definition $u = \frac{1}{1-\varepsilon}\ell$ holds, the values observed by nodes $i \in \mathcal{F}_1$ are (lower) bounded by ℓ and nodes $i \in \mathcal{F}_2$ are (upper) bounded by u, thus the overlap of the filters is valid.

Now we argue that there are at least k nodes in the set $V_1 \cup S_1 \cup V_2 \setminus S_2$. To this end, assume to the contrary that t is the first time step at which strictly less than k nodes are in the union of these sets. Now observe that the cases in the DENSEPROTOCOL in which nodes are deleted from one of V_1, S_1 or $V_2 \setminus S_2$ are 3.c.1., 3.c.2., and 3.b'.2..

Observe that in step 3.c.1. the algorithm moves i from S_1 and V_2 to V_1 and thus i is again part of the output and does not change the cardinality. In step 3.c.2. the node i is added to S_2 and SUBPROTOCOL is called afterwards. At this time t' node i is (again) part of the output of SUBPROTOCOL and thus there are sufficiently many nodes to choose as an output which is a contradiction to the assumption. In the remaining case 3.b'.2. DENSEPROTOCOL adds i to S_2. However, since at time t' strictly less than k nodes are in $V_1 \cup S_1 \cup (V_2 \setminus S_2)$, there are strictly more than $n-k$ nodes in $S_2 \cup V_3$ and thus, the algorithm would execute step 3.b'.1. instead. This leads to a contradiction to the assumption. By these arguments the correctness follows. □

Lemma 5.3.2. *The protocol* DENSEPROTOCOL *uses an amount of* $\mathcal{O}(k + \log n + \sigma \log(\varepsilon v_k) + (\sigma + \log(\varepsilon v_k)) \cdot SUB(\sigma, |L|))$ *messages in expectation.*

Proof. Initially the algorithm computes the Top-k and probes all nodes which are in the ε-neighborhood of the node observing the k-th largest value, using $\mathcal{O}(k \log n + \sigma)$ messages on expectation.

During each round r each node can only violate its filter at most constant times without starting the next round $r+1$ or leading to a call of SUBPROTOCOL based on the following simple arguments: All nodes i in V_1 or V_3 directly start the next round $r+1$ after a filter violation. Now fix a node $i \in V_2$ and observe that if it is not contained in S_1 and S_2 it is added to S_1 if a filter violation from below or to S_2 if a filter violation from above is observed. At the time this node i observes a filter violation in the same direction (i.e., from below if it is in S_1 and from above if it is in S_2) it is added to V_1 or V_3. In these cases the next filter violation will start the next round. The last case that remains is that it is added to both sets, S_1 and S_2. Observe that the SUBPROTOCOL is called and starts the next round or decides on one node (which may be different from the fixed node i) to be moved to V_1 or V_3.

Observe that at most $\sigma + 1$ nodes can perform filter violations without starting the next round since each node from V_1 or V_3 directly starts the next round and the number of nodes in V_2 is bounded by σ. Furthermore observe that after each round the interval L is halved thus, after at most $\log |L_0| + 1$ rounds the set L_r is empty.

Now focus on the SUBPROTOCOL which also halves L after termination or decides on one node $i \in V_2^{t'}$ to be moved to $V_1^{t'+1}$ or $V_3^{t'+1}$. Thus, it can be called at most $\sigma + \log(\varepsilon v_k)$ times, leading to the result as stated above. □

5.3.2 The SUBPROTOCOL

We propose an algorithm which is dedicated for the case in the execution of DENSE-PROTOCOL that one node i was added to S_1 and to S_2. In detail, it has observed a value which is larger than u_r and a value which is smaller than ℓ_r. As a short remark, if $i \in \mathcal{F}^*$ would hold, then $\ell^* \leq \ell_r$ follows and on the other hand if $i \notin \mathcal{F}^*$ holds, then $\ell^* \geq \ell_r$ follows, but DENSEPROTOCOL cannot decide $i \in \mathcal{F}^*$ in steps 3.c.2. or 3.c'.2.

Algorithm 9 SUBPROTOCOL

1. Define an interval $L'_0 \coloneqq L_r \cap [(1-\varepsilon)z, \ell_r]$, $S'_1 \coloneqq S_1$, and $S'_2 \coloneqq \emptyset$. Set $r' \coloneqq 0$ indicating the round.

2. The following **rules** are applied for (some) round r':
   Let ℓ'_r be the midpoint of $L'_{r'}$ and $u'_{r'} \coloneqq \frac{1}{1-\varepsilon}\ell'_{r'}$.
   For a node i the filter is defined as follows:
   If $i \in V_1$, $F'_i \coloneqq F_i$;
   If $i \in V_2 \cap (S'_1 \setminus S'_2)$, $F'_i \coloneqq [\ell_r, \frac{1}{1-\varepsilon}z]$.
   If $i \in V_2 \cap S'_1 \cap S'_2$, $F'_i \coloneqq [\ell'_{r'}, \frac{1}{1-\varepsilon}z]$;
   if $i \in V_2 \setminus S'$, $F'_i \coloneqq [\ell_r, u'_{r'}]$.
   if $i \in V_2 \cap (S'_2 \setminus S'_1)$, $F'_i \coloneqq [(1-\varepsilon)z, u'_{r'}]$;
   if $i \in V_3$, $F'_i \coloneqq [0, u'_{r'}]$;

   The output $\mathcal{F}(t)$ is defined as $V_1 \cup (S'_1 \setminus S'_2) \cup (S'_1 \cap S'_2)$ and sufficiently many nodes from $V_2 \setminus S'_2$.

SUBPROTOCOL (part 2)

3. Wait until time t', at which node i reports a filter violation:

   a. **If** $i \in V_1$, **then** terminate SUBPROTOCOL and set L_{r+1} to be the lower half of L_r.

   b. **If** $i \in (V_2 \setminus S')$ violates its filter from below

      b.1. **If** the server observed strictly more than k nodes with larger values than u_r **then**

      - set $L'_{r'+1}$ to be the upper half of $L'_{r'}$ and redefine $S'_1 \coloneqq S_1$.
      - **If** $L'_{r'+1}$ is defined to the empty set **then** terminate SUBPROTOCOL and define the last node i which was in $S'_1 \cap S'_2$ and observed a filter violation from above to be moved to V_3. If such a node does not exist, the node $i \in S_1 \cap S_2$ moves to V_3.

      b.2. **Else** add i to S'_1.

   c. **If** $i \in S'_1 \setminus S'_2$ violates its filter

      c.1. **If** i violates its filter from below **then** move i from V_2 and S'_1 to V_1.

      c.2. **Else** add i to S'_2 and update i'th filter.

   d. **If** $i \in S'_1 \cap S'_2$ violates its filter

      d.1. **If** i violates from below **then** move i to V_1 terminate the SUBPROTOCOL.

      d.2. **else**

      - define $L'_{r'+1}$ to be the lower half of $L'_{r'}$ and redefine $S'_2 \coloneqq \emptyset$.
      - **If** $L'_{r'+1}$ is defined to be the empty set **then** terminate SUBPROTOCOL and move i to V_3.

   e. **If** the server observed k nodes with values $v_i > u_r$ and $n - k$ nodes with values $v_i < \ell_r$ **then** call APRXTOP-K

   f. **If** $L'_{r'+1}$ was set increment r', update $u'_{r'}$, $\ell'_{r'}$, all filters using the rules in 2., and goto step 3.

   — And their symmetric cases —

   a'. **If** $i \in V_3$, **then**

   - set $L'_{r'+1}$ to be the upper half of $L'_{r'}$ and redefine $S'_1 \coloneqq S_1$.
   - **If** $L'_{r'+1}$ is defined to the empty set **then** terminate SUBPROTOCOL and define the last node i which was in $S'_1 \cap S'_2$ and observed a filter violation from above to be moved to V_3. If such a node does not exist the node $i \in S_1 \cap S_2$ moves to V_3.

   b'. **If** $i \in (V_2 \setminus S')$ violates its filter from above

      b'.1. **If** the server observed strictly more than $n-k$ nodes with a value less than ℓ_r, **then** terminate SUBPROTOCOL and set L_{r+1} to be the lower half of L_r.

      b'.2. **else** add i to S'_2.

   c'. **If** $i \in S'_2 \setminus S'_1$

      c'.1. **If** i violates its filter from above **then** move i from V_2 and S'_2 to V_3.

      c'.2. **else** add i to S'_1 and update i'th filter.

Analysis. We analyze the correctness of the SUBPROTOCOL in the following lemma and the number of messages used.

Lemma 5.3.3. *The protocol* SUBPROTOCOL *computes a correct output $\mathcal{F}(t')$ at any time t' at which a node $i \in S_1 \cap S_2$ exists.*

Proof. By definition the output consists of nodes from V_1, $S'_1 \setminus S'_2$, $S'_1 \cap S'_2$ and (arbitrary) nodes from $V_2 \setminus S'_2$ (cf. step 2.). Observe that by definition of the filters of the nodes in these subsets, the minimum of all lower endpoints of the filters is $\ell'_{r'}$ (in case the node is in S_1 and in S_2) following the rules in step 2. Also observe that the maximum of all upper endpoints of the filters of the remaining nodes (in subsets $V_2 \setminus S'$, $S'_2 \setminus S'_1$ or V_3) is $u'_{r'}$. Since by definition $u'_{r'} = \frac{1}{1-\varepsilon}\ell'_{r'}$ holds, the values observed by nodes $i \in \mathcal{F}_1$ are (lower) bounded by $\ell'_{r'}$ and nodes $i \in \mathcal{F}_2$ are (upper) bounded by $u'_{r'}$ thus, the overlap of the filters is valid.

Now we argue that there are at least k nodes in the sets V_1, $S_1 \setminus S_2$, $S_1 \cap S_2$, and $V_2 \setminus S_2$. To this end, simply assume to the contrary that at a time t' there are strictly less than k nodes in the union of the sets. It follows that at this time t', the algorithm has observed that there are strictly more than $n-k$ nodes with a value smaller than $\ell'_{r'}$. Thus, the algorithm would continue (compare case b'.1.) with a lower value of ℓ_r or, in case the interval L_r is empty, terminates (which is a contradiction).

By these arguments the correctness follows. □

Lemma 5.3.4. *The protocol* SUBPROTOCOL *uses at most $\mathcal{O}(\sigma \log |L|)$ messages on expectation.*

Proof. During each round r' each node can only violate its filter at most constant times without starting the next round $r'+1$ based on the following simple arguments: All nodes i in V_1 or V_3 directly start the next round $r'+1$ after a filter violation. Now fix a node $i \in V_2$ and observe that if it is not contained in S'_1 and S'_2 it is added to S'_1 if a filter violation from below or to S'_2 if a filter violation from above is observed. At the time this node i observes a filter violation in the same direction (i.e., from below if it is in S'_1 and from above if it is in S'_2) it is added to V_1 or V_3. In these cases the next filter violation will start the next round. The last case that remains is that it is added to both sets, S'_1 and S'_2. Observe that APRXTOP-K terminates if $i \in S'_1 \cap S'_2$ violates its filter from below (and moves $ito V_1$). Otherwise i violates its filter from above SUBPROTOCOL starts the next round $r'+1$.

Observe that at most $\sigma+1$ nodes can perform filter violations without starting the next round since each node from V_1 or V_3 directly starts the next round ($r+1$ from the

DENSEPROTOCOL or $r' + 1$ this protocol) and the number of nodes in V_2 is bounded by σ.

Furthermore observe that after each round the interval L', the guess of OPTs lower endpoint of the upper filter, is halved. The range of L' is upper bounded by the range of L thus, after at most $\log |L| + 1$ rounds the set L' is empty. □

Lemma 5.3.5. *Given a time point t at which* SUBPROTOCOL *is started. At the time t' which* SUBPROTOCOL *terminates, there is one node i that is moved from V_2 to V_1 or V_3 or the interval L_r (from* DENSEPROTOCOL*) is halved correctly.*

Proof. Focus on the cases in which L' is halved or there is a decision on a node i to move to V_1 or V_3 (cf. cases 3.b.1., 3.d.1. 3.d.2., 3.a'., and 3.c'.1.).

In step 3.b.1. the server observed at the time t' a filter violation from $i \in V_2 \setminus S'$ and there are (strictly) more than k nodes observed with a larger value than $u'_{r'}$. Observe that in this case for all subsets $\mathcal{S}$ with k elements there exists one node $i \notin \mathcal{S}$ which observed a value $v_i \geq u'_{r'}$, thus no matter which set is chosen by OPT, for the upper bound u^* for nodes $i \notin \mathcal{F}^*$ it holds: $u^* \geq u'_{r'}$, and since $u'_{r'} = \frac{1}{1-\varepsilon}\ell'_{r'}$ holds, it follows $\ell^* \geq \ell'_{r'}$. Furthermore if $L'_{r'+1}$ was defined as the empty set, and a node $i \in S'_1 \cap S'_2$ exists, observe that i observes a value $v_i \leq \ell'_{r'}$ and since in this case $u^* \geq u'_{r'}$ holds, $i \notin \mathcal{F}^*$ follows. If such a node i does not exist during the execution of SUBPROTOCOL, the node $i \in S_1 \cap S_2$ which initiated the SUBPROTOCOL can be decided to move to V_3 since during the execution of SUBPROTOCOL the interval L' is only halved to the upper half, thus $i \in S_1 \cap S_2$ observed a value $v_i < \ell_r = \ell'_{r'}$ and since $u^* \geq u'_{r'}$ holds, this $i \notin \mathcal{F}^*$ follows.

In step 3.d.1. the node i observed a value v_i which is larger than $\frac{1}{1-\varepsilon}z$ and thus has to be part of $\mathcal{F}^*$.

In step 3.d.2. the node i observed a value $v_i < \ell'_{r'}$. If during the execution of SUBPROTOCOL the set L' was defined as the upper half at least once then there was a node $j \in V_3$ or strictly more than k nodes which observed a larger value than $u'_{r'}$. It follows that this i cannot be part of $\mathcal{F}^*$. In case during the execution of SUBPROTOCOL the set L' is alway defined to the lower half, then $\ell'_{r'}$ is the lower end of L and since node i observed a value strictly smaller than $\ell'_{r'}$ it cannot be part of $\mathcal{F}^*$.

The arguments for case 3.a'. are similar to 3.b.1.

For the remaining case 3.c'.1. simply observe that i observed a smaller value than $(1-\varepsilon)z$ thus i cannot be part of $\mathcal{F}^*$ follows.

First, focus on the steps in which L is halved and observe that steps 3.a. and 3.b'.1. are the same cases as in the DENSEPROTOCOL. □

Lemma 5.3.6. *Given a time point t at which* DENSEPROTOCOL *is started. Let t' be the time point at which* DENSEPROTOCOL *terminates. During the time interval $[t, t']$ OPT communicated at least once.*

Proof. We prove that OPT communicated by arguing that ℓ^*, the lower endpoint of the upper filter, i.e., the filter for the output $\mathcal{F}^*$, is in the guess L_r at each round r ($\ell^* \in L^* \subseteq L_r$). Hence we show that although if we halve the interval L_r, the invariant $\ell^* \in L^* \subseteq L_r$ is maintained all the time of the execution of DENSEPROTOCOL and possible calls of SUBPROTOCOL.

In the following we assume to the contrary that OPT did not communicate throughout the interval $[t, t']$. We first argue for the execution of DENSEPROTOCOL and assume that the invariant by calls of SUBPROTOCOL hold by Lemma 5.3.5.

First focus on the DENSEPROTOCOL, which halves the interval L_r in steps 3.a., 3.b.1., 3.a'., and 3.b'.1.:

In step 3.a. in which a node $i \in V_1$ violates its filter from above and observes a value $v_i < \ell_r$, it holds: $i \in \mathcal{F}^*$ thus, $\ell^* < \ell_r$ follows.

In step 3.b.1. there are (strictly) more than k nodes with a larger value than u_r. It follows that for all subsets $\mathcal{S}$ (with k elements) there is one node $i \notin \mathcal{S}$ observing a value larger than u_r and thus, $\ell^* \geq (1-\varepsilon)u_r = \ell_r$ holds.

The case 3.a'. (which is symmetric to 3.a.) is executed if a node $i \in V_3$ observed a filter violation ($v_i > u_r$) which implies that the upper endpoint u^* of filter F_2 is larger than v_i and thus, $\ell^* \geq (1-\varepsilon)u_r = \ell_r$.

In step 3.b'.1. (which is symmetric to 3.b.1.) there are (strictly) more than $n-k$ nodes with a smaller value than ℓ_r. It follows that for all subsets $\mathcal{S}$ (with k elements) there is one node $i \in \mathcal{S}$ observing a value smaller than ℓ_r and thus, $\ell^* \leq \ell_r$ holds. □

Theorem 5.3.7. *There is an online algorithm for Approximate-Top-k-Position Monitoring which is $\mathcal{O}(\sigma^2 \log(\varepsilon v_k) + \sigma \log^2(\varepsilon v_k) + \log\log\Delta + \log\frac{1}{\varepsilon})$-competitive against an optimal offline algorithm which may use an error of ε.*

Proof. The algorithm works as follows. At time t at which the algorithm is started, the algorithm probes the nodes holding the $k+1$ largest values. If $v^t_{\pi(k+1,t)} < (1-\varepsilon)v^t_{\pi(k,t)}$ holds, the algorithm APRXTOP-K is called. Otherwise the algorithm DENSEPROTOCOL is executed. After termination of the respective call, the procedure starts over again.

Observe that if the condition holds, there is only one unique output and thus, the APRXTOP-K monitors the Top-k-Positions satisfying the bound of $\mathcal{O}(k + \log n + \log\log\Delta + \log\frac{1}{\varepsilon})$ on the competitiveness. If the condition does not hold, there is

at least one value in the ε-neighborhood of $v^t_{\pi(k,t)}$ and thus, the DENSEPROTOCOL monitors the approximated Top-k-Positions as analyzed in this section.

The number of messages used is simply obtained by adding the number of messages used by the respective algorithms as stated above. □

To obtain the upper bounds stated at the beginning, we upper bound σ by n and v_k by Δ: $\mathcal{O}(n^2 \log(\varepsilon\Delta) + n \log^2(\varepsilon\Delta) + \log\log\Delta + \log\frac{1}{\varepsilon})$. Note that for constant ε and assuming $\Delta = \mathcal{O}(2^n)$ we obtain a much simpler bound of $\mathcal{O}(n^2 \log\Delta)$ on the competitiveness.

5.4 Error Augmentation

In the previous Sections we proposed an algorithm which monitors the Top-k approximately against an approximate (filter-based) offline algorithm. We have seen that for the online algorithm it is much harder to find the optimal filters in comparison to the setting in which the offline algorithm has to compute the exact Top-k. In this section we consider the variant in which the offline algorithm is still allowed to introduce an error; however, we compare the online algorithm which is allowed to err by 2ε against an offline algorithm which is allowed to err by ε.

Corollary 5.4.1. *There is an online algorithm for Approximate-Top-k-Position Monitoring which is $\mathcal{O}(\sigma + \log n + \log\log\Delta + \log\frac{1}{\varepsilon})$-competitive against an optimal offline algorithm which may use an error of $\varepsilon' \leq \frac{\varepsilon}{2}$.*

Proof. At the initial time step t the algorithm probes the nodes holding the $k+1$ largest values. If $v^t_{\pi(k+1,t)} < (1-\varepsilon)v^t_{\pi(k,t)}$ holds the algorithm APRXTOP-K is called.

Otherwise the online algorithm simulates the first round of the DENSEPROTOCOL. That is, nodes are partitioned into V_1, V_2, and V_3 and the filters are defined as proposed (cf step 2. of DENSEPROTOCOL). Here all nodes with values larger than $\frac{1}{1-\varepsilon}(1-\frac{\varepsilon}{2})z$ are directly added to V_1 instead of adding to S_1, and nodes observing values smaller than $(1-\frac{\varepsilon}{2})z$ are added to V_3. Furthermore, if a filter violation from some node $i \in V_2$ is observed, it is directly moved (deleted from V_2 and added) to V_1 in case it violates from below, and added to V_3 if violated from above.

Whenever a node from V_1 (or from V_3) violates its filter the algorithm terminates. Additionally if (strictly) more than k nodes are in V_1 the algorithm is terminated or if (strictly) less than k nodes are in $V_1 \cup V_2$. If exactly k nodes are in V_1 and $n-k$ nodes are in V_3 the APRXTOP-K is executed.

For the following argumentation on the competitiveness we focus on the case that APRXTOP-K was not called since the analysis of APRXTOP-K holds here. Observe that OPT (with an error of ε') had to communicate on the basis of the following observation:

Let t' be the time at which the algorithm terminates. Assume to the contrary that OPT did not communicate during $[t, t']$. In case node $i \in V_1$ observes a filter violation from above, $\left(v_i < (1-\frac{\varepsilon}{2})z\right)$ and $\varepsilon' \leq \frac{\varepsilon}{2}$, OPT had to set $\ell^* \leq v_i$ and $u^* \geq z$, which leads to a contradiction to the definition of filters. In case node $i \in V_3$ observes a filter violation from below, $\left(v_i > \frac{1}{1-\varepsilon}(1-\frac{\varepsilon}{2})z\right)$ and $\varepsilon' \leq \frac{\varepsilon}{2}$, OPT had to set $u^* \geq v_i$ and $\ell^* \leq z$, which leads to a contradiction to the definition of filters. The fact that OPT had

to communicate in the remaining cases follows by the same arguments. Since all cases lead to a contradiction, the bound on the competitiveness as stated above follows. □

CHAPTER 6

FUTURE RESEARCH PERSPECTIVES

In this first part of the thesis we considered filter-based algorithms and gained some initial insights in this research area. However, since there was not much known about filter-based algorithms with respect to competitive analysis, we see several points of extensions and also see a large potential of further investigations.

Generalizations of Filter-based Protocols.

First of all it is worth to elaborate on problems for which a competitive algorithm exists without further restrictions. Similar to the work of Lam et al. [LLT10] we restricted the online and offline algorithms to be filter-based. Only through this was it possible to design an algorithm with a competitiveness independent of the length of the input stream.

Another approach is the model by Yi and Zhang in [YZ12]. In their work, they assume the optimal offline algorithm to be able to see the future data stream in advance, but it has to transmit the data from the sensor node to the server. In other words, one could think of the sensor node executing the offline algorithm, but having to transmit data on every change in the output to the server. Yi and Zhang mentioned that there is no protocol known to be competitive for $n > 1$ sensor nodes. Note that for monitoring the Top-k-Positions, this does not lead to a protocol with bounded competitiveness (for at least 2 nodes). And the same holds for monitoring the Top-k-Values. However, for the problem to output both, the Top-k Positions and the respective values we propose that there exists a protocol which admits bounded competitiveness.

More generally, the question arises which properties make a problem hard or easy to solve – with respect to the analysis we have done in this first part. Intuitively speaking, filters make it possible for sensor nodes to decide **locally** that new observed values do

not infect the output. A more explicit understanding of this phenomenon is needed. Also the explanation should be 'complete' in the sense that if *some* property out of the (potential) set of properties does not hold, the problem directly does not admit bounded competitiveness.

A good starting point is to consider the number of filters an algorithm is able to choose from. The examples we considered in this work only have a bounded number of potential filters. Additionally, a comparable work of Davis et al. [DEI06] has shown the connection between unbounded competitiveness and the fact that the number of filters that can be chosen from is unbounded. In general it is easy to verify that problems with a bounded number of outputs and filters admit bounded competitiveness. This we will elaborate further in the next section – however, the open question is whether there exists a problem with either an infinite number of outputs or filters that still admit bounded competitiveness.

Generic Filter-Based Approach.

The previous question asked in general about the existence of an algorithm. Now we ask whether we can explicitly define a generic algorithm which is competitive. For this, we assume that a given problem admits bounded competitiveness. Assuming that the number of filters and the number of outputs that can be chosen in the presence of the input at a given time step is bounded, it is trivial to come up with the following canonical algorithm:

> Consider a fixed time in space t. Choose an output **arbitrarily** and a filter witnessing its correctness **arbitrarily**. At a later time t', and if there was a filter violation, choose a new output which is correct throughout the time horizon $[t, t']$ (if necessary) and a new filter which was correct throughout $[t, t']$ **arbitrarily**.

Since there is a bounded number of potential outputs and for each output the number of (valid) filters are bounded, the fact that the competitiveness is bounded can be verified easily.

If we apply such a schema to the Top-k-Position monitoring problem (and assuming that a protocol for the one-shot computation uses $\mathcal{O}(\mathcal{C}_n)$ messages), an upper bound of $\mathcal{O}(\mathcal{C}_n + \Delta)$ follows. Although this is much worse than the protocol stated in this thesis, which is only $\mathcal{O}(\mathcal{C}_n + \log \Delta)$ competitive, this generic strategy reflects the rule

to choose the values $s_{k+1}+1, s_{k+1}+2, s_{k+1}+3, \ldots, s_k$ instead of the midpoint strategy.

Now consider the following variation of the generic approach: Instead of choosing an arbitrary output and an arbitrary filter, consider an output and a filter chosen uniformly at random from the set of outputs and filters. Without giving a proof we propose that the expected competitiveness (even against a strongly adaptive adversary) is $\mathcal{O}(\mathcal{C}_n + \log \Delta)$. That is, we propose that the competitiveness only increases by constant factors.

It would be nice to elaborate further with more examples. It might turn out that such a generic approach is very fruitful (still assuming a one-shot protocol is given) and makes it much easier to come up with results for monitoring a problem.

Generalizing to Different Models.

We now address an aspect which was not mentioned before but is still very important: The results presented here (seem to be) tightly coupled to our DMBC model (Distributed Monitoring model with a Broadcast Channel).

Revisiting our protocols the fellow reader observes that the analyzed competitiveness is based on the costs for the one-shot computation on the one hand and on additional costs while the instance 'evolves' over time. Indeed, the costs for the one-shot computation is highly dependent on the considered model and also differs significantly if the model changes. However, the – potentially more interesting – monitoring part might be reused across different variants. We consider two different models to elaborate on this more deeply: the Congested Clique model and a variant of the distributed streaming model which minimizes the communication per node (instead of total communication).

Congested Clique: First, observe that the Top-k-Position monitoring problem can be solved trivially using 1 round of communication: i.e., every node sends a message comprised of the nodes' ID and the value to its neighbors. We define filters for Top-k in the same way as presented in Chapter 4. As long as all nodes observe new values within their filters no communication round is needed to determine the new output. If a filter violation occurred, 1 round is sufficient to determine the new output or new filters. Observe that the problem in this model has a competitiveness of $\mathcal{O}(\log \Delta)$.

Monitoring Graph Streams.

It seems to be promising that graph-related problems (e.g. matching, shortest paths, etc.) makes more sense to consider within the congested clique model.

We see that the results are based on costs for the one-shot computation and on costs for 'not knowing the future'. Consider the Top-k-Monitoring which is $\mathcal{O}(k + \log n + \log \Delta)$ competitive. The one-shot computation has costs of $\mathcal{O}(k + \log n)$ where the costs for redefining filters for consecutive time steps needs $\mathcal{O}(\log \Delta)$ messages.

It makes sense to define the offline algorithm also to be distributed; i.e., each node knows its own future in advance, but the server is still not aware of it. We expect the bounds to become smaller, in comparison to the setting that we use in our analysis.

Smoothed Competitiveness.

Revisiting the worst-case instances (with respect to competitiveness) as a result of our analysis we see that the instances have a high similarity between consecutive time steps. Often only one single node observes a different, very characteristic value which leads to communication in that moment. The inclined reader might think of these instances to be fragile and potentially show improved bounds applying smoothed competitiveness.

Although this approach seems to be promising, it is technically hard to upper bound the expected coefficient between the online and offline costs, since both are random variables (simplified notation):

$$\mathbb{E}\left[\frac{ALG}{OPT}\right]$$

To prevent these difficulties, one could analyze the coefficient of the expected (online, offline) cost. However, an instance-wise comparison seems to make much more sense in this context (also simplified):

$$\frac{\mathbb{E}[ALG]}{\mathbb{E}[OPT]}$$

However, note that this is not an instance-wise comparison any longer and the insights gained by this analysis is (at least) questionable. We propose that for a restricted class of algorithms we can apply a general and substantial simplification of the analysis. Assuming the instance to be given at different time steps, we assume that the instance can be partitioned into phases in which OPT has costs of at least 1 or the instance has infinite size and the phase is the last phase. We propose that for sufficiently large instances, the following holds:

$$\mathbb{E}_I\left[\frac{ALG_I}{OPT_I}\right] \leq \mathcal{O}(\mathbb{E}_p[ALG_p])$$

That is, instead of calculating the expectation of a fraction of two random variables and with respect to the whole instance, it is sufficient to consider the expected cost of the algorithm if applied to a phase p. Furthermore, this phase p does not have worst-case length; the expected length of this phase is considered.

Sliding Windows

In this thesis we considered problems which task the coordinator to output a function on the data **currently** observed. That is, the observations in the past potentially make it simpler to evaluate a function at the current time step. However, are not necessitated for this computation. This excludes a fundamental property of streaming algorithms, namely that an algorithm has to make an **irrevocable** decision (whether to keep the data or to 'throw it away'). To some extent a filter can be seen as a rule for this decision; however, this fact only works in one direction. In case there is no violation of some sensor node's filter, all data can be thrown away. If there exists a node with a filter violation the nodes have to be able to access the value currently observed although they locally did not violate its filter.

In a step towards applying streaming techniques more naturally, problems with respect to sliding windows can be considered: For example, the server is tasked to output the k largest positions / values of the past w time steps (typically w for window size). Expressing our work within this problem formulation, we considered the case in which $w = 1$ holds.

We shortly discuss what enlightening insights we can expect if we tackle this interesting problem in the presence of sliding windows. Now we shortly discuss the problem and gain first insights: Following the work of Giannakopoulos and Koutsoupias [GK12] it is easy to verify that the worst case is a strongly monotonic decreasing function (initially at a value of v at time t). At time step $t + w$ the adversary is able to decide to generate input which is always 0 (for the next w time steps) or, again, present the largest value v.

In case the adversary chooses the larger value v at time $t + w$, the output need not change (and an offline algorithm need not change output and filter). The other case helps to argue that the protocol needs to store all values which are presented. Assuming a protocol does not store the data (or transmitted it to the server) the output is not correct. This yields that the standard competitiveness approach (even with the restriction of filters) does not lead to finite competitiveness.

However, we see potential for different techniques to be able to unfold: Applying Smoothed Competitiveness the first step is to consider one phase (of consecutive

time steps) until the output needs to change due to the perturbation of the input values. We denote such a phase p. The technique which we considered in the last section should reduce the problem of considering the expected costs for the whole instance, to this very simple phase p, and consider the expected costs for this phase with respect to the expected length. Now we divide the phase p into subphases of size w which follow the worst-case characteristics as described above. Assuming each node has local storage capabilities of at most half of the size of the sliding window, $\Omega(w)$ data items have to be transmitted to the server. Applying the techniques of Damerow et al. [DMadHR$^+$03, DMMadH$^+$12] we propose that the number of data items that have to be transmitted to the server are significantly reduced to $\mathcal{O}(\log w)$ due to the perturbation. However, a more detailed and more careful analysis is needed on how the effect of distributed observations affects the applicability of the technique. Furthermore, it would be interesting to see that the competitiveness drops down to be constant for polylogarithmic storage at the sensor nodes.

Part B:

Dynamic Algorithms

Chapter 7

Introduction to Dynamic Algorithms

In the following chapters we consider the problems of dynamically computing an approximation of the frequency, and the count distinct. Furthermore, we develop a distributed dynamic data structure which is used for Top-k and k-Select queries.

Until now, we designed and analyzed filter-based online algorithms, i.e., compare an online protocol against an offline algorithm which (is filter-based and) knows the whole instance in advance. Here, we concentrate on algorithms which initially determine an output and are able to update this output if the distributively given instance partially changes. We aim for algorithms which use significantly fewer costs to dynamically adapt to these changes assuming the fraction of the instance which changes is not too large.

7.1 Model Description

For the sake of self containment we restate the model defined in Section 2.1.

In our setting there are n distributed nodes identified by unique identifiers (IDs) from the set $\{1, \ldots, n\}$, each receiving a continuous data stream $(v_i^1, v_i^2, v_i^3 \ldots)$, which can be exclusively observed by the respective nodes. Also, at time t, a node i observes $v_i^t \in \mathbb{N}$ and does not know any $v_i^{t'}$, $t' > t$. We omit the index t if it is clear from the context.

Following the model in [CMY08], we allow that between any two consecutive time steps, a *communication protocol* exchanging messages between the server and the nodes may take place. The communication protocol is allowed to use an amount of rounds which is polylogarithmic in n and $\max_{1 \leq i \leq n}(v_i^t)$. The nodes can communicate to the coordinator, but cannot communicate to each other. They are able to store a constant number of integers, compare two integers and perform Bernoulli trials with success probability $2^i/n$ for $i \in \{0, \ldots, \log n\}$. The coordinator can, on the one hand, communicate to one device, and has, on the other hand, a broadcast channel to communicate one message received by all nodes at the same time. All the communication methods described above incur unit communication cost per message, the delivery is instantaneous, and we allow a message at time t to have a size at most logarithmic in n and $\max_{1 \leq i \leq n}(v_i^t)$.

Notation of Time. Recall that a time step t defines a point in time at which the sensor nodes obtain a new piece of input (v_i^t for node i at time t). Between two consecutive time steps t and $t + 1$ a communication protocol takes place. The protocol consists of multiple (communication) rounds: A sensor nodes performs local computations and may decide to send a message to the server. The server collects all messages, performs local computations and (may) decide for a message to broadcast to all sensor nodes.

Since all nodes are synchronized, the server is able to identify the situation that no sensor decided to send a message and, on the other hand, the sensor nodes can identify that the server did not decide to send a message. Furthermore, the server is able to collect and read all messages given by the sensor node, even if every node has decided to send a message: i.e., there is no restriction on the capacity of the communication channel in one round.

If the communication protocol of each node for the current time steps terminated, the server decides for the output of the function at time t and the sensor network proceeds from time step t to the next time step $t + 1$.

7.2 Problems Description

In this thesis, we consider the monitoring of different problems related to the *domain* of the network. The domain at time t is defined as $D_t \coloneqq \{v \in \{1, \ldots, \Delta\} \mid \exists i \text{ with } v_i^t = v\}$, the set of values observed by at least one node at time t. For ease of description let $s_1^t, \ldots, s_n^t$ be the *sorted* version of the data items $d_1^t, \ldots, d_n^t$ received at time step t.

Definition 7.2.1 (Frequency)**.** *For each $v \in D_t$ monitor the frequency $|N_t^v|$ of nodes in $N_t^v \coloneqq \{i \in \{1, \ldots, n\} \mid v_i^t = v\}$ that observed v at t; i.e., the number of nodes currently observing v.*

Definition 7.2.2 (Approximate k-Select)**.** *Let $k \in \{1, \ldots, n\}$ be given. The server is tasked to output a value v which is observed by sensor node 'close' to the k-th smallest observation; i.e., close to rank k. Formally, output v with $v \in \{s^t_{(1-\varepsilon)k}, \ldots, s^t_{(1+\varepsilon)k}\}$.*

Definition 7.2.3 (Approximate Count Distinct)**.** *For each $v \in D_t$ monitor the frequency $|N_t^v|$ of nodes in $N_t^v \coloneqq \{i \in \{1, \ldots, n\} \mid v_i^t = v\}$ that observed v at t; i.e., the number of nodes currently observing v.*

7.3 Related Work

The basis of the model considered in this paper is the *continuous monitoring model* as introduced by Cormode, Muthukrishnan and Yi in [CMY08]. In this model, there is a set of n distributed nodes, each observing a stream given by a multiset of items in each time step. The nodes can communicate with a central server, which in turn has the task, at any time t, to continuously compute a function f defined over all data observed across all streams up to time t. The goal is to design protocols aiming at the minimization of the number of bits communicated between the nodes and the server. In [CMY08], the monitoring of several functions is studied in their (approximate) threshold variants, in which the server has to output 1 if $f \geq \tau$ and 0 if $f \leq (1-\varepsilon)\tau$, for given τ and ε. Precisely, algorithms for the frequency moments $F_p = \sum_i m_i^p$ where m_i denotes the frequency of item i for $p = 0, 1, 2$ are given. F_1 represents the simple sum of all items received so far and F_0 the number of distinct items received so far. Since the introduction of the model, monitoring of several functions has been studied, such as the monitoring of frequencies and ranks by Huang, Yi and Zhang in [HYZ12]. The frequency of an item i is defined to be the number of occurrences of i across all streams up to the current time. The rank of an item i is the number of items smaller than i observed in the streams. Frequency moments for any $p > 2$ are considered by

Woodruff and Zhang in [WZ12]. A variant of the Count Distinct Monitoring Problem is considered by Gibbons and Tirthapura in [GT01]. The authors study a model in which each of two nodes receives a stream of items and at the end of the streams a server is asked to compute F_0 on the basis of both streams. A main technical ingredient is the use of so-called public coins, which, once initialized at the nodes, provide a way to let different nodes observe identical outcomes of random experiments without further communication. We will adopt this technique in Section 10.2. Note that the previously mentioned problems are all defined over the items *received so far*, which is in contrast to the definition of monitoring problems which we are going to consider and which are all defined only on the basis of the *current time step*. This fact has the implication that in our problems the monitored functions are no longer monotone, which makes its monitoring more complicated.

Concerning monitoring problems in which the function tracked by the server only depends on the current time step, there is also some previous work to mention. In [LLT10], Lam, Liu and Ting study a setting in which the server, at any time, needs to know the order type of the values currently observed. That is, the server needs to know which node observes the largest value, second largerst value and so on at time t. In [YZ12], Yi and Zhang consider a system only consisting of one node connected to the server. The node continuously observes a d-dimensional vector of integers from $\{1, \ldots, \Delta\}$. The goal is to keep the server informed about this vector up to some additive error per component.

Chapter 8

Fully Dynamic Algorithm for the Frequency Problem

In this section we design and analyze a fully dynamic algorithm for the FREQUENCY Problem, i.e., to output (an approximation) of the number of nodes currently observing value v, for each $v \in \{1, \ldots, \Delta\}$.

The problem is of fundamental interest since it provides approximations for several other problems, e.g. Maximum, Sum, Top-k, Quantiles, and Heavy Hitters.

We start by considering a single time step and present an algorithm that solves the subproblem of outputting the number of nodes that observe v within a constant multiplicative error bound. Afterwards, and based on this subproblem, a simple sampling algorithm is presented which solves the Frequency Problem for a single time step up to a given (multiplicative) error bound and with given error probability.

Furthermore, we consider multiple time steps and assume that the change in the frequency is upper bounded by a constant factor σ. By this, we can show significantly smaller communication bounds for multiple time steps in comparison to worst-case instances.

Observe that the FREQUENCY Problem inherits the DOMAIN Problem. Recall that the DOMAIN (of t) is the set of observed values at a specific time step t as defined in Definition 2.2.2. In fact, we use the DOMAINPROTOCOL (as presented in Algorithm 3, Section 3.4) as a subroutine and evaluate the frequency of each observed value afterwards.

8.1 Introduction & Results

Problem	Expected communication costs
Frequency (One Shot)	$\mathcal{O}\left(\frac{d}{\varepsilon^2}\log\frac{d}{\delta}\right)$
Frequency (Maintaining)	$\mathcal{O}\left(\sum_{t\in T}\lvert D_t\rvert\frac{1}{\varepsilon^2}\log\frac{\lvert D_t\rvert}{\delta}\right)$
	$\mathcal{O}\left((\sigma+\delta)\cdot T\frac{d}{\varepsilon^2}\log\frac{d}{\delta}\right)$

Recall the DOMAIN problem in Definition 2.2.2, which is to identify the set of values observed at a specific time step t. Let d_t be the size of the Domain D_t at the time step t, which is defined as $d_t \coloneqq |\{v \in \{1,\dots,\Delta\} | \exists i : v_i = v\}|$. Let $|N_t^v|$ denote the number of nodes that observe value v at time step t. The subscript t is suppressed if it is clear from the context. Here, we compute an (ε,δ)-approximation; i.e., for a given (multiplicative) error bound ε and a failure probability δ the algorithm ensures that for each value v the output $\tilde{n}_t^v$ fulfills $\tilde{n}_t^v \in [(1-\varepsilon)|N_t^v|, (1+\varepsilon)|N_t^v|]$ with probability at least $1-\delta$ at a time step t.

Note that if we assume that the domain does not change during a time interval of T time steps, applying the one-shot computation T times yields an upper bound of $\mathcal{O}(T\frac{d}{\varepsilon^2}\log\frac{d}{\delta})$. That is, the algorithm which exploits similarities between consecutive time steps uses less communication by a factor of σ.

A Short Technical Overview. In this chapter we apply simple standard sampling techniques. As a first step the DOMAINPROTOCOL is applied to identify the set of observed values using $\mathcal{O}(d)$ messages in expectation. Afterwards, for each value v we apply a variant of the DOMAINPROTOCOL to estimate the frequency $|N_t^v|$ of the value $v \in D_t$ by a constant factor with a given failure probability δ' using $\mathcal{O}(\log\frac{1}{\delta'})$ messages. Through this, we can apply a standard sampling technique to estimate $|N_t^v|$ up to (given) error ε and failure probability δ using $\mathcal{O}(\frac{1}{\varepsilon}\log\frac{1}{\delta})$ messages in expectation.

To reduce the communication for multiple time steps, the idea is straightforward. We consider a phase of at most T time steps and a single value v as follows: Initially, the frequency is approximated up to $\varepsilon/3$ error. Instead of calculating the frequency itself, the algorithm samples the number of nodes that observe v for the first time (denoted by *entering* nodes) and on the other hand the number of nodes that do not longer observe v (denoted by *leaving* nodes). Since this number of nodes that change is upper bounded by $\sigma|N_t^v|$, the protocol determines the frequency for each time step using the initial frequency and adds / substracts the sample for entering and leaving nodes.

Chapter Basis. Parts of the model, analysis, and results in the remainder of this chapter are based on the following publication:

- Bemmann, Biermeier, Bürmann, Kemper, Knollmann, Knorr, Kothe, Mäcker, M., Meyer auf der Heide, Riechers, Schaefer, Sundermeier. Monitoring of Domain Related Problems in Distributed Data Streams. In: *Structural Information and Communication Complexity - 24th International Colloquium (SIROCCO 2017)*, [BBB+17].

8.2 Frequencies – A One-Shot Computation

8.2.1 Constant Factor Approximation of Frequencies

In this section we will use the DOMAIN protocol to estimate the number of nodes that measure a certain value v. Observe that the expected maximal height of the geometric experiment increases with a growing number of nodes observing v. We exploit this fact and use it to estimate the number of nodes with value v, while still expecting constant communication cost only. For a given time step t and a value $v \in D_t$, we define an algorithm FREQUENCYCONSTANTFACTORAPPROXIMATION as follows:

Algorithm 10 FREQUENCYCONSTANTFACTORAPPROXIMATION

1. Apply DOMAINPROTOCOL (cf. Algorithm 3) for all sensor nodes i.
2. Let r_v denote the (first) communication round in which the server receives the response (for a value v).
3. The algorithm outputs $\tilde{n}^v_{\text{const}} \coloneqq 2^r$ as the estimation for $|N^v_t|$.

We show that we compute a constant factor approximation with constant probability. Then we amplify this probability using multiple executions of the algorithm and taking the median (of the executions) as a final result.

Lemma 8.2.1. *The algorithm* FREQUENCYCONSTANTFACTORAPPROXIMATION *estimates the number $|N^v_t|$ of nodes observing the value v at time t up to a factor of 8, i.e., $\tilde{n}^v_{const} \in [|N^v_t|/8, |N^v_t| \cdot 8]$ with constant probability.*

Proof. Let n^v be the number of nodes currently observing value v, i.e., $n^v \coloneqq |N^v_t|$. Recall that the probability for a single node to draw height h is $\Pr[h_i = h] = \frac{1}{2^h}$, if $h < \log n$, and $\Pr[h_i = h] = \frac{2}{2^h}$, if $h = \log n$. Hence, $\Pr[h_i \geq h] = \frac{1}{2^{h-1}}$ for all $h \in \{1, \ldots, \log n\}$.

We estimate the probability of the algorithm to fail, by analysing the cases that $\tilde{n}^v_{const}$ is larger than $\log n^v + 3$ or smaller than $\log n^v - 3$. We start with the first case and by applying a union bound we obtain:

$$\begin{aligned}\Pr[\exists i : h_i > \log n^v + 3] &\leq \Pr[\exists i : h_i \geq \lceil \log n^v \rceil + 3] \\ &= n^v \cdot \left(\frac{1}{2}\right)^{\lceil \log n^v \rceil + 2} \leq \frac{1}{4}.\end{aligned}$$

For the latter case we bound the probability that each node has drawn a height strictly smaller than $\log n^v - 3$ by

$$\begin{aligned}\Pr[\forall i : h_i < \log n^v - 3] &\leq \prod_i \Pr[h_i < \lceil \log n^v \rceil - 3] \\ &= \left(1 - \frac{1}{2^{\lceil \log n^v \rceil - 4}}\right)^{n^v} \leq \left(1 - \frac{8}{n^v}\right)^{n^v} \leq \frac{1}{e^8}.\end{aligned}$$

Thus, the probability that we compute an 8-approximation is bounded by

$$\begin{aligned}\Pr\left[\frac{n^v}{8} \leq 2^{h_i} \leq 8n^v\right] &= 1 - \left(\Pr[\exists i : h_i > \log n^v + 3] + \Pr[\forall i : h_i < \log n^v - 3]\right) \\ &\geq 1 - \left(\frac{1}{4} + \frac{1}{e^8}\right) > 0.7\end{aligned}$$

□

We apply an amplification technique to boost the success probability to arbitrary $1 - \delta'$ using $\Theta(\log \frac{1}{\delta'})$ parallel executions of the FREQUENCYCONSTANTFACTORAPPROXIMATION algorithm and choose the median of the intermediate results as the final output.

Corollary 8.2.2. *Applying* $\Theta\left(\log \frac{1}{\delta'}\right)$ *independent, parallel instances of* FREQUENCYCONSTANTFACTORAPPROXIMATION*, we obtain a constant factor approximation of* $|N^v_t|$ *with success probability at least* $1 - \delta'$ *using* $\Theta\left(\log \frac{1}{\delta'}\right)$ *msg. in expectation.*

Proof. Choose $d = \frac{45}{2} \ln \frac{1}{\delta'}$ to be the number of copies of the algorithm and return the median of the intermediate results. Let $\mathcal{I}_j$ be the indicator variable for the event that the j-th experiment does not result in an 8-approximation. By Lemma 8.2.1 the failure probability can be upper bounded by a constant, i.e., $\Pr[\mathcal{I}_j] \leq 0.3$. Hence, using a Chernoff bound, the probability that at least half of the experiments do meet

the required approximation factor of 8 is

$$\Pr\left[\sum_{j=1}^{d}\mathcal{I}_j \geq \frac{1}{2}d\right] \leq \Pr\left[\sum_{j=1}^{d}\mathcal{I}_j \geq \left(1+\frac{2}{3}\right)\cdot 0.3\cdot d\right] \leq e^{-\left(\frac{2}{3}\right)^2\cdot\frac{1}{3}\cdot 0.3\cdot d} = e^{-\frac{2}{45}\cdot d} = e^{-\frac{2}{45}\cdot\frac{45}{2}\ln\frac{1}{\delta'}} = \delta'.$$

Observe that if at least half of the intermediate results are within the demanded error bound, so is the median. Thus, the algorithm produces an 8-approximation of $|N_t^v|$ with a success probability of at least $1-\delta'$, concluding the proof. □

8.2.2 Arbitrary Approximation of Frequencies

To obtain an (ε,δ)-approximation, in Algorithm 11 we first apply the FREQUENCY-CONSTANTFACTORAPPROXIMATION algorithm to obtain a rough estimate of $|N_t^v|$ (cf. Algorithm 10). It is used to compute a probability p, which is broadcasted to the nodes, so that every node observing value v sends a message with probability p. Since the FREQUENCYCONSTANTFACTORAPPROXIMATION result $\tilde{n}^v_{\text{const}}$ in the denominator of p is close to $|N_t^v|$, the number of messages sent on expectation is independent of $|N_t^v|$. The estimated number of nodes observing v is then given by the number of responding nodes $\bar{n}^v$ divided by p, which, on expectation, results in $|N_t^v|$.

Algorithm 11 FREQUENCYEPSILONFACTORAPPROX($v \in D_t, \varepsilon, \delta$)

(Node i)

1. Receive p from the server.
2. Send a response message with probability p.

(Server)

1. Set $\delta' := \frac{\delta}{3}$
2. Call FREQUENCYCONSTANTFACTORAPPROXIMATION(v, δ') to obtain $\tilde{n}^v_{\text{const}}$.
3. Broadcast $p = \min\left(1, \frac{24}{\varepsilon^2\tilde{n}^v_{\text{const}}}\cdot\ln\frac{1}{\delta'}\right)$.
4. Receive $\bar{n}^v$ messages.
5. Compute and output estimated number of nodes in N_t^v as $\tilde{n}^v = \bar{n}^v/p$.

Lemma 8.2.3. *The algorithm* FREQUENCYEPSILONFACTORAPPROX *as given in Algorithm 11 provides an (ε,δ)-approximation of* $|N_t^v|$.

Proof. The algorithm obtains a constant factor approximation $\tilde{n}^v_{\text{const}}$ with probability $1-\delta'$. The expected number of messages is $\mathbb{E}[\bar{n}^v] = n^v\cdot p$.

We start by estimating the conditional probability that more than $(1+\varepsilon)\,n^v p$ responses are sent under the condition that $\tilde{n}^v_{\text{const}} \leq 8n^v$ and $p < 1$. In this case we

have

$$p = \frac{24}{\varepsilon^2 \tilde{n}^v_{\text{const}}} \cdot \ln \frac{1}{\delta'} \geq \frac{3}{\varepsilon^2 n^v} \cdot \ln \frac{1}{\delta'},$$

hence using a Chernoff bound it follows

$$p_1 \coloneqq \Pr\left[\bar{n}^v \geq (1+\varepsilon) n^v p \,|\, \tilde{n}^v_{\text{const}} \leq 8n^v \wedge p < 1\right] \leq e^{-\frac{\varepsilon^2}{3} n^v \cdot \frac{3}{\varepsilon^2 n^v} \cdot \ln \frac{1}{\delta'}} = \delta'.$$

Likewise the probability that less than $(1-\varepsilon)\, n^v p$ messages are sent under the condition that $\tilde{n}^v_{\text{const}} \leq 8n^v$ and $p < 1$ is

$$\begin{aligned} p_2 &\coloneqq \Pr\left[\bar{n}^v \leq (1-\varepsilon) n^v p \,|\, \tilde{n}^v_{\text{const}} \leq 8n^v \wedge p < 1\right] \\ &\leq e^{-\frac{\varepsilon^2}{2} n^v \cdot \frac{3}{\varepsilon^2 n^v} \cdot \ln \frac{1}{\delta'}} \leq e^{-\frac{3}{2} \ln \frac{1}{\delta'}} < \delta'. \end{aligned}$$

Next consider the case that $\tilde{n}^v_{\text{const}} > 8n^v$ and $p < 1$ holds. Using

$$\Pr\left[\tilde{n}^v_{\text{const}} > 8n^v\right] \leq \Pr\left[\tilde{n}^v_{\text{const}} > 8n^v \vee \tilde{n}^v_{\text{const}} < \frac{n^v}{8}\right] \leq \delta'$$

and $p_i \cdot \Pr\left[\tilde{n}^v_{\text{const}} \leq 8n^v\right] \leq p_i$ for $i \in \{1, 2\}$,

$$\begin{aligned} &\Pr\left[(1-\varepsilon) n^v p < \bar{n}^v < (1+\varepsilon) n^v p \,|\, p < 1\right] \\ &\quad \geq 1 - \left(\Pr\left[\tilde{n}^v_{\text{const}} > 8n^v\right] + (p_1 + p_2)\right) \geq 1 - 3\delta' = 1 - \delta. \end{aligned}$$

For the last case $p = 1$, we have $\Pr\left[(1-\varepsilon) n^v p < \bar{n}^v < (1+\varepsilon) n^v p \,|\, p \geq 1\right] = 1$, by using $\bar{n}^v = n^v$. Now, $\Pr\left[(1-\varepsilon) n^v p < \bar{n}^v < (1+\varepsilon) n^v p\right] \geq 1 - \delta$ directly follows. □

Lemma 8.2.4. *Algorithm* FREQUENCYEPSILONFACTORAPPROX *as given in Algorithm 11 uses* $\Theta(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ *messages on expectation.*

Proof. Recall that each of the n^v nodes sends a message with probability p, leading to $n^v \cdot p$ messages on expectation. First assume that the constant factor approximation was successful, i.e., $\frac{n_1}{8} \leq \tilde{n}^v_{\text{const}} \leq 8n_1$. If $p < 1$, we have

$$n^v \cdot p = n^v \frac{24}{\varepsilon^2 \tilde{n}^v_{\text{const}}} \cdot \ln \frac{1}{\delta'} \leq \frac{24 \cdot 8}{\varepsilon^2} \cdot \ln \frac{1}{\delta'} = \Theta\left(\frac{1}{\varepsilon^2} \log \frac{1}{\delta}\right).$$

If $p = 1$, by definition $\frac{24}{\varepsilon^2 \tilde{n}^v_{\text{const}}} \cdot \ln \frac{1}{\delta'} \geq 1$, hence $\tilde{n}^v_{\text{const}} = \mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta'}\right)$. Thus, $n^v p \leq 8\tilde{n}^v_{\text{const}} p = \mathcal{O}\left(\frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta'}\right)$.

For the case that the constant factor approximation was not successful, note that

$\Pr\left[\tilde{n}^v_{\text{const}} < \frac{1}{8\cdot 2^i} n^v\right] \leq \frac{1}{e^{2^{i+3}}}$ holds analogously to the calculation in Lemma 8.2.1. Also, for $\tilde{n}^v_{\text{const}} \geq \frac{1}{8\cdot 2^i} n^v$ and $p < 1$, we have

$$n^v p \leq 8 \cdot 2^i \cdot \tilde{n}^v_{\text{const}} \cdot \frac{24}{\varepsilon^2 \tilde{n}^v_{\text{const}}} \cdot \ln\frac{1}{\delta} = 2^i \cdot \Theta\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right).$$

Similarly, for $p = 1$, we have $n^v p \leq 8 \cdot 2^i \cdot \tilde{n}^v_{\text{const}} = 2^i \cdot \Theta\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)$ as in this case, $\tilde{n}^v_{\text{const}} = \mathcal{O}\left(\frac{1}{\varepsilon^2}\cdot\log\frac{1}{\delta'}\right)$. Hence, we can conclude

$$\begin{aligned}\mathbb{E}\left[\bar{n}^v\right] &\leq \Theta\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)\cdot\Pr\left[\tilde{n}^v_{\text{const}} \geq \frac{1}{8}n^v\right] \\ &\quad + \sum_{i=0}^{\infty}\Pr\left[\frac{1}{8\cdot 2^{i+1}}n^v \leq \tilde{n}^v_{\text{const}} < \frac{1}{8\cdot 2^i}n^v\right]\cdot 2^{i+1}\cdot\Theta\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)\end{aligned}$$

$$\begin{aligned}&\leq \Theta\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)\left(1+\sum_{i=0}^{\infty}\frac{2^{i+1}}{e^{2^{i+3}}}\right) \leq \Theta\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)\left(1+\sum_{i=0}^{\infty}2^{i+1-2^{i+3}}\right)\\ &\leq \Theta\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)\left(1+\sum_{i=0}^{\infty}2^{-i}\right) = \Theta\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right).\end{aligned}$$

□

Theorem 8.2.5. *There exists an algorithm that provides an (ε,δ)-approximation for the Frequency Monitoring Problem for T time steps using $\Theta\left(\sum_{t\in T}|D_t|\frac{1}{\varepsilon^2}\log\frac{|D_t|}{\delta}\right)$ messages in expectation.*

Proof. In every time step t we first identify D_t by applying DOMAIN using $\Theta(|D_t|)$ messages on expectation. On every value $v \in D_t$ we then perform algorithm FREQUENCYEPSILONFACTORAPPROX(v,ε,$\frac{\delta}{|D_t|}$), using $\Theta\left(|D_t|\frac{1}{\varepsilon^2}\log\frac{|D_t|}{\delta}\right)$ messages in expectation for a single time step, while achieving a probability (using the union bound) of $1 - \frac{|D_0|\delta}{|D_0|} = 1 - \delta$ for one time step the estimation to be an ε-approximation (for each value v). Applied for each of the T time steps, we obtain a bound as claimed. □

8.3 Maintaining Frequencies over Multiple Time Steps

Applying FREQUENCYEPSILONFACTORAPPROX in every time step is a good solution in worst-case scenarios. But if we assume that the change in the set of nodes observing a value is small in comparison to the size of the set, we can do better.

We extend the FREQUENCYEPSILONFACTORAPPROX such that in settings where from one time step to another only a small fraction σ of nodes change the value they measure, the amount of communication can be reduced, while the quality guarantees remain intact. We define σ such that

$$\forall t : \sigma \geq \frac{|N_{t-1}^v \setminus N_t^v| + |N_t^v \setminus N_{t-1}^v|}{|N_t^v|}.$$

Note that this also implies that $D_t = D_{t-1}$ holds for all time steps t; i.e., the set of measured values stays the same over time.

The extension is designed so that in comparison to FREQUENCYEPSILONFACTORAPPROX, also in settings with many changes the solution quality and message complexity asymptotically do not increase. The idea is the following: For a fixed value v, in a first time step FREQUENCYEPSILONFACTORAPPROX is executed (defining a probability p in Step 3 of Algorithm 11). In every following time step, up to $1/\delta$ consecutive time steps, nodes that start or stop measuring a value v send a message to the server with the same probability p, while nodes that do not observe a change in their value remain silent. In every time step t, the server uses the accumulated messages from the first time step and all messages from nodes that started measuring v in time steps $2 \dots t$, while subtracting all messages from nodes that stopped measuring v in the time steps $2 \dots t$. This accumulated message count is then used similarly as in FREQUENCYEPSILONFACTORAPPROX to estimate the total number of nodes observing v in the current time step. The algorithm starts again if a) $1/\delta$ time steps are over, so that the probability of a good estimation remains good enough, or b) the sum of estimated nodes to start/stop measuring value v is too large. The latter is done to ensure that the message probability p remains fitting to the number of nodes, ensuring a small amount of communication, while guaranteeing an (ε, δ)-approximation.

Let n_t^+, n_t^- be the number of nodes that start measuring v in time step t or that stop measuring it, respectively, i.e., $n_t^+ = |N_t^v \setminus N_{t-1}^v|, n_t^- = |N_{t-1}^v \setminus N_t^v|$, and $\bar{n}_t^+$ and $\bar{n}_t^-$ the number of them that sent a message to the server in time step t. In the following we call nodes contributing to n_t^+ and n_t^- *entering* and *leaving*, respectively.

Algorithm 12 CONTINUOUSFREQUENCYEPSILONAPPROX(v, ε, δ)

(Node i)

1. If $t = 1$, take part in FREQUENCYEPSILONFACTORAPPROX called in Step 2 by the server.
2. If $t > 1$, broadcast a message with probability p if $v_i^{t-1} = v \wedge v_i^t \neq v$ or $v_i^{t-1} \neq v \wedge v_i^t = v$.

(Server)

1. Set $\delta' := \delta^2$.
2. Set $t := 1$ and run FREQUENCYEPSILONFACTORAPPROX(v, $\varepsilon/3$, δ) to obtain $\bar{n}_1, p$.
3. Output $\tilde{n}_1 = \frac{\bar{n}_1}{p}$.
4. Repeat at the beginning of every new time step $t > 1$:
   (a) Receive messages from nodes changing the value to obtain $\bar{n}_t^+$ and $\bar{n}_t^-$.
   (b) Break if $t \geq 1/\delta$ or $\left(\sum_{i=1}^t \bar{n}_i^+ + \sum_{i=1}^t \bar{n}_i^-\right)/p \geq \bar{n}_1/2$.
   (c) Output $\tilde{n}_t = \left(\bar{n}_1 + \sum_{i=1}^t \bar{n}_i^+ - \sum_{i=1}^t \bar{n}_i^-\right)/p$.
5. Go to Step 2.

Lemma 8.3.1. *For any $v \in D_1$, the algorithm* CONTINUOUSFREQUENCYEPSILONAPPROX *provides an (ε,δ)-approximation of $|N_t^v|$.*

Proof. By the same arguments as in Lemma 8.2.3, we obtain an (ε,δ')-approximation of n_1. In any further time step we compute our estimate over the sum of all received messages ($\bar{n}_1$, arrivals and departures). If too many nodes change their measured value, we redo a complete estimation of the nodes in N_t^v.

Recall that $\tilde{n}_t$ is the random variable giving the estimated number of nodes by the algorithm, and $\tilde{n}_t^+ = \frac{\bar{n}^+}{p}, \tilde{n}_t^- = \frac{\bar{n}^-}{p}$ are the random variables giving the estimated arrivals and departures in that time step. We look at any time step $t > 1$ where the restart criteria are not met: Since $\tilde{n}_t = \tilde{n}_1 + \sum_{i=2}^t \left(\tilde{n}_i^+ - \tilde{n}_i^-\right)$ and the linearity of expectation, for any time $t \geq 1$ we can use a Chernoff bound as in Lemma 8.2.3 to show that the estimation is an (ε, δ')-approximation.

Using a union bound on the fail probability of up to $1/\delta$ time steps, we get a $1 - \frac{1}{\delta} \cdot \delta' = 1 - \delta$ probability of having a correct estimation in any time step. □

Lemma 8.3.2. *For a fixed value v and $T' = \min\{\frac{1}{2\sigma}, \frac{1}{\delta}\}$, $\sigma \leq \frac{1}{2}$, time steps,* CONTINUOUSFREQUENCYEPSILONAPPROX *uses $\Theta\left(\frac{1}{\varepsilon^2} \log \frac{1}{\delta}\right)$ messages on expectation.*

Proof. The message complexity depends on the initial size $|N_1^v|$ and on the number of nodes leaving and entering N^v in those time steps, which is bounded by σ. If FREQUENCYEPSILONFACTORAPPROX obtained a correct probability p in Step 1, i.e.,

$p = \Theta(\frac{1}{n_1})$, the expected number of messages (in case $p < 1$) is

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T'} \bar{n}_t \,\middle|\, p = \Theta\left(\frac{1}{n_1}\right)\right] &= \mathbb{E}\left[\bar{n}_1 + \sum_{i=2}^{T'} \bar{n}_i^+ + \bar{n}_i^- \,\middle|\, p = \Theta\left(\frac{1}{n_1}\right)\right] \\ &= \left(n_1 + \sum_{i=2}^{T'} n_i^+ + n_i^-\right) p \leq (n_1 + T'\sigma n_1)\, p \\ &= \Theta\left(\left(1 + \min\left\{\frac{1}{2\sigma}, \frac{1}{\delta}\right\}\sigma\right) \cdot \frac{1}{\varepsilon^2} \log \frac{1}{\delta}\right).\end{aligned}$$

Considering the case where FREQUENCYEPSILONFACTORAPPROX estimated wrong, the message complexity could increase greatly if the probability p is too large for the actual number of nodes (i.e., an underestimation leads to high message complexity). But the probability to misestimate by some constant factor (which would increase the message complexity by that factor) decreases exponentially in this factor (as shown in Lemma 8.2.4 for FREQUENCYEPSILONFACTORAPPROX), leaving the expected number of messages to be $\Theta\left(\left(1 + \min\left\{\frac{1}{2\sigma}, \frac{1}{\delta}\right\}\sigma\right) \cdot \frac{1}{\varepsilon^2} \cdot \log \frac{1}{\delta}\right) = \Theta\left(\frac{1}{\varepsilon^2} \log \frac{1}{\delta}\right)$. □

Theorem 8.3.3. *There exists an (ε,δ)-approximation algorithm for the Frequency Monitoring Problem using* $\Theta\left(|D_1|\,(1 + T \cdot \max\{2\sigma, \delta\})\,\frac{1}{\varepsilon^2} \log \frac{|D_1|}{\delta}\right)$ *messages in expectation for* T *consecutive time steps, if* $\sigma \leq 1/2$.

Proof. The algorithm works by first applying DOMAIN to obtain D_1 and then applying CONTINUOUSFREQUENCYEPSILONAPPROX(v, ε, $\delta/|D_1|$) for every $v \in D_1$. By Lemma 8.3.1 we know that in every time step and for all $v \in D_1$, the frequency of v is approximated up to a factor of ε with probability $1 - \delta/|D_1|$. We divide the T time steps into intervals of size $T' = \min\{\frac{1}{2\sigma}, \frac{1}{\delta}\}$ and perform CONTINUOUSFREQUENCYEPSILONAPPROX on each of them for every value $v \in D_1$. There are $\lceil \frac{T}{T'} \rceil \leq 1 + T \cdot \max\{2\sigma, \delta\}$ such intervals. For each of those, by Lemma 8.3.2 we need $\Theta\left(\left(1 + \min\{\frac{1}{2\sigma}, \frac{1}{\delta}\}\sigma\right) \cdot 1/\varepsilon^2 \log \frac{|D_1|}{\delta}\right)$ messages on expectation for each $v \in D_1$. This yields a complexity of $\Theta\left(|D_1|\,(1 + T \cdot \max\{2\sigma, \delta\})\,\frac{1}{\varepsilon^2} \log \frac{|D_1|}{\delta}\right)$ due to $\min\{\frac{1}{2\sigma}, \frac{1}{\delta}\}\sigma \leq \frac{1}{2\sigma} \cdot \sigma = \Theta(1)$. Using a union bound over the fail probability for every $v \in D_1$, a success probability of at least $1 - \frac{|D_1|\delta}{|D_1|} = 1 - \delta$ follows. □

By Theorem 8.2.5, trivially repeating the single step algorithm FREQUENCYEPSILONFACTORAPPROX needs $\Theta\left(T|D_1|\frac{1}{\varepsilon^2} \log \frac{|D_1|}{\delta}\right)$ messages on expectation for T (because the number of nodes in N_t^v for any $v \in D_1$ is at least $N_1^v/2$ in every time step of that interval). Hence, the number of messages sent when using CONTINUOUSFREQUENCYEPSILONAPPROX is reduced in the order of $\max\{2\sigma, \delta\}$.

Chapter 9

A Communication-Efficient Data Structure for Top-k and k-Select Queries

In this chapter we consider the rank related problems of Top-k and k-Select. In contrast to all previous chapters, we generalize the considered setting: Only when there is a query for the Top-k or for k-Select, is the output determined. We allow the parameters to be different from query to query and furthermore we allow for multiple k-Select queries at the same time step.

The approach in this chapter is orthogonal to the filter-based approach in Chapter 4. Using filters, the output is 'preserved' and the protocol uses less communication by finding a new certificate that the output is correct. Assuming the output changes between each time step, the filter-based approach has no benefit upon the naive algorithm. Intuitively speaking, here we consider the case that the Top-k changes between each query completely; i.e., the goal is to reuse parts of previous computations to compute the current output not from scratch.

Our results are based on the idea of keeping a (distributed) data structure which is used to answer a query and is informed about each update. More precisely, at every point in time, the data structure keeps track of an approximation of a data item with rank k. These approximations can be exploited by the protocols for a Top-k or k-Select computation to significantly decrease the communication and interestingly also the time bounds, making this approach a very powerful tool.

9.1 Introduction & Results

We present a distributed data structure for our Model with the following properties: In each step t, each client i receives a data item d_i^t as above. For ease of description let $s_1^t, \ldots, s_n^t$ be the *sorted* version of the data items $d_1^t, \ldots, d_n^t$ received at time step t. Our data structure supports the following operations:

STRONG SELECT: Output $d \in \{s^t_{(1-\varepsilon)k}, \ldots, s^t_{(1+\varepsilon)k}\}$
WEAK SELECT: Output d with $s^t_{k \cdot \log^{c_1} n} \leq d \leq s^t_{k \cdot \log^{c_2} n}$, with $c_1, c_2 > 1$

Our data structure gives the following *performance guarantees*:

- The expected amortized total communication cost for an update (amortized over all updates of the data items received by clients) is $\mathcal{O}(1/\text{polylog}\, n)$, the number of rounds is $\mathcal{O}(\log n)$.
- WEAK SELECT does not need any communication. The output is correct with probability at least $1 - 1/\text{polylog}\, n$.
- The expected total communication cost for a STRONG SELECT operation is bounded by $\mathcal{O}(1/\varepsilon^2 \log 1/\delta \; + \; \log^2 \log n)$, the expected number of rounds is $\mathcal{O}(\log \log \frac{n}{k})$. The output is correct with probability at least $1 - \delta$.
- The expected total communication cost for TOP-k is $\mathcal{O}(k + \log \log n)$, the expected number of rounds is $\mathcal{O}(\log \log n)$. The output is always correct.

(Recall that a one-shot computation of Top-k uses $\Theta(k + \log n)$ messages in expectation.) The protocols furthermore allow for a trade-off between the expected number of messages used and the number of communication rounds (i.e., time) to be executed. The bounds stated above represent the case in which the protocol is executed with a parameter aiming at minimizing the communication bounds.

Chapter Basis. Parts of the model, analysis, and results in the remainder of this chapter are based on the following publication:

- Biermeier, Feldkord, M., and Meyer auf der Heide. A Communication-Efficient Distributed Data Structure for Top-k and k-Select Queries. In: *15th International Workshop on Approximation and Online Algorithms (WAOA, 2017)*, [BFMM17].
- Feldkord, M., and Meyer auf der Heide. A Dynamic Distributed Data Structure for Top-k and k-Select Queries. In: *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, [FMM18].

9.2 Outline of the Data Structure

Our data structure maintains a *Sketch*(t) about the data items received at time t in the server, at every time step t as follows: As above, let $s_1^t, \ldots, s_n^t$ be the *sorted* version of the data items $d_1^t, \ldots, d_n^t$ received at time step t. Fix sufficiently large constants $c_1, c_2, c > 1$. A *Sketch*(t) is a subset of data items denoted by $\{r_1^t, \ldots, r_m^t\}$, where $m \leq \log n$. We call *Sketch*(t) correct if it consists of a set of data items $\{r_1, \ldots, r_m\}$ such that, for each $k = 1, \ldots, n$, there exists a r_j such that $s^t_{k \cdot \log^{c_1} n} \leq r_j \leq s^t_{k \cdot \log^{c_2} n}$ holds. We say the data item r_j is the representative of the set of data items d with $s_{k \cdot \log^{c_1} n} \leq d \leq s_{k \cdot \log^{c_2} n}$. *INIT* denotes the process of computing *Sketch*(t), where the input $d_1^t, \ldots, d_n^t$ of step t is given to the n sensor nodes.

Observation 9.2.1. *Consider a time step t at which the Init operation is called. A correct Sketch(t) is also a correct Sketch(t'), for a $t' > t$, if at most $\log^c(n)$ values of the clients are updated during the time interval $(t, t']$.*

This observation holds, because in the worst case the rank of a fixed data item, facing $\log^c n$ updates, can change by at most $\log^c n$. Since we allow the representatives to be upper bounded by $s^t_{k \cdot \log^{c_2} n}$ simply observe that this still holds after $\log^c n$ updates, for sufficiently large choices of constants. On the other hand, to prevent that the representative r_j does not become smaller than s_k^t, the data structure computes a data item $r_j > s^t_{k \cdot \log^{c_1} n}$. Note that the constants c_1 and c_2 depend on c. However, if the constants are (beforehand) chosen large enough, this ensures that after $\log^c n$ updates *Sketch*(t) is also a *Sketch*(t').

Lemma 9.2.2. *INIT is executed correctly with probability at least $1 - 1/\text{polylog}\, n$. It needs expected total communication of $\mathcal{O}(\log n)$ and $\mathcal{O}(\log n)$ rounds.*

We present the INIT algorithm and the necessary technical basis to prove this lemma in Section 9.3. We prove that the algorithm computes a *Sketch*(t) correctly in Theorem 9.3.6 and present the performance guarantees in Theorem 9.3.9.

The next operation *UPD* denotes the process of updating *Sketch*(t), in response to the updates of data items received in step t.

Lemma 9.2.3. *UPD can be done using expected amortized (w.r.t. number of updates of data items in the nodes) total communication of $\mathcal{O}(1/\text{polylog}\, n)$, the amortized number of rounds is constant (assuming each update is processed at a different time step). For every step t, the computed Sketch(t) is correct with probability at least $1 - 1/\text{polylog}\, n$.*

The UPD algorithm is presented in Section 9.4. In this section, we shortly argue its correctness in Lemma 9.4.1 and show communication bounds in Lemma 9.4.2.

By definition of a correct *Sketch*(t), the following observation holds.

Observation 9.2.4. *Given a correct Sketch*(t)*,* WEAK SELECT *can be executed without any communication. It is correct with probability* $1 - 1/\text{polylog}\, n$*.*

We present this observation shortly in Section 9.5.

Lemma 9.2.5. *Given a correct Sketch*(t)*,* STRONG SELECT *can be correctly computed with probability* $1 - \delta$*. It needs* $\mathcal{O}(1/\varepsilon^2 \log 1/\delta + \log^2 \log n)$ *communication and* $\mathcal{O}(\log \log n)$ *communication rounds.*

This result is considered in Section 9.6. The algorithm is based on three phases which are analyzed independently. The main result of this section is presented in Theorem 9.6.8.

Lemma 9.2.6. *Given a correct Sketch*(t)*,* TOP-k *can be computed using expected total communication of* $\mathcal{O}(k + \log \log n)$ *and* $\mathcal{O}(k + \log \log n)$ *communication rounds. The output is always correct.*

The TOP-k algorithm is presented in Section 9.6.2. On the total communication is argued in Lemma 9.6.10 and the number of rounds in Lemma 9.6.11.

The lemmata above imply the performance guarantees formulated in the previous section.

9.3 Initialization of the Data Structure

We start the presentation of our results with the goal to prove the first lemma. We propose the algorithm INIT which computes the *Sketch*(t) at a time step t. Since (a variation of) this algorithm is reused in later sections, we describe a procedure CFS (ConstantFactorSelect) with different parameters (see Algorithm 13).

Protocol Description. The high-level idea of CoFaSel is as follows: Initially each node is defined to be active. The protocol samples a node uniformly at random and broadcasts its value. All nodes with a larger data item deactivate themselves. This process is repeated until the remaining nodes are sampled with probability 1.

However, since the server does not know which nodes remain active, a sample cannot be chosen directly. Instead, we let the nodes proceed a random process such that the server can probe each node with a certain outcome on the basis of the random process. We consider this random process in more detail: Each node i chooses a height h_i from a geometric distribution; i.e., the number of coin flips with success probability

p until one coin flip was successful. (Observe that based on the definition of p, the expected maximal height $\max_i h_i$ varies. This fact can be used to trade-off between the expected number of messages and the number of rounds the algorithm uses.)

Intuitively speaking, we build a distributed (not binary) searchtree in which the heights are chosen randomly and the algorithm follows the path to the sensor node observing the minimum value. The $\log n$ nodes on this path yield an approximation of the data items with respect to their ranks. We will exploit this fact and show that a data item at a specific level can be used to approximate a given rank successfully.

Algorithm 13 INIT()

CoFaSel($\phi, h_{\max}, k$) [ConstantFactorSelect]

1. Each node i defines a random variable h_i, i.i.d. drawn from a geometric distribution with $p = (1 - \phi)$, and redefines $h_i := \min\{h_i, h_{\max}\}$.
2. Server defines $d_{\min} := \infty$, keeps a set $S := \emptyset$, and initializes $cnt := 0$.
3. **if** $k = 1$ **then** $h_{\min} := 1$.
4. **else** $h_{\min} := \lfloor \log_{1/\phi}(7k) \rfloor + 1$. (let $\alpha := \log_{1/\phi}(7k) - \lfloor \log_{1/\phi}(7k) \rfloor$)
5. **for** $h := h_{\max}$ **to** $h_{\min}$ **do**
6. Server probes all nodes i with $d_i < d_{\min}$ and $h_i = h$.
7. Let $a_1 < a_2 < \ldots < a_j$ be the responses, ordered by their values.
8. The representative $r_h := a_1$ is added to S as tuple (a_1, h).
9. **if** $h > h_{\min} > 1$ **then** Server redefines $d_{\min} := a_1$ **else** $d_{\min} := a_{(1/\phi)^\alpha}$.
10. **output** $d_{\min}$

INIT()

1. **call** CoFaSel$(\phi := \frac{1}{2}, h_{\max} = \log n, k = 1)$.

Initialize. We only need a simple variant of the CoFaSel protocol as follows: The INIT operation defines $p := 1/2$, the success probability of each coin flip. That is, each sensor node has a height of 2 in expectation. Thus, observe that the expected maximum of n nodes is $h_{\max} = \log n$. For each height h the server keeps the smallest response of the sensor nodes in the data structure.

Correctness: Initialize Computes *Sketch*(t). Recall that the data structure is asked to answer each request for a data item of rank k by a representative r, where r groups a set of requests with different ranks with the same response. To this end, we divide the ranks $1, \ldots, n$ into classes $C_1, \ldots, C_m$, where m is chosen sufficiently large such that each data item belongs to a class. The exact number of classes is based on a constant that is defined by the analysis; however, note that $m = \mathcal{O}(\log n)$ holds.

We define a representative for each class which is the response for a request of any rank in the next-smaller class. Furthermore, the height of a class represents the expected maximum height found within this class, such that our representative will have a height value within the noted bounds. In the following we use the constant $\kappa > 1$ chosen sufficiently large which represents the constants in the bounds on the precision and the success probability. Furthermore, let $\mathcal{H} := \log_{1/\phi}(\log(n))$ to ease the notation. The idea of classes is captured in the following definition:

Definition 9.3.1 (Classes)**.** *A* Class C_ℓ^t *consists of all data items* d_j^t *with* $rank(d_j^t) \in [\log^{6\ell\kappa}(n), \log^{6(\ell+1)\kappa}(n))$. *We define* $h_{\min}(C_\ell) := (6\ell\kappa + 1\kappa)\mathcal{H}$ *and* $h_{\max}(C_\ell) := (6\ell\kappa + 7\kappa)\mathcal{H}$. *The height of the class* C_ℓ^t *is given by* $h(C_\ell^t) := (h_{\min}(C_\ell^t), h_{\max}(C_\ell^t)]$.

By abuse of notation we introduce $d_i^t \in C_\ell^t$ which shortens $rank(d_i^t) \in C_\ell^t$. Furthermore let $class(d)$ be the class where the data item d belongs to; i.e., for a given d, $class(d)$ gives the class C_ℓ^t such that $d \in C_\ell^t$ holds.

Definition 9.3.2 (Inner Classes)**.** *We denote by an* inner class $I_{\ell,\tau}^t$ *(where* $\tau \in \{0, 1, 2\}$ *holds) the set of data items* d_i^t *with a rank between* $\log^{6\ell\kappa+2\kappa}(n)$ *and* $\log^{6\ell\kappa+4\kappa}(n)$. *The height of* $I_{\ell,\tau}^t$ *is* $h(I_{\ell,\tau}^t) = ((6\ell\kappa + (2\tau + 1)\kappa)\mathcal{H}, (6\ell\kappa + (2\tau + 3)\kappa)\mathcal{H}]$.

We omit the time step t in our notation whenever it is clear from the context.

Definition 9.3.3 (Well-Shaped)**.** *The data items in an inner class* $I_{\ell,\tau}$ *are* well-shaped *if for each data item* $d_i \in I_{\ell,\tau}$ *it holds* $h_i \leq (6\ell\kappa + (2\tau + 3)\kappa)\mathcal{H}$.

We start by analyzing the outcome of the INIT operation. That is, we show that a class is well-shaped with sufficiently large probability in Lemma 9.3.4 and argue that the data structure has one representative in Theorem 9.3.6, afterwards.

Lemma 9.3.4. *After an execution of* INIT*, the inner class* $I_{\ell,\tau}$ *is well-shaped with probability at least* $1 - \log^{-\kappa}(n)$.

Proof. Recall that for a fixed data item d_i and sensor node i the probability for $h_i > h$ is ϕ^h. Fix an inner class $I_{\ell,\tau}$ and consider the data items $d_i \in I_{\ell,\tau}$. We upper bound

the probability that there is a data item with a height of at least h with $h := (6\ell\kappa + (2\tau+3)\kappa)\mathcal{H}$ by applying the union bound as follows:

$$\begin{aligned}\Pr[\exists d_i \in C_{\ell,\tau} \mid h_i > h] &\leq \left(\log^{6\ell\kappa+(2\tau+2)\kappa}(n) - \log^{\ell 8\kappa+2\tau\kappa}(n)\right) \cdot \phi^h \\ &\leq \log^{6\ell\kappa+(2\tau+2)\kappa}(n) \cdot \log^{-(6\ell\kappa+(2\tau+3)\kappa)}(n) \\ &\leq \log^{-\kappa}(n)\end{aligned}$$

□

Lemma 9.3.5. *Consider the inner class $I_{\ell,1}$. There is a data item $d_i \in I_{\ell,1}$ with $h_i > (6\ell\kappa + 3\kappa)\mathcal{H}$ with high probability.*

Proof. Here we simply upper bound the probability that each data item in the inner class has a height of at most h as follows:

$$\begin{aligned}\Pr[\forall d_i \in I_{\ell,1} \mid h_i \leq (6\ell\kappa + 3\kappa)\mathcal{H}] &\leq \left(1 - 2^{-(6\ell\kappa+3\kappa)\mathcal{H}}\right)^{|I_{\ell,1}|} \\ &\leq \left(1 - \log^{-(6\ell\kappa+3\kappa)} n\right)^{\log^{6\ell\kappa+4\kappa} n - \log^{6\ell\kappa+2\kappa} n} \\ &\leq \left(\frac{1}{e}\right)^{\frac{1}{2}\log^{\kappa} n} \leq n^{-\frac{1}{2}\log(e)\log^{\kappa-1}(n)} \leq n^{-c},\end{aligned}$$

for some constant c. □

We can now prove the first part of Lemma 9.2.2, i.e., that INIT computes a correct *Sketch*(t). Technically, we show a result which is more restricted than the stated precision of Lemma 9.2.2, as follows:

Theorem 9.3.6. *After execution of INIT there exists, for each rank k, a data item in Sketch(t) with rank between $k \cdot \log^{2\kappa}(n)$ and $k \cdot \log^{10\kappa}(n)$ with probability at least $1 - \log^{-\kappa+2}(n)$.*

Proof. First consider a fixed inner class $I_{\ell,\tau}$ for a fixed $\ell \in \mathbb{N}$ and $\tau \in \{0,1,2\}$. On the basis of Lemma 9.3.4 we can show that the distribution of the random heights is well-shaped with a probability at least $1 - \log^{-\kappa}(n)$. Now, with high probability there is a data item with such a height for sufficiently large κ and n due to Lemma 9.3.5. These observations together show that there is a data item d identified and stored in DS with probability at least $1 - \log^{-\kappa+1}(n)$.

Furthermore, note that the number of inner classes is upper bounded by $\log n$. The argument stated above applied to each class leads to the result that for each inner class

there exists a data item in the data structure, and each inner class is well-shaped with probability at least $1 - \log^{\kappa-2}(n)$ (by simply applying the union bound). □

Communication Bounds. In the following we show the second part of Lemma 9.2.2, i.e., that the number of messages used by INIT is upper bounded by $\mathcal{O}(\log n)$ and the same bound holds for the number of rounds. We start by analyzing the bound on the total communication.

We show an upper bound on the communication used by the COFASEL protocol analyzing the expected value of a mixed distribution: Intuitively speaking, consider the path from the root to the maximum in a non-binary searchtree. For each node i on the path consider the number of siblings j with a smaller data item, i.e., $d_j < d_i$. To bound the expected number of such siblings j, we first consider on a fixed height h the number of tries G_h until the first node j' has drawn a height $h_{j'} > h$ (for each height h this results in the geometric sequence, Definition 9.3.7). On the basis of G_h, we consider the number of nodes that have drawn precisely the height $h_{j'} = h$ (for each height h, the geocoin experiment, Definition 9.3.8).

Note that this analysis turns out to be very simple since independence can be exploited in a restricted way and leads to a proper analysis with respect to small constants. Furthermore, note that we used this notation to analyze the number of messages used by the TOP-kPROTOCOL and present it here for the reason of self containment.

Definition 9.3.7. *We call a sequence $G = (G_1, \ldots, G_m)$ of m random experiments a* geometric sequence *if each G_h is chosen from a geometric distribution with $p_h^{geo} := \phi^h$. We denote its $size(G) := \sum_h G_h$ and say it* covers *all nodes if $size(G) \geq n$.*

For the analysis, we choose a fixed length of $m := \log_{1/\phi}(n)$ and modify G to $G' = (G_1, \ldots, G_{m-1}, n)$ such that G' covers all nodes with probability 1.

On the basis of a given geometric sequence, we define a sequence describing the number of messages sent by the nodes on a given height. We take the number of nodes G_j as a basis for a Bernoulli experiment where the success probability is the probability that a node sends a message on height h_j. This is $\Pr[h = h_j \mid h \leq h_j] = \frac{\phi^{h-1}(1-\phi)}{1-\phi^h}$.

Definition 9.3.8. *We denote a* geocoin experiment *by a sequence $C = (C_1, \ldots, C_m)$ of random variables C_h which are drawn from the binomial distribution $Binom\big(n = G_h, p_h^{bin} = \frac{\phi^{h-1}(1-\phi)}{1-\phi^h}\big)$; i.e., C_h out of G_h successful coin tosses and each coin toss is successful with probability p_h^{bin}.*

We are now prepared to prove the second part of Lemma 9.2.2, i.e., a bound on the total communication for CFS and thus for INIT.

Theorem 9.3.9. *Let $h_{\max} \geq \log_{1/\phi}(n)$ hold. The* CoFaSel *protocol uses an expected number of $\frac{1-\phi}{\phi}\log_{1/\phi}(n) + \frac{1}{\phi}$ messages in total.*

Proof. The number of messages sent is upper bounded by a geocoin experiment C. Let $\mathcal{H} \coloneqq \log_{1/\phi}(n)$. For $h < \mathcal{H}$ we use that the geometric distribution is memory-less and hence

$$\mathbb{E}[C_h] = (1 - p_h^{geo}) \cdot (p_h^{bin} + \mathbb{E}[C_h]) = (1 - \phi^h) \cdot \left(\frac{\phi^{h-1}(1-\phi)}{1-\phi^h} + \mathbb{E}[C_i] \right) \ .$$

This can simply be rewritten as $\mathbb{E}[C_h] = \frac{1-\phi}{\phi}$.

For $h \geq \mathcal{H} = \log_{1/\phi}(n)$ we bound the number of messages by the total number of nodes with height at least $\mathcal{H}$. These can be described as the expectation of a Bernoulli experiment with n nodes and success probability $\phi^{\mathcal{H}-1}$ and hence we can bound $\mathbb{E}[C_{\geq \mathcal{H}}] \leq \phi^{\mathcal{H}-1} \cdot n = \frac{1}{\phi}$.

In total, we get

$$\sum_h \mathbb{E}[C_h] = \left(\sum_{h=1}^{\mathcal{H}-1} \mathbb{E}[C_i] \right) + \mathbb{E}[C_{\geq \mathcal{H}}] \leq \frac{1-\phi}{\phi} \log_{1/\phi}(n) + \frac{1}{\phi} \ ,$$

concluding the proof. □

We conclude the proof for the first lemma by this simple observation on the number of rounds. By the definition of the protocol it is easy to see that the server simply sends a broadcast message for each height h and receives a message by those nodes which fulfill a specific rule. Since the server can process each received message in the same round, $h_{\max}$ is obviously a strict upper bound for the number of rounds.

Observation 9.3.10. *The* INIT *operation uses at most $h_{\max} = \log n$ rounds.*

9.4 Update

To keep the data structure up to date we apply the following simple straightforward strategy: As long as there are fewer than $\log^c n$ updates processed since the last call of INIT, the precision of the approximated ranks can also only differ by an additional $\mathcal{O}(\log^c n)$ (for a predefined constant $c > 1$). We apply a simple standard counting technique to verify that the current number of processed UPD operations is $\mathcal{O}(\log^c n)$ in expectation. If more UPD operations are identified, the current data items in the data structure are discarded and the *Sketch*(t) is built from scratch.

Algorithm 14 UPD(i, d) [Executed by node i]

1. Update d_i^t by $d_i^{t+1} := d$.
2. Flip a coin with probability $p_{cnt} = \frac{c}{\log^\kappa(n)} \cdot \log n$.
3. **if** the coin flip was successful **then**
4. send a message to the server; increase cnt.
5. **if** $cnt = c \cdot \log n$ holds **then** [Executed by Server]
6. restart the protocol, i.e., **call** INIT()

Consider the protocol for the UPD operation as presented in Algorithm 14. It applies a randomized counting technique to identify that there are more than $\Theta(\log^\kappa n)$ updates since the last INIT operation. It is easy to verify by applying standard Chernoff bounds that the protocol identifies $cnt \leq 2\,c\,\log n$ with high probability. Thus, and applying a Chernoff bound again, it follows that the number of UPD operations that took place since the last INIT operation is upper bounded by $2\,\log^{2c} n$ with high probability. With this, we can show the first part of Lemma 9.2.3.

Lemma 9.4.1. *After the last call of INIT, there are at most $\Theta(\log^\kappa n)$ UPD operations processed with high probability.*

The UPD operation sends a message with probability p_{cnt}, so it is easy to verify that the expected number of messages sent is upper bounded by p_{cnt}.

Now consider a sufficiently large instance (i.e., sufficiently many UPD operations). Assume that for a time step t at which INIT is called, t' denotes the next time step at which INIT is called to rebuild the data structure. Observe that $\mathcal{O}(\log n)$ messages where sent during $[t, t']$ by UPD and INIT operations in total. Since $\Omega(\log^{c/2} n)$ UPD operations where called with high probability, the amortized bound for one single UPD operation follows.

For the number of communication rounds, consider the same interval $[t, t']$ as described above. Since one execution of UPD uses a constant number of rounds (excluding the call of INIT) and the INIT operation is called a constant number of times, each UPD operation only uses an amortized constant number of rounds. These observations conclude the argumentation for the second part of Lemma 9.2.3:

Lemma 9.4.2. *The UPD operation uses $\mathcal{O}(1/\text{polylog}\, n)$ messages in expectation and amortized $\mathcal{O}(1)$ number of rounds.*

9.5 Weak Select

For the sake of a complete presentation we shortly describe how the *Sketch*(t) is used to answer a weak approximate k-Select request.

The WEAK SELECT operation simply identifies the class ℓ in which the data item d with rank k is expected (see Algorithm 15). Then, the representative r in the class on level $\ell + 1$ is chosen.

Algorithm 15 WEAK SELECT(k)

1. Determine ℓ such that $k \in C_\ell$ holds.
2. **output** representative $r \in I_{\ell+1}$.

Note that by the correctness of INIT and its analysis on the precision the correctness of the protocol follows. It is also easy to see that the protocol is executed by the server and thus does not need any further communication to the sensor nodes. Since no further argumentation is needed, we restate the following observation for completeness:

Observation 9.5.1. *Given a correct Sketch*(t)*, WEAK SELECT can be executed without any communication. It is correct with probability* $1 - 1/\text{polylog}\, n$.

9.6 Strong Approximate k-Select

In this section we present an algorithm which gives an (ε, δ)-approximation for the k-Select problem; i.e., a data item d is identified with a rank between $(1-\varepsilon)k$ and $(1+\varepsilon)k$ with probability at least $1-\delta$. In other words, we propose an algorithm and analyze its performance guarantee as claimed in Lemma 9.2.5.

Algorithm 16 STRONG SELECT(ϕ, k, ε, δ)

1. **call** WEAK SELECT(k) and
   denote by (d', h') the returned data item and its height.

   [Phase 1]
2. Determine ℓ' such that $k \cdot \log^c n \in C_{\ell'}$ holds.
3. **repeat** until a data item d'' is found (i.e., $d'' \neq$ nil)
4. Each node i with $d_i \leq d'$ executes:
5. Call CFS(ϕ, h, $k \cdot \log^c n$) and let (r, h_r) be the data item and height.
6. **if** $h_r \in h(C_{\ell'})$ **then** $d'' \coloneqq r$.

   [Phase 2]
7. **for** $j = 1, \ldots, c \log 1/\delta'$ **do in parallel**
8. Each node i with $d_i \leq d''$ executes:
9. **call** CoFaSel(ϕ, h_r, k) on the active nodes and let (d_j^*, h_j') the output.
10. $d^* \coloneqq Median(d_1^*, \ldots, d_{c \log 1/\delta'}^*)$

    [Phase 3]
11. Each node i with $d_i < d^*$ executes:
12. Toss a coin with $p \coloneqq \min\left(1, \frac{c}{k} \cdot \mathcal{S}_{\varepsilon,\delta}\right)$.
13. On success send d_i to the server.
14. The server sorts these values and outputs $d_{\tilde{k}}$, the $p \cdot k$-th smallest item.

Algorithm Description. We apply a standard sampling technique to select a data item as required. However, the data item given by the WEAK SELECT operation is too weak to directly be followed by a sampling technique (cf. Phase 3 in Algorithm 16). Thus, we add the following two phases:

1) A data item d' is identified, such that a polylogarithmic error bound holds with high probability. It might be that a large number of sensor nodes (i.e., $\omega(k \cdot$

polylog n)) 'survive' till the last phase and apply the costly sampling technique. With this step the event only occurs with probability at most $1/n$.

2) The second phase applies $c \log \frac{1}{\delta'}$ calls of CFS to identify data items that have a rank between k and $42\,k$ with constant probability each. This number of calls is to amplify the (success) probability that the final data item d^* has a rank between k and $42\,k$ to at least $1 - \delta'$.

Note that the internal probability δ' will be defined as $1/\text{polylog}\, n$ which is a result of the analysis. Important is that the calls of CFS do not change the information of *Sketch*(t) stored in the data structure. Here, they are only used 'read-only' and are not overwritten.

Analysis Outline. We split our analysis and consider each phase separately. First, we show that Phase 1 determines a data item d' with a rank which is by a polylogarithmic factor larger than k with high probability. This needs $\mathcal{O}(\log \log n)$ messages and $\mathcal{O}(\log \log n)$ rounds in expectation.

Afterwards, we consider Phase 2 which determines a data item d'' with a rank only a constant factor larger than k with probability at least $1 - \delta'$, where δ' can be chosen arbitrarily small.

Finally, Phase 3 applies a sampling technique to determine the final data item d which yields the property as required by Lemma 9.2.5.

We use the notation as given in the protocol and use d' to denote the data item given by the WEAK SELECT operation, d'' the data item determined by Phase 1, and d^* the data item given by Phase 2. We do not need any further analysis for the property of data item d' since we analyzed its precision (and the given success probability bounds) in the past sections.

Analysis of Phase 1. We consider Phase 1 of the STRONG SELECT operation and analyze the precision of the rank of item d'', the expected number of messages and the number of communication rounds.

Lemma 9.6.1. *For a given constant c, there exist constants c_1, c_2, such that Phase 1 as given in Algorithm 16 outputs a data item d'' with a rank between $7k \cdot \log^{c_1}(n)$ and $7k \cdot \log^{c_2}(n)$ with probability at least $1 - n^{-c}$.*

Proof. We use a simple argument to argue on the probability to obtain a data item within a multiplicative polylogarithmic precision bound:

Consider the event that the rank is strictly smaller than $7k \cdot \log^{c_1} n$. Thus, one node i of the $7k \log^{c_1} n - 1$ nodes has drawn a height $h_i \geq \log(7k \log^{c} n)$. We show (by applying Chernoff bounds) that this probability is upper bounded by $n^{-c'}$, where c' depends on c and c_1. For the remaining case (i.e., the rank is strictly larger than $7k \cdot \log^{c_2} n$) the same argument is applied.

Let X denote the rank of the data item d'' which is identified by STRONG SELECT. Now let X_1 be drawn from $Binom(n = 7k \log^{c_1} n, p = (1/2)^{\log(7k \log^{c} n)})$, and let $X_2 \sim Binom(n = 7k \log^{c_2} n, p = (1/2)^{\log(7k \log^{c} n)})$. Observe that it holds $\gamma_1 = \mathbb{E}[X_1] = \log^{c_1 - c} n$ and $\gamma_2 = \mathbb{E}[X_2] = \log^{c_2 - c} n$. Thus, $\Pr[X < 7k \log^{c_1} n] \leq \Pr[X_1 > (1 + (1/2) \log^{c - c_1} n) \cdot \log^{c_1 - c} n] \leq \exp(-\frac{1}{12}(\log^{c - c_1} n)) \leq n^{-\frac{c - c_1 - 1}{12}}$.

We obtain by the same argument similar results for the probability of the event that the rank is larger than the claimed bound. □

Lemma 9.6.2. *Phase 1 uses an amount of $\mathcal{O}(\log \log n)$ messages in expectation.*

Proof. We apply the law of total expectation and first consider the event that WEAK SELECT is successful. Afterwards, the number of messages for a failed call is considered.

First, consider the case that the WEAK SELECT operation is successfully within the precision bounds. Then, $\mathcal{O}(\log \log n)$ messages on expectation are used in this phase. On the other hand, consider the number of messages used if the number of nodes that take part in this phase is n. Then, the protocol needs $\mathcal{O}(\log n)$ messages. However, the probability that WEAK SELECT is not within these bounds is $1/\text{polylog}\, n$ which concludes the proof. □

To upper bound the time needed for Phase 1, simply determine the range of h and observe that this range is bounded by $\mathcal{O}(\log \log n)$. Since one data item is found with probability at least $1 - 1/\text{polylog}\, n$, in expectation after the second repetition a data item is found.

Observation 9.6.3. *Phase 1 of Algorithm 16 uses $\mathcal{O}(\log \log n)$ number of rounds.*

Analysis of Phase 2. Now consider one execution of the lines 7 to 10 as given in Algorithm 16 (and restated in Algorithm 17).

Lemma 9.6.4. *One execution of lines 8 and 9 of Phase 2 in Algorithm 16 outputs a data item d with $rank(d) \in [k, 42k]$, with probability at least 0.6.*

Proof. The algorithm outputs the $(1/\phi)^{\alpha}$ smallest data item d_j^* the server is as a response on height $h = h_{\min}$. To analyze its rank, simply consider the random number

Algorithm 17 STRONG SELECT [Phase 2 restated]

7. **for** $j = 1, \ldots, c \log 1/\delta'$ **do in parallel**
8. Each node i with $d_i \leq d''$ executes:
9. **call** CoFaSel(ϕ, h_r, k) on the active nodes and let (d_j^*, h_j') the output.
10. $d^* := Median(d_1^*, \ldots, d_{c \log 1/\delta'}^*)$

X of nodes i that observed smaller data items $d_i < d$. The claim follows by simple calculations: (i) $\Pr[X < k] \leq \frac{1}{5}$ and (ii) $\Pr[42k > X] \leq \frac{1}{5}$.

The event that X is (strictly) smaller than k holds if there are at least $(1/\phi)^\alpha$ out of k nodes with a random height at least h_{min}. Let X_1 be drawn by a binomial distribution $Binom(n = k, p = \phi^{h_{\text{min}}-1})$. It holds $\mathbb{E}[X_1] = k \cdot \phi^{h_{\text{min}}-1} = \frac{1}{7} \cdot (\frac{1}{\phi})^\alpha$. Then, $\Pr[X < k] \leq \Pr[X_1 \geq (\frac{1}{\phi})^\alpha] = \Pr[X_1 \geq (1+6)\frac{1}{7\phi^\alpha}] \leq \exp(-\frac{1}{3}\frac{1}{7\phi^\alpha}6^2) \leq \frac{1}{5}$.

On the other hand, the event that X is (strictly) larger than $42k$ holds if there are fewer than $(1/\phi)^\alpha$ out of $42k$ nodes with a random height of at least h_{min}. Let X_2 be drawn by a binomial distribution $Binom(n = 42k, p = \phi^{h_{\text{min}}-1})$. It holds $\mathbb{E}[X_2] = (42k)\phi^{h_{\text{min}}-1} = (42k)(7k)^{-1}\phi^{-\alpha} = \frac{6}{\phi^\alpha}$. Then, $\Pr[X > 42k] \leq \Pr[X_2 < \frac{1}{\phi^\alpha}] = \Pr[X_2 < (1-(1-\frac{1}{6}))\frac{6}{\phi^\alpha}] \leq \exp(-\frac{1}{2}(\frac{6}{\phi^\alpha}(1-\frac{1}{6})^2) \leq \exp(-\frac{25}{12}) \leq \frac{1}{5}$. □ □

Note that we apply a standard boosting technique, i.e., we use $\mathcal{O}(\log \frac{1}{\delta'})$ independent instances, and consider the median of the outputs of all instances to be the overall output (cf. Algorithm 17). Thus, an output in the interval $[k, 42\,k]$ is determined with probability at least $1 - \delta'$.

Observation 9.6.5. *Phase 2 of Algorithm 16 outputs a data item d^* with $rank(d^*) \in [k, 42k]$ with probability at least $1 - \delta'$.*

Lemma 9.6.6. *Assume $\delta' \geq n^{-c}$ for a constant $c > 1$. The second phase of Algorithm 16 uses $\mathcal{O}(\log \frac{1}{\delta'} \cdot \log\log \frac{n}{k})$ messages in expectation.*

Proof. Consider one instance of Phase 2 and applying arguments from Theorem 9.3.9, the algorithm uses $\mathcal{O}(\log\log \frac{n}{k})$ messages in expectation for each iteration of Steps 8 and 9. This number of messages is multiplied by $\mathcal{O}(\log \frac{1}{\delta'})$, since we apply this number of executions in parallel.

It remains to show that the parallel execution does not need further messages to separate each execution from the others: In more detail, each instance of Steps 8 and 9 has to be executed with an additional identifier. Since $\delta' \leq n^{-c}$ holds, the identifier has a range of integer numbers between 1 and $\mathcal{O}(\log n)$ and thus needs additional $\mathcal{O}(\log\log)$ bits. Since a machine word has a size of $\mathcal{O}(\mathcal{B} + \log n)$ the identifier can be

Algorithm 18 STRONG SELECT [Phase 3 restated]

11. Each node i with $d_i < d^*$ executes:
12. Toss a coin with $p \coloneqq \min\left(1, \frac{c}{k} \cdot \mathcal{S}_{\varepsilon,\delta}\right)$.
13. On success send d_i to the server.
14. The server sorts these items and outputs $d_{\tilde{k}}$, the $p \cdot k$-th smallest item.

added to the message (or sent as a separate message such that the number of messages has a constant overhead). This concludes the proof. □

Since we run the $\mathcal{O}(\log \frac{1}{\delta'})$ instances in parallel, and the server is able to process all incoming messages within the same communication round, the number of communication rounds does not increase by the parallel executions.

Observation 9.6.7. *Phase 2 of Algorithm 16 uses $\mathcal{O}(\log \log n)$ rounds.*

Analysis of Phase 3 We are now prepared to propose the last phase of the algorithm which fulfills the required precision as stated in Lemma 9.2.5.

We consider the final phase of the algorithm, i.e., we apply a standard sampling technique (cf. Algorithm 18): The server broadcasts the value d^* which (as a result of the analysis of Phase 2) has a rank between k and $42\,k$ with probability at least $1-1/\text{polylog}\, n$. Each node i compares its data item d_i^t with d^* and only takes part in the sampling process if and only if $d_i^t \leq d^*$ holds. Then, with probability $p = \frac{c}{k}\frac{1}{\varepsilon^2}\log\frac{1}{\delta}$ node i sends its data item to the server. In turn, the server sorts each data item and outputs the $p \cdot k$-th smallest item, which has a rank of k in expectation.

For the sake of readability we introduce the notation $\mathcal{S}_{\varepsilon,\delta} \coloneqq \frac{1}{\varepsilon^2}\log\frac{1}{\delta}$ and are now prepared to show Lemma 9.2.5:

Theorem 9.6.8. *Define $\delta' \coloneqq 1/\text{polylog}\, n$. The* STRONG SELECT *operation (as presented in Algorithm 16) selects a data item $d_{\tilde{k}}$ with a rank in $[(1-\varepsilon)\,k, (1+\varepsilon)\,k]$ with probability at least $1-\delta$ using $\mathcal{O}(\mathcal{S}_{\varepsilon,\delta} + \log^2 \log n)$ messages in expectation and $\mathcal{O}(\log_{1/\phi} \log n)$ communication rounds.*

Proof. From Lemma 9.6.6 we obtain that Phase 2 of the protocol uses an amount of at most $\mathcal{O}(\log\log\frac{n}{k}\log\frac{1}{\delta'})$ messages in expectation and runs for $\mathcal{O}(\log\log n)$ communication rounds. The remaining steps of Algorithm 16 need only one additional communication round and thus the stated bound on the communication rounds follows. We omit the proof for the correctness of the algorithm; i.e., with demanded probability the k-th smallest data item is approximated, since it is based on a simple argument using Chernoff bounds.

It remains to show the upper bound on the number of messages used. Formally, we apply the law of total expectation and consider the event that Phase 2 of Algorithm 16 determined a data item d^* with rank $k \leq rank(d^*) \leq 42k$ and the event $rank(d^*) > 42k$.

Observe that the sampling process in steps 2 and 3 yields $\mathcal{O}(\frac{rank(d^*)}{k}\mathcal{S}_{\varepsilon,\delta})$ messages in expectation. Consider the event that Phase 2 determined a data item d^* with rank $k \leq rank(d) \leq 42k$. Then, Phase 3 uses $\mathcal{O}(\mathcal{S}_{\varepsilon,\delta})$ messages in expectation. Now consider the event that Phase 2 determined a data item d^* with $d > 42\,k$. It uses $\mathcal{O}\left(\frac{\log^c n}{k}\mathcal{S}_{\varepsilon,\delta}\right)$ messages in expectation. Since the probability for this event is upper bounded by δ', the conditional expected number of messages is $\mathcal{O}\left(\frac{\log^c(n)}{k}\mathcal{S}_{\varepsilon,\delta} \cdot \delta'\right)$. Defining $\delta' \coloneqq \log^{-c} n$ the bound follows as claimed. □

9.6.1 One-Shot Approximate k-Select

For the sake of self-containment we propose a bound which considers all nodes to take part in the protocol.

Corollary 9.6.9. *Let c be a sufficiently large constant. Furthermore, let $N = n$, $\phi \coloneqq \frac{1}{2}$, $h_{\max} \coloneqq \log n$, and $\delta' \coloneqq \frac{1}{\log^c(n)}$. The protocol uses an amount of at most $\mathcal{O}(\mathcal{S}_{\varepsilon,\delta} + \log n)$ messages in expectation and $\mathcal{O}(\log(\frac{n}{k}))$ expected rounds.*

This represents the case (with respect to the choice of ϕ) that a small number of messages and a large number of communication rounds are used. This observation is complemented by a lower bound of $\Omega(\log n)$.

9.6.2 Top-k

In this section we shortly discuss algorithms which identifies all k smallest data items currently observed by the sensor nodes, i.e., at a fixed time step t.

Note that by applying the MAXIMUMPROTOCOL k times and using the *Sketch*(t) from our data structure, the problem can be solved using $\mathcal{O}(k \cdot \log\log n)$ messages in expectation and $\mathcal{O}(k \cdot \log\log n)$ rounds. By applying the STRONG SELECT operation from the previous section (denote the output by d_K) and selecting all of the nodes i with a data item $d_i \leq d_K$, a bound of $\mathcal{O}(k + \log^2 \log n)$ expected messages and $\mathcal{O}(\log\log n)$ rounds in expectation follows. These bounds are subject to be improved to $\mathcal{O}(k+\log\log n)$ expected messages and $\mathcal{O}(k+\log\log n)$ expected rounds. Without our *Sketch*(t) the algorithm needs $k + \log n + 2$ expected messages and $\mathcal{O}(k + \log n)$ expected rounds, which might be of independent interest. We show a more general result which allows to trade-off between number of messages and number of rounds. This translates to $k+\frac{1-\phi}{\phi}\log_{1/\phi} n+\frac{1}{\phi}$ expected total communication and $\mathcal{O}(\phi\cdot k+\log_{1/\phi} n)$ expected rounds for an arbitrarily chosen $1/n \leq \phi \leq 1/2$.

Analysis. To prove that there is a protocol which uses $\mathcal{O}(k + \log\log n)$ messages in expectation simply observe that the probability to send a message for a sensor node within the Top-k is 1. Consider the remaining nodes; i.e., consider the set V' of nodes that are not in the Top-k. To bound the number of messages, we simply upper bound the number of messages used to find the maximum within V'. Since WEAK SELECT gives a data item such that $k \cdot \log^{c_2} n$ nodes remain, and by the arguments in Theorem 9.3.6, it holds:

Lemma 9.6.10. *The TOP-k operation uses $\mathcal{O}(k+\log\log n)$ messages in expectation.*

We consider the number of rounds, which concludes the proof for Lemma 9.2.6.

Lemma 9.6.11. *The TOP-k operation uses at most $\mathcal{O}(k + \log\log n)$ exp. rounds.*

Proof. We structure the proof in two steps: First, we analyze the number of rounds used to determine the minimum (i.e., the data item with rank 1), and second, the number of communication rounds used to determine the Top-k.

Observe that the algorithm uses a linear amount of steps (linear in h), until it reaches $h_{\text{min}} = 1$, after which the minimum is found. Afterwards, in each step the algorithm recursively probes for nodes successively smaller than the currently largest values that are added to the output set S. Note that by the analysis in Theorem 9.3.9, the number of nodes that send a message in expectation in each round is $(1-\phi)/\phi$ (for

$h < \log_{1/\phi}(n)$). Thus, in each communication round there are $\Omega(\frac{1}{\phi})$ nodes in expectation that send a message, such that after an expected number of $\mathcal{O}(\phi \cdot k)$ rounds the TOP-KPROTOCOL terminates. □

Chapter 10

Fully Dynamic & Filter-Based Approximate Count Distinct

In the previous chapters we analyzed the designed protocols which exploited similarities between consecutive time steps. In the first part, filter-based protocols are analyzed using competitiveness and the goal was to identify filters to verify the output. In the second part, the number of sensor nodes which observe new values are bounded.

Since we applied both techniques to the Top-k problem we see that the concern of these techniques are orthogonal and might be potentially combined. Intuitively speaking, we understand the filter-based protocols for the Top-k problem as finding a new certificate (filter) which verifies the output (the k nodes which observed the Top-k) to be correct. The improved communication bounds stem from the fact that finding a new filter is less communication intense in comparison to calculating each output from scratch. This way of applying competitiveness yields a worst-case instance in which the output does not change within a phase such that an optimal offline algorithm does not need to communicate within this phase. On the other hand, our maintaining protocols concentrate on instances which change the output in every time step.

In this chapter we consider the Approximate Count Distinct problem and apply both a filter-based protocol which is analyzed using competitive analysis on the one hand and is a maintaining protocol on the other hand.

10.1 Introduction & Contribution

Variant	Expected communication costs
One Shot	$\mathcal{O}\left(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)$
Multiple Time Steps	$\mathcal{O}\left(T\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)$
	$\mathcal{O}\left((\sigma+\delta)\frac{\log(n)\cdot R^*}{\lvert D_t\rvert}\cdot T\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\right)$

We start the presentation of the protocol for the Approximated Count Distinct problem for multiple time steps by considering only one time step. Again, this approach can be applied at each of the T time steps (similar to the previous chapters).

In contrast to the previous chapters the multiple time step case consists of two parts in the stated bound. On the one hand we reduce the communication by applying the analysis similar to the analysis of the frequencies (cf. Chapter 8) which results in the factor of $\mathcal{O}(\sigma+\delta)$ and, on the other hand, we apply competitive analysis which introduces the factor of $\mathcal{O}\left(\frac{\log(n)\cdot R^*}{|D_t|}\right)$. Note that the factor introduced by the filter-based approach does not introduce an overhead (up to constant factors).

A Short Overview The essential idea of the approach in this chapter is very simple: First, consider the values $\{1,\ldots,\Delta\}$ which can be observed by the nodes and determine a sample S_Δ of these values (where value v is iid. within S_Δ with prob. p).

Given S_Δ we apply the Domain Monitoring strategy: That is, we aim at finding a representative which witnesses that this value is both sampled and observed.

However, the information whether v_i is within S_Δ and thus, the node that takes place in the protocols afterwards has to be determined 'efficiently'; i.e., each node has to determine the answer without sending the value to the server and waiting for an answer. For this situation we apply the public coin schema: each node can determine S_Δ consistently without the need to transmit S_Δ itself.

Chapter Basis.

- Bemmann, Biermeier, Bürmann, Kemper, Knollmann, Knorr, Kothe, Mäcker, M., Meyer auf der Heide, Riechers, Schaefer, Sundermeier. Monitoring of Domain Related Problems in Distributed Data Streams. In: *Structural Information and Communication Complexity - 24th International Colloquium (SIROCCO 2017)*, [BBB$^+$17].

10.2 Count Distinct Monitoring – One Shot

In this section we present an (ε,δ)-approximation algorithm for the Count Distinct Monitoring Problem. The basic approach is similar to the one presented for monitoring the frequency of each value. That is, we first estimate $|D_t|$ up to a (small) constant factor and then use the result to define a protocol for obtaining an (ε, δ)-approximation.

Assuming that, at any fixed time t, each value was observed by at most one node, it would be possible to solve this problem with expected communication cost of at most $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ (per time step t and per value $v \in D_t$) using the same approach as in the previous section. Since this assumption is generally not true, we aim at simulating such behavior that for each value in the domain only one random experiment is applied. We apply the concept of *public coins*, which allows nodes measuring the same value to observe identical outcomes of their random experiments. To this end, nodes have access to a shared random string R of fully independent and unbiased bits. This can be achieved by letting all nodes use the same pseudorandom number generator with a common starting seed, adding a constant number of messages to the bounds proven below. We assume that the server sends a new seed in each phase by only loosing at most a constant factor in the amount of communication used. However, we can drop this assumption by checking whether there are nodes that changed their value such that only in rounds in which there are changes new public randomness is needed. The formal description of the algorithm for a constant factor and an ε-approximation are given in Algorithm 19 and Algorithm 20, respectively.

We consider the access of the public coin to behave as follows: Initialized with a seed, a node accesses the sequence of random bits R bitwise; i.e., after reading the j'th bit, the node next accesses bit $j + 1$. Observe the crucial fact that as long as each node accesses the exact same number of bits, each node observes the exact same random bits simultaneously. Algorithm 19 essentially works as follows: In a first step, each node draws a number from a geometrical distribution using the public coin. By this, all nodes observing the same value v obtain the same height h_v. In the second step we apply the strategy as in the previous section to reduce communication if lots of nodes observe the same value: Each node i draws a number g_i from a geometrical distribution without using the public coin. Afterwards, all nodes with the largest height g_i among those with the largest height h_v broadcast their height h_v.

Note that only (at most n) nodes that observe value v with $h_v = max_{v'} h_{v'}$ may send a message in Algorithm 19. Now, all nodes observing the same value observe the same outcome of their random experiments determining h_v. Hence, by a similar

Algorithm 19 FREQUENCYCONSTANTFACTORAPPROXIMATION

(Node i, observes value $v = v_i$)

1. Draw a random number h_v as follows:
   Consider the next $\Delta \cdot \log n$ random bits $b_1, \ldots, b_{\Delta \cdot \log n}$ from R. Let h be the maximal number of bits $b_{v \cdot \log n+1}, \ldots, b_{v \cdot \log n+1+h}$ that equal 0. Define $h_v := \min\{h, \log n\}$.
2. Let g_i' be a random value drawn from a geometric distribution with success-probability $p = 1/2$ and define $g_i = \min(g_i', \log n)$ (without accessing public coins).
3. Broadcast drawn height h_v in round $r = \log^2 n - (h_v - 1) \cdot \log n - g_i$ unless a node i' has broadcasted before.

(Server)

1. Receive a broadcast message containing height h in round r.
2. Output $\hat{d}_t = 2^h$.

reasoning as in Lemma 8.2.1, one execution of the algorithm uses $\mathcal{O}(1)$ messages.

Using the algorithm given in Algorithm 19 and applying the same idea as in the previous section, we obtain an (ε, δ)-approximation as given in Algorithm 20: Each node tosses a coin with a success probability depending on the constant factor approximation (for which we have a result analogous to Corollary 8.2.2). Again, all nodes use the public coin so that all nodes observing the same value obtain the same outcome of this coin flip. Afterwards, those nodes which have observed a success apply the same strategy as in the previous section; that is, they draw a random value from a geometric distribution, and the nodes having the largest height send a broadcast.

Algorithm 20 FREQUENCYEPSILONFACTORAPPROX

(Node i)

1. Flip a coin with success probability $p = 2^{-q} = \frac{c \log 1/\delta}{\varepsilon^2 \hat{d}_t}$, $q \in \mathbb{N}$ as follows:
   Consider the next $\Delta \cdot q$ random bits $b_1, \ldots b_{\Delta \cdot q}$. The experiment is successful if and only if all random bits $b_{v \cdot q+1}, \ldots, b_{v \cdot q+q}$ equal 0. The node deactivates (and does not take part in Steps 2. and 3.) if the experiment was not successful.
2. Draw a random value h_i' from a geometric distribution and define $h_i = \min(h_i', \log n)$ (without accessing public coins).
3. Node i broadcasts its value in round $\log n - h_i$ unless a node i' with $v_i^t = v_{i'}^t$ has broadcasted before.

(Server)

1. Let S_t be the set of received values.
2. Output $\tilde{d}_t := |S_t|/p$

Using arguments analogous to Lemmas 8.2.3 and 8.2.4 and applying FREQUENCYEPSILONFACTORAPPROX for T time steps, we obtain the following theorem.

Theorem 10.2.1. *There exists an (ε, δ)-approximation algorithm for the Count Distinct Monitoring Problem for T time steps using $\mathcal{O}(T \cdot \frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ messages on expectation.*

10.3 Approximate Count Distinct Monitoring

In this section we consider the problem for multiple time steps and parameterise the analysis with respect to instances in which the domain does not change arbitrarily between consecutive time steps. Recall that for monitoring the frequency from a time step $t-1$ to the current time step t, all nodes that left and all nodes that entered toss a coin to estimate the number of changes. However, to identify that a node observes a value which was not observed in the previous time step, the domain has to be determined exactly.

We apply the following idea instead: For each value $v \in \{1, \ldots, \Delta\}$ we flip a (public) coin. We denote the set of values with a successful coin flip as the *sample*. Afterwards, the algorithm only proceeds on the values of the sample, i.e., in cases in which a node observes a value with a successful coin flip and no node observed this value in previous time steps, this value contributes to the estimate $\tilde{d}_t^+$ at time t. Regarding the (sample) of nodes that leave the set of observed values, the DOMAINMONITORING algorithm is applied to identify which (sampled) values are not observed any longer (and thus contribute to $\tilde{d}_t^-$). Analogous to Lemma 8.3.1, we have the following lemma.

Algorithm 21 CONTINUOUSFREQUENCYEPSILONAPPROX(ε, δ)

1. Compute $\delta' = 2\,\delta^2$
2. Broadcast a new seed value for the public coin.
3. Compute an (ε, δ')-approximation $\tilde{d}_1$ of $|D_1|$ using Algorithm 20. Furthermore, obtain the success probability p.
4. Repeat for each time step $t > 1$:
   (a) Each node i applies Algorithm 2 (EXISTENCEMONITORING) if the observed value v_i is in the sample set. Let $\hat{d}_t^-$ be the number of values (in sample set) which left the domain and $\hat{d}_t^+$ the number of nodes that join the sample.
   (b) Server computes $\tilde{d}_t = \tilde{d}_1 + \sum_{i=2}^{t} \hat{d}_i^+ / p - \sum_{i=2}^{t} \hat{d}_i^- / p$.
   (c) Break if $t = 1/\delta$ or $\left(\sum_{i=2}^{t} \tilde{d}_i^+ + \sum_{i=2}^{t} \tilde{d}_i^-\right) / p$ exceeds $\tilde{d}_1/2$.
5. Set $t = 1$ and go to Step 2.

Lemma 10.3.1. CONTINUOUSFREQUENCYEPSILONAPPROX *yields an (ε,δ)-apprx. of $|D_t|$ in any time step t.*

For the number of messages, we argue on the basis of the previous section. However, in addition the DOMAINMONITORING algorithm is applied. Observe that the size of the domain changes by at most $n/2$, and consider the case that this number of nodes observed the same value v. The expected cost (where the expectation is taken w.r.t. whether v is within the sample) is $\mathcal{O}(\log n \cdot R^* \cdot p) = \mathcal{O}\big(\frac{\log n \cdot R^*}{|D_t|\varepsilon^2} \log \frac{1}{\delta}\big)$. Similar to Theorem 8.3.3, we then obtain the following theorem.

Theorem 10.3.2. CONTINUOUSFREQUENCYEPSILONAPPROX *provides and* (ε, δ)*-approximation for the Count Distinct Monitoring Problem for* T *time steps using an amount of* $\Theta\left((1 + T \cdot \max\{2\sigma, \delta\}) \frac{\log(n) \cdot R^*}{|D_t| \cdot \varepsilon^2} \log \frac{1}{\delta}\right)$ *messages on expectation, assuming* $\sigma \leq 1/2$ *holds.*

Assuming the instance is not too short, i.e., T is large enough, the bounds follow as stated at the beginning of this chapter.

CHAPTER 11

FUTURE RESEARCH PERSPECTIVES

In this very last chapter we sketch some ideas which follow up on the results shown in this part of the thesis.

More Research on the Distributed Data Structure

We see further applications of the sketch in our data structure in Chapter 9. Among others, one (direct) application is to output an (axis-aligned) bounding box for the given data points. An interesting problem to consider is as follows: Each sensor node observes its position in the plane and our task is to output the (sensor nodes that form the) convex hull. The sensor nodes are mobile; i.e., they can move between two time steps by a bounded speed. Let n_h denote the number of nodes on the convex hull and observe that $\Omega(n_h)$ messages are needed to determine the output. With the algorithms in this paper the convex hull can be computed using $\mathcal{O}(n_h \cdot \log n)$ messages. We ask whether we may apply (some variant of) our sketch such that $\mathcal{O}(n_h \cdot \log \log n)$ messages are sufficient to determine the points on the convex hull.

Revisiting the analysis of our data structure we observe that we reduce the communication especially if the adversary changes only a few data items at a time. Additionally, we analyze a worst-case adversary who changes data items with a small rank, i.e., with a polylogarithmic rank. It might be of interest to consider restrictions of the adversary to prove stronger bounds: The node which observes a new data item is chosen uniformly at random, or the new data item observed is close to the old value.

Combination of Filter-Based and Dynamic Algorithms

For this aspect, we consider the Top-k-Position Monitoring problem. We elaborated on this problem in great detail; however, we did not combined the techniques into one

algorithm with one analysis. We are convinced that both (filter and dynamic algorithms) are potentially combinable and propose that it is possible to show a result of $\mathcal{O}(k+\log m+\log \Delta)$, where m denotes the number of nodes that the adversary is able to update between two time steps. Note that this result on the competitiveness is not strict; that is, there are still $\mathcal{O}(k+\log n)$ messages needed on expectation to identify the k largest positions (and their respective values). The data structure from Chapter 9 can be extended to lead to this result.

Sticking to the same problem, a combination could also be in the next perspective: In Chapter 9 we assumed that the entire Top-k is changed from one time step to the next one. However, if there is an overlap of the 'old' Top-k and the 'new' one we can use the old set to filter out unnecessary communication: intuitively speaking, if there is at least one node which stays in the Top-k set than with $\mathcal{O}(\log k)$ messages in expectation to identify the largest and smallest values within the old Top-k.

We leave the Top-k-Position example and turn to a general perspective. Combinations of the two techniques might lead to interesting approaches and results: especially for problems which need large filters, e.g. the SUM problem, in which every node needs a specific filter. For such problems a sampling technique and a filter technique only with respect to the sample can be applied in combination. This might lead to significantly smaller bounds which are comparable to the results in Chapter 10.

Structure on the Input

In this thesis we did not assumed any structure on the input. The overall goal was to generate insights in the setting and have results which are independent of specific applications. The assumption that an adversary changes the k largest values (to name the standard example) might be too pessimistic. There is a large potential of other instance generation processes which still lead to valuable insights without performing just simulations alone.

One possibility is to model the adversary as a completely random process which chooses m sensor nodes uniformly at random to change its value. In fact, we expect significant better bounds in comparison to the worst-case analysis. However, it is unclear how large the effect on the bounds are. Furthermore, it seems unlikely that in a sensor network the sensors change their values in such a 'smooth' manner. Consider the case in which a fire starts to burn: The sensors which start to sense larger and larger values might come up with larger values in the next time step. These considerations yield an analysis framework in which we mix worst-case analysis with a randomized proportion: The adversary defines the sensor nodes to observe an update and by a ran-

dom process these positions become perturbed. We expect to see a transition between the bounds for the worst-case and the pure random process.

Bibliography

[ABC09] Chrisil Arackaparambil, Joshua Brody, and Amit Chakrabarti. Functional monitoring without monotonicity. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming*, pages 95–106, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Agg07] Charu Aggarwal. *Data streams: models and algorithms*, volume 31. Springer, 2007.

[BBB+17] Pascal Bemmann, Felix Biermeier, Jan Bürmann, Arne Kemper, Till Knollmann, Steffen Knorr, Nils Kothe, Alexander Mäcker, Manuel Malatyali, Friedhelm Meyer auf der Heide, Sören Riechers, Johannes Schaefer, and Jannik Sundermeier. Monitoring of domain-related problems in distributed data streams. In *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017, Porquerolles, France, June 19-22, 2017, Revised Selected Papers*, pages 212–226, 2017.

[BFMM17] Felix Biermeier, Björn Feldkord, Manuel Malatyali, and Friedhelm Meyer auf der Heide. A communication-efficient distributed data structure for top-k and k-select queries. In *Approximation and Online Algorithms - 15th International Workshop, WAOA 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers*, pages 285–300, 2017.

[BO03] Brian Babcock and Chris Olston. Distributed top-k monitoring. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 28–39. ACM, 2003.

[CMY08] Graham Cormode, Muthu Muthukrishnan, and Ke Yi. Algorithms for distributed functional monitoring. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '08)*, pages 1076–1085. SIAM, 2008.

[CMY11] Graham Cormode, Muthu Muthukrishnan, and Ke Yi. Algorithms for Distributed Functional Monitoring. *ACM Transactions on Algorithms*, 7(2):21:1–21:20, 2011.

[Cor13] Graham Cormode. The continuous distributed monitoring model. *ACM SIGMOD Record*, 42(1):5–14, 2013.

[DEI06] Sashka Davis, Jeff Edmonds, and Russell Impagliazzo. Online algorithms to minimize resource reallocations and network communication. In *Proceedings of the 9th international conference on Approximation Algorithms for Combinatorial Optimization Problems, and 10th international conference on Randomization and Computation*, pages 104–115. Springer-Verlag, 2006.

[DMadHR+03] Valentina Damerow, Friedhelm Meyer auf der Heide, Harald Räcke, Christian Scheideler, and Christian Sohler. Smoothed motion complexity. In *European Symposium on Algorithms*, pages 161–171. Springer, 2003.

[DMMadH+12] Valentina Damerow, Bodo Manthey, Friedhelm Meyer auf der Heide, Harald Räcke, Christian Scheideler, Christian Sohler, and Till Tantau. Smoothed analysis of left-to-right maxima with applications. *ACM Transactions on Algorithms (TALG)*, 8(3):30, 2012.

[FFG+18] Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-dynamic bin packing with little repacking. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 107. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[FMM18] Björn Feldkord, Manuel Malatyali, and Friedhelm Meyer auf der Heide. A dynamic distributed data structure for top-k and k-select

queries. In *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*, pages 311–329, 2018.

[FMS93] Esteban Feuerstein and Alberto Marchetti-Spaccamela. Dynamic algorithms for shortest paths in planar graphs. *Theoretical Computer Science*, 116(2):359–371, 1993.

[GK12] Yiannis Giannakopoulos and Elias Koutsoupias. Competitive analysis of maintaining frequent items of a stream. In *Proceedings of the 13th Scandinavian Conference on Algorithm Theory*, SWAT'12, pages 340–351, Berlin, Heidelberg, 2012. Springer-Verlag.

[GT01] Phillip Gibbons and Srikanta Tirthapura. Estimating simple functions on the union of data streams. In *Proceedings of the 13th annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '01)*, pages 281–291. ACM, 2001.

[HYZ12] Zengfeng Huang, Ke Yi, and Qin Zhang. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '12)*, pages 295–306. ACM, 2012.

[LLT10] Tak Wah Lam, Chi-Man Liu, and Hing-Fung Ting. Online Tracking of the Dominance Relationship of Distributed Multi-dimensional Data. In *Proceedings of the 8th International Workshop on Approximation and Online Algorithms (WAOA '10)*, volume 6534 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2010.

[LPPSP03] Zvi Lotker, Elan Pavlov, Boaz Patt-Shamir, and David Peleg. Mst construction in o (log log n) communication rounds. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 94–100. ACM, 2003.

[MBP06] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 635–646. ACM, 2006.

[MG85] John M Marberg and Eli Gafni. *An optimal shout-echo algorithm for selection in distributed sets*. University of California (Los Angeles). Computer Science Department, 1985.

[MMM15] Alexander Mäcker, Manuel Malatyali, and Friedhelm Meyer auf der Heide. Online Top-k-Position Monitoring of Distributed Data Streams. In *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS '15)*, pages 357–364. IEEE, 2015.

[MMM16] Alexander Mäcker, Manuel Malatyali, and Friedhelm Meyer auf der Heide. On Competitive Algorithms for Approximations of Top-k-Position Monitoring of Distributed Streams. In *Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS '16)*, pages 700–709. IEEE, 2016.

[MP80] James Ian Munro and Mike Stewart Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.

[Mut05] Muthu Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236, August 2005.

[RSS86] Doron Rotem, Nicola Santoro, and Jeffrey Sidney. Shout echo selection in distributed files. *Networks*, 16(1):77–86, 1986.

[SF13] Robert Sedgewick and Philippe Flajolet. *An introduction to the analysis of algorithms*. Pearson Education India, 2013.

[WZ12] David Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the 44th Symposium on Theory of Computing (STOC '12)*, pages 941–960. ACM, 2012.

[YZ12] Ke Yi and Qin Zhang. Multidimensional online tracking. *ACM Transactions on Algorithms*, 8(2):12, 2012.

[ZCPT09] Zhenjie Zhang, Reynold Cheng, Dimitris Papadias, and Anthony KH Tung. Minimizing the communication cost for continuous skyline maintenance. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 495–508. ACM, 2009.