Monitoring Churn in Wireless Networks

Stephan Holzer[1] Yvonne-Anne Pignolet[2] Jasmin Smula[1] Roger Wattenhofer[1]
{stholzer, smulaj, wattenhofer}@tik.ee.ethz.ch, yvonne-anne.pignolet@ch.abb.com
[1]Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland
[2] ABB Corporate Research, Dättwil, Switzerland (part of work was done while at IBM Research, Zurich Research Laboratory, Switzerland)

Abstract

Wireless networks often experience a significant amount of churn, i.e. the arrival and departure of nodes, and often it is necessary to keep all nodes informed about all other nodes in the network. In this paper we propose a distributed algorithm for single-hop networks that detects churn, meaning that the nodes observe other nodes joining or leaving the network and inform all other nodes in the network about their observations. Our algorithm works correctly even if the nodes which join or leave and the respective points in time are chosen by an adversary in a worst-case fashion. The delay until notification is small, such that all nodes of the network are informed about changes quickly, in asymptotically optimal time. We establish a trade-off between saving energy and minimizing the delay until notification for single- and multi-channel networks.

Keywords: wireless networks, no collision detection, sensor networks, group membership, monitoring.

1. Introduction

In traditional (wired) distributed systems, the *group membership problem* has been studied thoroughly (we refer to [8] for a survey). The basic premise of group membership is to know which other nodes are there, for instance to share the load of some task. Nowadays large parts of wired networks are replaced by Bluetooth or wireless LAN since one does not have to build an expensive communication infrastructure first, but can communicate in "ad hoc" mode immediately. This motivates a revisit of the *group membership problem* in a wireless context: imagine for example a bunch of wireless sensors, distributed in an area to observe that area. From time to time some of the nodes fail, maybe because they run out of energy, maybe because they are maliciously destroyed. On the other hand, from time to time some more sensors are added. Despite this *churn* (nodes joining and leaving [23]), all nodes should be aware of all present nodes, with small delay only. To account for the self-organizing flavor and the wireless context we decided to change the name from group membership

to *self-monitoring* in this paper. We present an efficient algorithm for the self-monitoring problem in an adversarial setting.

Reducing the frequency of checking for changes, and thus the number of messages exchanged per time period, prolongs the time interval until every node is informed about changes. Since energy as well as communication channels are scarce resources for wireless devices, we evaluate a trade-off between energy and delay until notification for single- and multi-channel networks. For single-channel networks, our algorithm can be applied to multi-hop networks using [2], which shows that algorithms designed for single-hop networks can be efficiently emulated on multi-hop networks.

This article is structured as follows. In Section 2 we review work under the same communication model and related problems, followed by a description of the model and a formal definition of the problem we analyze in Section 3. The monitoring algorithm and a proof of its time complexity are provided in Section 4. In this section, we assume that the number of channels available is large. Subsequently we prove lower bounds and algorithmic modifications necessary for a bounded number of channels and the consequences for the time complexity and energy consumption in Section 5.

2. Related Work

Many algorithms have been designed for wireless networks under varying assumptions concerning the communication model (reception range, collision detection, transmission failures, etc.). There are many problems that are non-trivial even in single-hop networks. We focus here on networks where nodes cannot distinguish collisions from noise (no-collision detection model). The ability to detect collisions can lead to an exponential speed up, e.g., as shown in [17] for leader election. Moreover we consider the energy expenditure for transmission and listening. Basic algorithms for these networks can be used as services or building blocks for more complex algorithms and applications. Among them are initialization (n nodes without IDs are assigned labels $1, \ldots, n$) [19], leader election [18, 20], size approximation [4, 14], alerting (all nodes are notified if an event happens at one or more nodes) [16], sorting (n values distributed among n nodes, the i^{th} value is moved to the i^{th} node) [15, 25], selection problems like finding the minimum, maximum, median value [24] and computing the average value [18], and do-all (schedule t similar tasks among n nodes with at most f failures) [6] and information exchange despite adversarial interference (n nodes inform each other about $n - t$ values, an adversary can disturb communication on t channels by jamming) [12, 11]. Note that in contrast to our work the adversary examined in these papers cannot let nodes join or crash. Moreover we cannot apply existing size approximation algorithms to estimate the number of newly joined nodes, since they do not handle node failures and they do not give high probability results for a small number of joining nodes. The time and energy complexity of these services are summarized in Table 1.

Some of the algorithms require knowledge of the number of nodes n or an approximation of n, some are based on the assumption that nodes have unique IDs

Service	Adversary	Time	Energy per node	Source
Initialization	no	$O(n)$	$O(\log\log n)$	[21]
	$O(\log n)$ energy	$O(n)$	$O(\sqrt{\log n})$	[19]
Leader Election	no	$O(\log n)$[1]	$O(\log\log n)$	[20]
	no	$O(\log^2 n)$[2]	$O(\log\log n)$	[13]
	$O(\log n)$ energy	$O(\log^3 n)$	$O(\sqrt{\log n})$	[18]
Size Approx.	no	$O(\log^{2+\epsilon} n)$	$O(\log^{\epsilon}\log n)$	[13]
	no	$O(\log^2 n)$[3]	$O(\log n)$	[4]
	$O(\log n)$ energy	$O(\log^{2.5} n \log\log n)$	$O(\log n \log\log n)$	[14]
Wake up	no	$O(\log^2 n)$	–	[26]
ϵ-Mutual Exclusion	no	$O(\log(n)\log(1/\epsilon)$	-	[3]
Alerting	no	$O(\log^3 n)$	$O(\log n/\log\log n)$	[16]
Sorting	no	$O(n\log n)$	$O(\log^2 n)$	[15]
	no	$O(n\log n\log^* n)$	$O(\log n\log^* n)$	[15]
Selection	no	$O(n)$	$O(1)$	[24]
Average	ϵn channels	$O(n\log n)$[4]	$O(n\log n)$	[18]
Info. Exchange	t channels	$O(n/t^2 + t^5\log^2 n)$	-	[12]
	t channels	$O(nt^3\log n)$	-	[11]

Table 1: Services for wireless single-hop networks consisting of n nodes without the ability to detect collisions. None of these services is designed to tolerate churn. The second row describes the resilience of the service to jamming. An adversary jamming t channels can prevent communication on t channels. Nodes can never receive a message transmitted on such a channel. An adversary with bounded energy can disturb channels until its energy has been used up (one energy unit per channel and time slot). [1]In expectation. [2]With high probability. [3]This algorithm uses only bit messages. [4]If the maximum difference between two values is $O(n)$. The analyses of the algorithms for information exchange and wake up do not include the energy consumption.

and the number of communication channels available as well as the maximum message size varies (typically $O(\log n)$ bits per message). The adversaries the algorithms tolerate differ as well. Negative results for such networks include the impossibility of acknowledged radio broadcasting [5] (transmitting a message from one special node called the source to all other nodes and informing the source about its completion) and the lower bound of $\Omega(n\log n)$ time complexity for deterministic leader election [10, 17]. Our work can be seen as continuous initialization with the extension that more information is available. New nodes can join the network later and are given a label (position in the ID table). After each round of our self-monitoring algorithm, these labels are updated and in addition all nodes know which nodes have failed. Moreover, the ID table can be used to designate a leader and all nodes are aware of the current network size. In [1], a routing problem is studied in a multi-channel, single-hop, time slotted scenario and energy is considered as well. However, the algorithm they propose is not suitable for our application, since it requires a preprocessing phase of $O(n)$ time slots.

One of the problems underlying the monitoring problem is the dynamic broadcast problem, where an adversary can continuously inject packets to be delivered to all participants of the network, see [7] for (im)possibility results and algorithms (nodes are assumed not to crash in this model).

The problem we solve can be viewed as a special case of the continuous

gossip problem, introduced in [9] recently: an adversary can inject rumors as well as crash and restart participating nodes at any time, yet the rumors need to reach their destination before a deadline. The authors analyze the problem in a message passing model with unbounded message size and no collisions and devise an algorithm with a guaranteed per-round message complexity. Our update items can be viewed as rumors that directly depend on the crashes and restarts and the deadlines are related to the number of crashes and restarts in a time interval.

3. Model

The network consists of a set of wireless nodes, each with a built-in unique ID. All nodes are within communication range of each other, i.e., every node can communicate with every other node directly (single-hop). New nodes may join the network at any time, and nodes can leave or crash without notice. To simplify the presentation of the algorithms and their analysis, we assume time to be divided into synchronized time slots. Messages are of bounded size, each message can only contain the equivalent of a constant number of IDs. Messages between nodes are sent over so-called communication channels, or just channels. If only one channel is available, we talk about single-channel networks, otherwise multi-channel networks. We first assume that the number of properly divided communication channels is rather large, a requirement we drop later. The fluctuation of nodes in the network is called churn [23]. The severity of churn is determined by the number of nodes joining/leaving per time interval. We exclude Byzantine behavior and assume that as soon as a node crashes, it does not send any messages anymore. Due to the churn, the number of nodes in the network varies over time.

In each time slot a node v is in one of three operating states: `transmit` (v broadcasts on some channel k), `receive` (v monitors some channel k) or `sleep` (v does not send or receive anything). In the states `transmit` and `receive`, v can choose an arbitrary channel k from all available communication channels. A transmission is successful, if exactly one node is transmitting on channel k at a time, and all nodes monitoring this channel receive the message sent. If more than one node transmits on channel k at the same time, listening nodes can neither receive any message due to interference (called a *collision*) nor do they recognize any communication on the channel (this is known as *no collision detection*). The energy dissipation of v is defined to be the sum of the energy for transmission and reception (cf. for example [13]). Because in current embedded systems transmitting and receiving consumes several orders of magnitude more energy than sleeping or local computations, we set the energy consumption for being in state `transmit` or `receive` to unity and neglect the energy used in state `sleep` or for local computations. The nodes have sufficient memory to store an *ID table* containing all IDs of currently participating nodes and execute the provided algorithms. By n_t we denote the number of entries in the correct ID table at time t. Since nodes which just joined do not immediately know the

whole ID table (due to the limited message size), we define an ID table with entries for all n_t positions to be *good*. Otherwise it is called *fragmentary*.

At any time, an adversary may select arbitrary nodes to crash, or it may let new nodes join the network. In this case, all nodes should be aware of this change and update their ID tables accordingly as soon as possible. However, the adversary may not modify or destroy messages. Since messages are of bounded size, nodes can learn at most a constant number of identifiers per message. As each node can receive at most one message per time slot, with any algorithm it needs at least $c_{\min}$ time units on average (for some constant $c_{\min}$) until the nodes know about *one* crash or join that just occurred. In other words, if *on average* more than rate $r_{\max} := c_{\min}^{-1}$ nodes crash (or join) per time unit, no algorithm can handle the information (cf. [7] for the maximum tolerable average message rate in a dynamic broadcast setting). In the following, we define an adversary and monitoring algorithm accordingly. Crashes or joins that occur at the same point in time are called a burst. Denote by b the number of crashes / joins that happen in a maximal burst and by $\tilde{b}$ the maximal burst size that an algorithm tolerates.

Definition 3.1 (c-Adversary, $(c, \tilde{b})$-Adversary)**.** *We call an adversary a* c-adversary *if it lets nodes join and crash arbitrarily as long as: 1. There remains at least one node having a good ID table in the network at any time. 2. On average the number of adversarial joins / crashes is at most one node in c time slots. The adversary has full knowledge of the algorithm and can coordinate crash and join events with the aim of making the algorithm fail. A fixed burst* $(c, \tilde{b})$-adversary *is a c-adversary who lets at most $\tilde{b}$ nodes join or leave the network during every period of $c \cdot \tilde{b}$ time slots.*

4. Monitoring Algorithm

In this section, we present the Monitoring Algorithm that solves the self-monitoring problem and prove that it takes only a short time after a burst happened until all nodes have updated their ID tables correctly. First we briefly describe the different steps of an algorithm that works correctly if the maximal size of a burst is upper bounded by a known value $\tilde{b}$ and explain how to use this algorithm to obtain our Monitoring Algorithm that works for unknown burst size b. After that we consider the different steps in greater detail.

The algorithm we propose is asymptotically optimal in the sense that it can survive in a setting where on average one crash or join occurs in c time units, for a constant c. We can tolerate bursty churn (a large number of nodes joining or leaving during a small time interval). Similarly to an optimal algorithm, we need time to recover from bursts since the number of newly joining (or crashed) nodes is bounded according to the message size. The algorithm can also tolerate churn while trying to recover from previous bursts; again the only limit is that nodes can only learn about $r := c^{-1}$ crashed or joined nodes per time unit on average. Indeed, the adversary may crash all but one node at the same instant (killing all nodes is a special case, leading to an initialization problem,

which we do not address here). Clearly, learning about nodes that have left or joined takes time, depending on the size of the bursts. If there is a burst of β joins or crashes, an optimal algorithm needs at least $\beta \cdot c_{\min}$ time until the corresponding information at all nodes is up-to-date, for some constant $c_{\min}$. Our algorithm needs time $\beta \cdot c$ for some constant $c > c_{\min}$. If bursts happen while recovering from previous bursts, delays occur because more information has to be distributed and the amount of information per message is limited. In our algorithm the delay until all nodes are informed about all changes is asymptotically optimal: the algorithm handles the maximum average rate of churn any algorithm can tolerate in this communication model.

Our algorithm is partially randomized. However, randomness is only required for detecting new nodes since this part cannot be done in a deterministic fashion. (This is because the number of joiners is not bounded at all. If nodes want to join the network, they have to transmit their ID at some point, and because the number of such potential joiners is unknown, they have to transmit somehow randomly to break ties.) All other parts of the algorithm are deterministic, which might be of interest in a setting where only updates on crashed nodes are needed and no nodes join the network.

Main Theorem 4.1. *There exists a* monitoring algorithm *that tolerates c-adversaries with maximum burst size b with logarithmic additive overhead: $O(b + \log n_t)$ time slots after an event all nodes have updated the corresponding entries in their ID tables. The number of channels needed for this algorithm is $O(n_t / \log n_t)$*

Remark 4.2. *Nodes with fragmentary ID tables update their ID table as well and learn the IDs of all nodes in the network in parallel to executing the monitoring algorithm (see Section 4.6). Eventually they have good ID tables (see Section 4.7). This takes at most $\Theta(n_t)$ time slots, as we explain later (cf.* INVARIANT 4.3*). Section 5 gives lower bounds and describes how to adapt the algorithm for fewer channels.*

At first, we describe a fixed burst monitoring algorithm $A_{\tilde{b}}$ which works correctly if the burst size is in the order of $\tilde{b}$, ALGORITHM 1. Then, from a family of fixed burst monitoring algorithms FBMA $\{A_{\tilde{b}}\}_{\tilde{b}\in\mathbb{N}}$ that tolerate $(c/4, \tilde{b})$-adversaries we construct a monitoring algorithm B. Each $A_{\tilde{b}}$ might fail if the churn is too large – since we do not know b beforehand, we derive an algorithm B that adapts to the bursts by searching for a good value for $\tilde{b}$ with a binary doubling search procedure.

From now on, we use the term "FBMA" as an abbreviation for "fixed burst monitoring algorithm". Let us now consider the FBMA $A_{\tilde{b}}$ for a fixed $\tilde{b} \in \mathbb{N}$, ALGORITHM 1. In order to work correctly if the bursts are smaller than anticipated and to detect its failure it requires the following invariant.

Invariant 4.3. *The ID tables of all nodes that have been in the network for $\Theta(n_t)$ time slots always contain the same entries. Nodes that joined more recently know their position (according to the ascending order of the IDs) in the* ID table.

To ensure that this invariant holds when starting the algorithm, we may assume that at time 0 there is only a single designated node active, and all other nodes still need to join. This leads to the same *ID table* at all nodes.

Theorem 4.4. *If* INVARIANT 4.3 *holds at the start, then for all* $\tilde{b} \in \mathbb{N}$, FBMA $A_{\tilde{b}}$ *(*ALGORITHM 1*) tolerates* $(c, \tilde{b})$*-adversaries for a constant* c*. Furthermore each node detects if the algorithm failed* $c \cdot (\tilde{b} + \log n_t)$ *time slots after a stronger adversary caused a burst larger than* $2\tilde{b}$*. The energy consumption and the time for detection is asymptotically optimal.*

Proof. In brief, ALGORITHM 1 repeats a loop consisting of six steps to maintain up-to-date information in the *ID tables* of the nodes. Each step is fully distributed and does not need a central entity to control its execution. Subsequently, we call one execution of the loop of the FBMA $A_{\tilde{b}}$ a *round*. We now briefly describe each step of the algorithm and indicate its time complexity. Detailed descriptions of the steps can be found in subsequent sections, and the same applies to explanations of the time complexities.

Algorithm 1 $A_{\tilde{b}}$ for a fixed $\tilde{b} \in \mathbb{N}$

loop forever // same *ID table* (INV. 4.3) at all nodes
1: partition nodes into sets of size $O(\tilde{b})$;
2: detect crashed nodes in each set on separate channels in parallel;
3: detect joined nodes;
4: disseminate information on crashed and joined nodes to all nodes;
5: stop if burst too large;
6: all nodes update their *ID table*;

Step 1 – partition nodes into sets: Nodes are divided into $N \in O(1 + n_t/\tilde{b})$ sets $V := \{S_1, \ldots, S_N\}$. Based on the information in their *ID table*, the nodes can determine which set they belong to by following a deterministic procedure. Each set appoints nodes as representatives of the set and designates their replacements in case they crash (details in SECTION 4.1). No communication, time complexity $O(1)$.

Step 2 – detect crashed nodes in each set on separate channels: Each set $S_I \in V$ executes an algorithm to detect its crashed nodes. No communication between sets takes place. To avoid collisions each set carries out its intra-set communication on a separate channel. To find out if any of the set members in S_I have crashed, each node sends a "hello" message in a designated time slot. All other nodes of the set detect who did not send a message and generate the information to disseminate: a list of so-called update items U_I (details in SECTION 4.2). N channels necessary, time complexity $O(\tilde{b})$.

Step 3 – detect joined nodes: New nodes listen to learn the tolerated burst size $\tilde{b}$ and when to try joining. They send requests to join to S_1 with probability $1/\tilde{b}$. In expectation at least one node can join in a constant number of rounds if the estimate $\tilde{b}$ is in $\Theta(b)$. Detected joiners are added to U_1 together with a note that they joined. After $O(\tilde{b} + \log n_t)$ time slots S_1 decides whether

the estimate $\tilde{b}$ needs to be doubled due to too many joiners. Its decision is correct with high probability (whp), that is with probability greater than $1 - n_t^{-\gamma}$ for any but fixed constant γ (details in SECTION 4.3). One channel necessary, time complexity $O(\tilde{b} + \log n_t)$.

Step 4 – disseminate information on crashed and joined nodes to all nodes: Now every set S_I has a list U_I of update items containing the IDs of crashed and joined nodes in the set. To distribute this information, each set becomes a vertex of a balanced binary tree and the representative nodes communicate with the representatives of neighboring vertices in the tree according to a pre-computed schedule. If a representative crashes, there are $\tilde{b}$ replacements to take over its job. No collisions occur due to the schedule (details in SECTION 4.4). N channels necessary, time complexity $O(\tilde{b} + \log n_t)$.

Step 5 – stop if burst too large: If the adversary is too strong, information on some of the sets is missing, or more than $\tilde{b}$ nodes crashed or tried to join. In this case, all nodes are notified and the execution of the algorithm stops (details in SECTION 4.5). N channels necessary, time complexity $O(\tilde{b} + \log n_t)$.

Step 6 – all nodes update their *ID table*: If the algorithm did not stop, every node now has the same list $U = \bigcup_{I=1}^{N} U_I$ and can update its *ID table*. INVARIANT 4.3 holds. No communication, time complexity $O(1)$.

Remark 4.5. *Newly arrived nodes do not know the* ID table *yet and have to learn the IDs of all present nodes in asymptotically optimal time, described in* SECTION 4.5. *However, even with fragmentary* ID tables *they can participate in the algorithm, see* SECTION 4.6.

While steps 1 and 6 are executed locally and hence the time complexity is constant, steps 2–5 require communication between nodes. The following sections describe the steps in more detail and examine their time complexity as well as prove that the INVARIANT 4.3 at the beginning of the loop holds (as long as b is bounded by $\tilde{b}$ – otherwise the algorithm detects that it failed). The number of channels used is N. If $n_t < 4\tilde{b} + 4$, only one set is constructed in Step 2, and the dissemination step 4 as well as the step 5 to detect oversized bursts can be simplified. Thanks to the definition of the adversary there is always at least one node alive which has a good ID table. As a consequence a proof for the case $n_t \geq 4\tilde{b} + 4$ implies the other case, thus we focus on it in the remainder of this paper. □

A lower bound and the optimality of the algorithm's energy consumption is proved in Section 5.

Proof of Main Theorem 4.1. We construct a monitoring algorithm B from a family of fixed burst monitoring algorithms FBMA $A_{\tilde{b}}$ (ALGORITHM 1). It executes algorithms $A_{\tilde{b}_i}$ from the above family with estimated values $\tilde{b}_i$ for b, starting with $\tilde{b}_1 := \log n_t$ (we do not start with $\tilde{b}_1 = 1$ because the running time of A always exceeds $\log n_t$ due to the dissemination step). If algorithm A detects its failure, we know that an algorithm A tolerating $(c/4, \tilde{b}_i)$-adversaries is not sufficient and B doubles the estimated value of b to $\tilde{b}_{i+1} := 2\tilde{b}_i = 2^i \log n_t$. Let

the adversary's maximal burst be b. After at most $\log(b/\log n_t)+1$ repetitions, the algorithm A succeeds and so does B. The total time needed by B is at most

$$\sum_{i=1}^{\log\left(\frac{b}{\log n_t}\right)+1} \frac{c}{4}\cdot(\tilde{b}_i+\log n_t) \;<\; \frac{c}{4}\sum_{i=0}^{\log\left(\frac{b}{\log n_t}\right)} 2^{i+1}\log n_t \le \frac{c}{4}\cdot 2^2\cdot\frac{b}{\log n_t}\cdot\log n_t = c\cdot b.$$

The number of channels necessary is maximal for b_1, i.e., at most $O(n_t/b_1) = O(n_t/\log n_t)$ channels are used. □

Remark 4.6 (Adaptability)**.** *After a maximal burst of size b happened, the above procedure always needs as much time as it needed for the big burst for all later bursts. The algorithm can be modified to update a network the quicker the smaller the current burst is by setting the estimate $\tilde{b}$ to $\tilde{b}_1$ after every successful update (proofs need to be adjusted slightly in a few spots).*

4.1. Partition Nodes into Sets (Step 1)

Compute sets: If $\tilde{b} \ge n_t/4 - 4$, the network forms one large set. If $\tilde{b} < n_t/4 - 4$, let $s := 2\tilde{b} + 2$ and partition the n_t nodes into $N := \lceil \frac{n_t}{s} \rceil - 1$ sets $S_1, \ldots, S_N$. Each set is of size s, except S_N which contains between s and $2s$ nodes. The nodes are assigned to the sets in a canonical way, based on their ID's position in the sorted *ID table* $\{id_1 < id_2 < \cdots < id_{n_t}\}$. Set S_I is the set $S_I := \{id_{(I-1)\cdot s+1}, \ldots, id_{I\cdot s}\}$ for $1 \le I \le N-1$ and $S_N = \{id_{(N-1)\cdot s+1}, \ldots, id_{N\cdot s}, \ldots, id_{n_t}\}$. We denote the index of S_I by a capital I and call it the ID of the set. Let us denote the set of all sets $\{S_1, \ldots, S_N\}$ by V (since the sets are the vertices of a communication graph in the dissemination step 4). Note that there is no ambiguity in the mapping of nodes to sets and thus we sometimes write S_{I_v} to refer to the set to which node v belongs to.

Compute representatives: In the subsequent steps, the sets communicate with each other. To this end, representative senders and receivers are chosen to act on behalf of the set. Moreover, for each representative, the set appoints $\tilde{b}$ replacement nodes to monitor the representative and take over if it crashes. Each set S_I designates two sets of nodes consisting of $\tilde{b}+1$ nodes: $R^{sender} := \{id_{(I-1)s}, \ldots, id_{(I-1)s+\tilde{b}}\}$ and $R^{receiver} := \{id_{(I-1)s+\tilde{b}+1}, \ldots, id_{(I-1)s+|S_I|-1}\}$. In each set we appoint the node with smallest ID to be the representative sender / receiver of S_I, denoted by r_I^{sender}, $r_I^{receiver}$. Its replacements are the other $\tilde{b}$ nodes in $R^{receiver}$ and R^{sender}. The i^{th} replacement node of a representative (which is the node with i^{th}-smallest ID of the corresponding set) takes over the role of the representative in case the representative as well as the replacement nodes 1 to $i-1$ crashed. Each node v can compute the index i_v of its ID in the ID-table. From i_v the node v can compute the set S_{I_v} to which it belongs. Then v can check easily if it is its set's representative sender/receiver or the i^{th} replacement by looking at its position in the sorted ID table. The replacement nodes listen in all time slots whether their representative is sending or receiving messages in order to detect its failure and have the same knowledge as the representative. Thus they are able to take over the representative's role

Algorithm 2 Crash Detection

```
1: compute index I_v of v's set S_{I_v} based on id_v;
2: U_{I_v} := ∅
3: for k := 0, ..., |S_{I_v}| − 1 do
4:     if i_v = I_v · |S_{I_v}| + k then
5:         send "Im here!" on channel I_v;
6:     else if no message received on channel I_v then
7:         U_{I_v} := U_{I_v} ∪ {id_{I_v · |S_{I_v}| + k}};
```

immediately. To keep things simple we often write that "S_I sends an update item to S_J" instead of "the representative sender r_I^{sender} of S_I sends information on some crashed or new node to the representative receiver $r_J^{receiver}$ of S_J". In some cases the introduced notation of representatives is used to clarify what exactly the algorithm does.

Remark 4.7. *As no communication is necessary, the time complexity of Step 1 is* $O(1)$ *and no channels are needed.*

4.2. Detect Crashed Nodes in each Set in Parallel (Step 2)

Let the time slot in which the current round of the algorithm starts be t_0. All nodes that crash in time slot t_0+1 or later might not necessarily be detected during this execution of the loop but in the next one, i.e. at most $O(\tilde{b}+\log n_t)$ time slots later. Each set S_I detects separately, which of its members crashed. Observe that if the estimate $\tilde{b}$ of b is correct, it is not possible that all nodes in a set S_I crash during the execution of a round. This is because there are at least $2\tilde{b}+2$ nodes in a set S_I and the burst size is limited to $b \leq 2\tilde{b}$. In the case that $\tilde{b}$ is smaller than b a whole set S_i could crash, but this is detected and $\tilde{b}$ is increased as we show later. Set S_I uses the channel I for communication among its set members to avoid collisions with other sets. Each node v is assigned a unique time slot to inform the other set members that it is still alive (ALGORITHM 2, lines 4–5). In all other time slots, v listens to the other set members to determine crashed nodes, i.e., when v does not receive a message in the time slot corresponding to a certain ID (line 6) it assumes that the node with this ID has crashed and adds it to U_I (line 7).

Theorem 4.8. *When repeating* ALGORITHM 2 *continuously, any crashed node in the network is detected at most two rounds (*$O(\tilde{b})$ *time slots) after it crashed unless the algorithm fails (this is the case that the estimate* $\tilde{b}$ *is too small).*

Proof. There are $O(\tilde{b})$ nodes in each set, thus each set can complete the crash detection in $O(\tilde{b})$ time slots. If there are N channels available, all sets can execute this algorithm simultaneously. If a node crashes after sending its message, its failure is detected the next time ALGORITHM 2 is executed. □

If the burst is too large it can happen that all nodes of a set S_I crash. The algorithm detects this case in Step 5 (Stop if Burst too Large) and as the total time complexity of the algorithm is $O(b+\log n_t)$ the following statement holds.

Corollary 4.9. *Using* ALGORITHM 1*, it takes time* $O(b + \log n_t)$ *until a crash is detected. Furthermore* N *channels are used (one by each set* S_I*).*

4.3. Detect Joined Nodes (Step 3)

Apart from detecting nodes that have disappeared, the network needs to be able to integrate new nodes: Let $j \leq \tilde{b}$ be the number of such joining nodes. These nodes listen on channel 1 for a message containing the current number of nodes n_t and the estimated $\tilde{b}$. This message is sent by the representative of set S_1. When a joiner has received such a message, it waits for a time slot and then tries to join by sending a message with its ID with probability $p := 1/\tilde{b}$ on channel 1. If there has not been a collision, the representative sender of the set S_1 replies to the successful joiner with a welcome message. Otherwise each unsuccessful joiner repeats sending messages with this probability followed by listening for a reply or a stop message in the next time slot. The representative sender transmits a stop message after $d' \cdot \max(\log n_t, \tilde{b})$ time slots for some constant d' depending on the error probability one can tolerate (see proof of Lemma 4.10). The probability that a joiner is successful is constant if $j < \tilde{b}$ and hence the joiners attach to the network in a constant number of rounds in expectation. ALGORITHM 3 the behavior of nodes eager to join the network in pseudo-code.

Algorithm 3 Join Algorithm

For new nodes that want to join

```
1: while attached = false do
2:   repeat
3:     listen on channel 1;
4:   until received message "b̃ bursts, n_t nodes"
5:   p := 1/b̃;
6:   loop
7:     send message "hello, id" on channel 1;
8:     listen on same channel;
9:     if received welcome message then
10:      attached := true;
11:    else if received "stop joining" then
12:      break;
```

Lemma 4.10. *In expectation a node attaches to the network in less than 3.3 rounds if* $j < \tilde{b}$*.*

Proof. Since $j < \tilde{b}$ the probability that a joiner is successful in a certain time slot is at least $1/\tilde{b}(1 - 1/\tilde{b})^{j-1} \geq 1/e\tilde{b}$. Thus the probability that a joiner is the only sender at least once during $\tilde{b}$ time slots is greater than $1 - (1 - 1/e\tilde{b})^{\tilde{b}} > 1 - e^{-e^{-1}} > 0.3$. Hence the expected number of rounds until a node has joined is less than 3.3. □

Algorithm 4 Join Detection

For nodes in set S_1 in the network

1: $count := 0$;
2: **for** $k := 0, \ldots, 4d' \cdot \max(\log n_t, \tilde{b})$ **do**
3: **if** ($i_v = r_1^{sender}$ and $k \mod 4 = 0$ and $k < d' \log n_t$) **then**
4: send message "$\tilde{b}$ bursts, n_t nodes" on channel 1;
5: **else if** received message from r_1^{sender} **then**
6: $count := count + 1$;
7: **else if** received message from joiner id_j **then**
8: **if** $i_v = r_1^{sender}$ **then**
9: send message "welcome" on channel 1;
10: $U_1 := U_1 \cup \{id_j\}$;
11: **if** $count \geq \frac{2d' \log n_t}{e^2}(1 - \frac{2}{\tilde{b}})$ **then**
12: $U_1 := U_1 \cup \{$"$\tilde{b}$ too small"$\}$;
13: **if** $i_v = r_1^{sender}$ **then**
14: send message "stop joining" on channel 1;

We now describe the procedure the nodes in S_1 follow to detect if the current estimate for $\tilde{b}$ is in the correct order of magnitude. The representative sender r_1^{sender} transmits messages on channel one every second time slot reserved for the joiners until it tried $d' \log n_t$ times for some constant d' to be defined later (line 3 of ALGORITHM 4). Hence, every second opportunity for new nodes to join is blocked $d' \log n_t$ times. The other nodes in S_1 count the number of times the representative sender of S_1 transmits successfully using the variable *count* (line 6). If the nodes receive a message from a joiner with id_j, the representative sender replies with a welcome message (line 9). After this loop, the set decides that $\tilde{b}$ is too small for the current number of joiners if *count* is less than a threshold $\tau = 2d' \log n_t \cdot e^{-2} \cdot (1 - 2/\tilde{b})$ (line 11), and lets the other sets know about this in the next step. To this end, all nodes in S_1 insert an additional update item to U_1 which has the highest priority to be forwarded to all other nodes. ALGORITHM 4 describes the behavior of nodes of the network in pseudo-code.

Using Chernoff bounds we show that this decision is correct w.h.p. This procedure only prolongs the period until nodes are detected by a constant factor. For the probability analysis of this procedure we use the following well-known results:

Fact 4.11 (e.g. in [22]). *For all $y \geq 1, |x| \leq y$ the following holds:*

$$e^x \geq \left(1 + \frac{x}{y}\right)^y \geq e^x \left(1 - \frac{x^2}{y}\right)$$

Fact 4.12 (Chernoff-Inequalities, e.g. in [22]). *Let $X_1, \ldots, X_n$ be independent Bernoulli-distributed random variables with $\Pr[X_i = 1] = p_i$ and $\Pr[X_i = 0] = 1 - p_i$. Then the following inequalities hold for $X := \sum_{i=1}^n X_i$ and $\mu := \mathrm{E}[X] = \sum_{i=1}^n p_i$:*

(i) $\Pr[X \geq (1+\delta)\mu] \leq e^{-\mu\delta^2/3}$ *for all* $0 < \delta \leq 1$,

(ii) $\Pr[X \leq (1-\delta)\mu] \leq e^{-\mu\delta^2/2}$ *for all* $0 < \delta \leq 1$.

Lemma 4.13. *For any constants $d_1, d_2 > 0$, there is a parametrization of the algorithm such that the probability that S_1 decides that $\tilde{b}$ is too small even though there are fewer than $\tilde{b}$ joiners is $n_t^{-d_1}$ and the probability that S_1 decides that $\tilde{b}$ is large enough even though there are more than $2\tilde{b}$ joiners is $n_t^{-d_2}$ by choosing d' appropriately.*

Proof. Let X be the random variable counting the number of successful message transmissions for the representative sender of S_1.

Case $j < \tilde{b}$. The expected value of X is

$$\begin{aligned}
\mathrm{E}[X \mid j < \tilde{b}] &= \sum_{i=1}^{d' \log n} \Pr[\textit{no joiner sends in slot } i \mid j < \tilde{b}] \\
&\geq d' \log n_t \cdot \Pr[\textit{no joiner sends in slot } 1 \mid j < \tilde{b}] \\
&\geq d' \log n_t \left(1 - \frac{1}{\tilde{b}}\right)^{\tilde{b}} \\
&\geq d' \log n_t \cdot e^{-1} \left(1 - \frac{1}{\tilde{b}}\right) \quad \geq \quad \frac{d' \log n_t}{2e},
\end{aligned}$$

where the second inequality follows since nodes that appear newly in the single-hop area during the join-detection do not participate (they do not know $\tilde{b}$ and wait for the next time the join-detection step is executed). The last inequality holds for all $\tilde{b} \geq 2$. We assume this to be true since $\tilde{b} \geq \log n_t$, which is asymptotically larger than 2.
Due to Chernoff Bound 4.12 (ii), the probability that fewer than $\tau = 2d' \log n_t \cdot e^{-2} \cdot \left(1 - \frac{2}{\tilde{b}}\right)$ messages are received correctly is

$$\Pr[X \leq \tau \mid j < \tilde{b}] = \Pr[X \leq (1-\delta_1)\mu \mid j < \tilde{b}] \leq e^{-\mu\delta_1^2/2}$$

with $\delta_1 = 1 - \tau/\mu$. The necessary conditions for Chernoff, i.e., $0 < \delta_1 \leq 1$, can easily be validated. Because

$$\delta_1 = 1 - \frac{\tau}{\mu} = 1 - \frac{2e^{-2}\left(1 - \frac{2}{\tilde{b}}\right)}{\left(1 - \frac{1}{\tilde{b}}\right)^j} \geq 1 - \frac{2\left(1 - \frac{1}{\tilde{b}}\right)^{2\tilde{b}}}{\left(1 - \frac{1}{\tilde{b}}\right)^{\tilde{b}}} \geq 1 - 2e^{-1},$$

we write

$$\Pr[X \leq \tau \mid j < \tilde{b}] \leq e^{-\mu\delta_1^2/2} \leq e^{-\gamma_1 d' \log n_t} \leq n_t^{-d_1},$$

for suitable constants γ_1 and d'. By tuning the parameter d', i.e. influencing the number of transmission trials of the representative sender of S_1, d_1 can be

made arbitrarily large, leading to an arbitrarily small probability and thus a result w.h.p.

Case $j > 2\tilde{b}$**.** We observe that $\Pr[X \geq \tau \mid j > 2\tilde{b}] \leq \Pr[X \geq \tau \mid j = 2\tilde{b}]$, so we upper bound only the latter probability. Again, we compute a lower bound on the expected value of X:

$$\begin{aligned} \mathrm{E}[X \mid j = 2\tilde{b}] &= \sum_{i=1}^{d' \log n} \Pr[\textit{no joiner sends in slot } i \mid j = 2\tilde{b}] \\ &= d' \log n_t \left(1 - \frac{1}{\tilde{b}}\right)^{2\tilde{b}} \quad \geq \quad d' \log n_t \cdot e^{-2} \cdot \left(1 - \frac{2}{\tilde{b}}\right) \\ &\geq d' \log n_t \cdot e^{-2} \cdot \frac{2}{5}, \end{aligned}$$

where we used Fact 4.11, and the last inequality follows when we assume that $\tilde{b} \geq 5$. As before, we assume this due to $\tilde{b}$ being at least $\log n_t$, which is asymptotically larger than 5.
Using Chernoff bound 4.12 (i), the probability that more than $\tau = 2d' \log n_t \cdot e^{-2} \cdot \left(1 - \frac{2}{\tilde{b}}\right)$ messages are received correctly is

$$\Pr[X \geq \tau \mid j = 2\tilde{b}] = \Pr[X \geq (1 + \delta_2)\mu \mid j = 2\tilde{b}] \leq e^{-\mu\delta_2^2/3}$$

where $\delta_2 = \tau/\mu - 1$. Again, the necessary conditions for Chernoff, i.e., $0 < \delta_2 \leq 1$, are valid for $\tilde{b} \geq 5$. Because

$$\delta_2 = \frac{\tau}{\mu} - 1 = \frac{2e^{-2}\left(1 - \frac{2}{\tilde{b}}\right)}{\left(1 - \frac{1}{\tilde{b}}\right)^{2\tilde{b}}} - 1 \geq 2\left(1 - \frac{2}{\tilde{b}}\right) - 1 \geq \frac{1}{5}$$

for $\tilde{b} \geq 5$, we obtain

$$\Pr[X \geq \tau \mid j = 2\tilde{b}] \leq e^{-\mu\delta_2^2/3} \leq e^{-\gamma_2 d' \log n_t} \leq n_t^{-d_2}$$

for suitable constants γ_2 and d'. Due to the same reasons as in the latter case, we have obtained a correct result w.h.p. This completes the proof that the probability to make a wrong decision is very small and the nodes can determine whether $\tilde{b}$ was chosen in the correct order of magnitude w.h.p. □

Remark 4.14. *As discussed in Section 2, there are energy-efficient size approximation algorithms. However, letting an unknown number of nodes join cannot be solved with the help of these algorithms, since they do not handle node failures and they do not give high probability results for a small number of joining nodes.*

After joining, the new nodes listen on channel 1 until the end of the current loop. In addition, all nodes have to execute the algorithm described in Section 4.7 to make sure that the new nodes have a complete ID table eventually.

Remark 4.15. *We could use more sets than one (currently S_1) to listen to joining nodes. As we only need to make sure that new nodes join in a constant number of rounds and that the error probability is low, we use only the set S_1 for the sake of simplicity.*

4.4. Disseminate Crash/Join-Information to all Nodes (Step 4)

In the previous sections we discussed how each set S_I detects crashed nodes and accepts new nodes that want to join the network. This information is stored in a (possibly empty) list U_I of *update items*, where each update item consists of the ID of the node it refers to and whether the node has crashed or joined the network. This list U_I needs to be distributed to all other sets. To this end, the representatives of each set communicate with representatives of other sets to compute the set $U = \bigcup_{I=1}^{N} U_I$ of all changes in the network.

Theorem 4.16. *If $b \leq \tilde{b}$, the update items are disseminated within time $O(\tilde{b} + \log n_t)$ with* ALGORITHM 5. *Otherwise, a failure of the algorithm is detected in* STEP *5.*

Idea: First, the sets are mapped to vertices of a communication graph G (in our case this is a tree[1]). This is done deterministically within each node and no messages need to be exchanged. Second, neighboring sets exchange information repeatedly until the information reaches all sets.

Definition 4.17 (Family of communication graphs)**.** *Let $\mathcal{C}$ be an infinite family of undirected communication graphs $C_N = (V_N, E_N)$ over N vertices which have the property that the degree of each vertex is bounded by d_N. Furthermore we require that each C_N can be computed deterministically only from knowledge of N, as well as a function S_N, where*

$$\begin{aligned} s_N : V_N \times \{1, \ldots, l_N\} &\longrightarrow \{1, \ldots, N\} \times \{1, \ldots, N\} \\ (v, t) &\longmapsto (\kappa_{send}, \kappa_{receive}). \end{aligned}$$

This function S_N determines a schedule s_N of length l_N that tells each vertex $v \in V$ that it should send in time slot $t \in \{1, \ldots, l_N\}$ on channel $\kappa_{send} \in \{1, \ldots, N\}$ (denoted by $s_N(v,t)_1$) and receive on channel $\kappa_{receive} \in \{1, \ldots, N\}$ (denoted by $s_N(v,t)_2$) respectively – in such a way that within l_N time slots all neighbors of G are able to exchange exactly one message (containing one piece of information) with each other without collisions. The diameter of a communication graph C_N is denoted by diameter(C_N).

[1] We decided to present the algorithm in this slightly more general way such that it is easy to replace the family of communication graphs. This is useful to handle unreliable communication where information being transported from a leaf to the root is very unlikely. Using expander graphs might help in this case, since they also have logarithmic diameter and constant degree but are more robust: after a short time (say $f(n)$) the information is copied to $2^{f(n)/O(1)}$ nodes with not too small a probability. Compared to the tree, it is more likely that at least one of the many copies of the information reaches the destination.

Definition 4.18 (Trees)**.** *Let $\mathcal{C} := \{C_N \mid N \in \mathbb{N}\}$ be the family of rooted binary trees over N nodes of height $\lfloor \log N \rfloor$. In $C_N := (V_N, E_N)$ we have the vertices $V_N := \{1, \ldots, N\}$ and for each vertex $v \in V_N \setminus \{1\}$ there are edges $(v, \lfloor v/2 \rfloor)$ and $(\lfloor v/2 \rfloor, v)$ connecting v to its parent $\lfloor v/2 \rfloor$.*

Lemma 4.19. *A schedule s_N of length 4 can be computed deterministically for any member C_N of the above tree family.*

Proof. Each node v in odd levels of the tree (that is $\lfloor \log_2(v) \rfloor$ is odd) exchanges one message (both ways) with child $2v$ in the first time slot and with child $2v+1$ in the second time slot – observe that children are in even levels. Then each node v in even levels of the tree exchanges one message (both ways) with child $2v$ in the third time slot and with child $2v+1$ in the fourth time slot. Every node u sends only on its own channel u to avoid collisions – receivers tune to this channel. The complete schedule is given by

$$s_N(v,1) = \begin{cases} (v, 2v) & : \lfloor \log_2(v) \rfloor \text{ odd} \\ (v, \lfloor v/2 \rfloor) & : \lfloor \log_2(v) \rfloor \text{ even} \end{cases} \quad s_N(v,2) = \begin{cases} (v, 2v+1) & : \lfloor \log_2(v) \rfloor \text{ odd} \\ (v, \lfloor v/2 \rfloor) & : \lfloor \log_2(v) \rfloor \text{ even} \end{cases}$$

$$s_N(v,3) = \begin{cases} (v, 2v) & : \lfloor \log_2(v) \rfloor \text{ even} \\ (v, \lfloor v/2 \rfloor) & : \lfloor \log_2(v) \rfloor \text{ odd} \end{cases} \quad s_N(v,4) = \begin{cases} (v, 2v+1) & : \lfloor \log_2(v) \rfloor \text{ even} \\ (v, \lfloor v/2 \rfloor) & : \lfloor \log_2(v) \rfloor \text{ odd} \end{cases}$$

If a channel (vertex) on (to) which a node v should send or listen is not in the range of $\{1, \ldots, N\}$, then v can be sure that the corresponding node does not exist and just sleeps in this slot – this happens for the root and the leaves. □

Corollary 4.20. *The family of trees $\mathcal{C} := \{C_N \mid N \in \mathbb{N}\}$ from* DEFINITION 4.18 *combined with the schedules s_N from* LEMMA 4.19 *is a family of communication graphs, where the diameter $diameter(C_N)$ of C_N is $2 \cdot \lceil \log n_t \rceil$, the degree of each node is bounded by $d_N = 3$ and the length of any schedule s_N is 4.*

We now describe the algorithm in more detail.

In the first part of the algorithm, all nodes start with the same good *ID table*, what we can assume according to INVARIANT 4.3. From the information n_t stored in the ID table, each set v of the N sets computes deterministically without communication (line 1) the communication graph $G := C_N$ as well as the schedule S_N of length l_N.

The second part consists of $O(diameter(G) + \tilde{b})$ phases (one send or receive operation, described in lines 3–10 for the representative sender and lines 2–7 for the representative receiver of Algorithm 5), each of $l_N + d_N$ time slots. During each phase each vertex is able to send one update item to each of its (at most) d_N neighbors and receive one update item from each of its (at most) d_N neighbors. This communication takes place by adhering to the previously computed schedule s_N of length l_N. Thus in each phase each vertex exchanges messages with its neighbors by letting its representatives follow Algorithm 5. The vertices maintain two lists of update items. In the first list U are the items the set knows of, while the second list U' contains the items it has forwarded already. In the first of all phases, the first list is set to $U := U_I$, the list of the IDs determined in the detection step, and the second list $U' := \emptyset$ is empty (line 3). After the completion of the second part, U equals U' and contains all items. In each phase, set S_I sends the information of the lowest ID in $U \setminus U'$

Algorithm 5 Deterministic Dissemination

Sender:

1: compute schedule s_N for $G := C_N := (\underbrace{\{S_1, \ldots, S_N\}}_{\text{vertices } V_N}, \underbrace{E_N}_{\text{edges}})$;
2: $U' := \emptyset$;
3: **for** $t = 1, \ldots, diameter(G) + \tilde{b}$ **do**
4: **for** $j = 1, \ldots, l_N$ **do**
5: $item_{send} := \min_{item \in U \setminus U'}\{D\}$
or "no news" if U empty;
6: send $item_{send}$ on channel $s_N(I_v, j)_1$;
7: $U' := U' \cup \{item_{send}\}$;
8: **for** $j = l_N + 1, \ldots, l_N + d_N$ **do**
9: receive item $item_{receive}$ on channel I_v;
10: $U := U \cup \{item_{receive}\}$;
11: send U on channel I_v;

Receiver:

1: compute schedule s_N for $G := C_N := (\underbrace{\{S_1, \ldots, S_N\}}_{\text{vertices } V_N}, \underbrace{E_N}_{\text{edges}})$;
2: **for** $t = 1, \ldots, diameter(G) + \tilde{b}$ **do**
3: **for** $j = 1, \ldots, l_N$ **do**
4: receive $item_j$ on channel $s_N(I_v, j)_2$;
5: **for** $j = l_N + 1, \ldots, l_N + d_N$ **do**
6: send $item_j$ on channel I_v
unless it is "no news";
7: $U := U \cup \{item_j\}$;

to its (at most) d_N neighbors and receives (at most) d_N update items from its d_N neighbors. Depending on the outcome of each phase, the lists U and U' are updated.

See ALGORITHM 5 for a description in pseudo-code.

First we show that exchanging messages with neighboring vertices is possible for two representatives in each set within time $l_N + d_N$ if none of them crashes (LEMMA 4.21). We argue later in LEMMA 4.23 that we can tolerate $\tilde{b}$ crashes during the execution and in SECTION 5 we establish a time/energy/channel trade-off for fewer channels.

Lemma 4.21. *All sets transmitting update items to their (at most) d_N neighbors and receiving (at most) d_N update items from their (at most) d_N takes time $l_N + d_N$ when the number of channels N is equal to the number of sets and no node crashes.*

Proof. We adhere to the schedule s_N. As we noted before, all nodes computed the same graph G and schedule s_N such that all global communication activities are consistent with the local computation of v. This takes l_N time. Afterward the receiver $r_I^{receiver}$ reports the newly received update items (there are at most

d_N, one from each neighbor) to r_I^{sender} on the set's channel I during time slots $l_N+1, \ldots, l_N+d_N$ of this phase (lines 5–6 of the receiver's part). r_I^{sender} receives this information and adjusts U and U' accordingly (lines 8–10 of the sender's part). All these computations happen in a deterministic way based on the same information (stored in each node) and yield the same schedule for the whole graph in each node. □

Observe that no set (vertex) crashes completely as the adversary is bounded to let at most $\tilde{b}$ nodes crash during the execution of the algorithm. Hence there are $\tilde{b}$ nodes ready to replace the representatives. In LEMMA 4.23 we prove that repeating the procedure from LEMMA 4.21 $O(diameter(G) + \tilde{b})$ times leads to full knowledge of U. First we prove a weak version of this lemma (LEMMA 4.22). We extend this lemma to hold despite crashes during execution (LEMMA 4.23).

Lemma 4.22. *All vertices can learn the set U that contains all update items after $O((diameter(G) + \tilde{b}) \cdot (l_N + d_N))$ time slots if no nodes crash during the execution of this algorithm.*

Proof. W.l.o.g., let $U := \{item_1, \ldots, item_{\tilde{b}}\}$ be a sorted list of update items. By induction on i we prove that $item_i$ is known to all vertices S_I in G after $O((diameter(G) + i) \cdot (l_N + d_N))$ time slots of executing ALGORITHM 5 if no nodes crash during the execution.

Base case $i = 1$: Any representative v that receives $item_1$, always immediately communicates $item_1$ to its neighbors in the next phase since $item_1$ is the first item in v's sorted list $U \setminus U'$. Thus item $item_1$ has been broadcast to all nodes after $diameter(G) + 1$ phases if no nodes crash during this computation.

Inductive step $i \to i + 1$: Let us assume the induction hypothesis for i. Item $item_{i+1}$ can only be delayed (in line 5 of the sender's part) by items with smaller indices. Let $item_j$ be the item with the largest index that delays $item_{i+1}$ on any of the shortest paths to any of the vertices in G. Then $item_{i+1}$ is known by all vertices in G one phase after $item_j$. By the induction hypothesis, this is after $diameter(G) + j + 1$ phases. We remember $j \leq i$ to obtain the induction hypothesis for $i + 1$. □

Lemma 4.23. LEMMA 4.22 *remains true even when up to $\tilde{b}$ nodes crash during execution.*

Proof. If a representative crashes during the dissemination step (either a sender or a receiver), a replacement node realizes the crash of its representative at most one phase later since the replacement is listening to all actions of the representatives and thus detects whether it sent all messages it was supposed to send. If it did not send a message during a phase it must have crashed and the next replacement node steps up to be the new representative (in case no information needs to be sent by a representative in a time slot it does not matter whether it crashed). This is possible since the replacement nodes have exactly the same information as the representative and know when the representative should send what message. For the same reason they are able to know how many replacements happened before and thus when it is their turn to jump in

to retransmit the necessary message in the next phase after the crash. Since at most $\tilde{b}$ nodes can crash, there are never more than $\tilde{b}$ retransmissions necessary. This can lead to a delay of at most $\tilde{b}$ phases and the statement follows. □

Proof of THEOREM 4.16. We combine LEMMA 4.22 and LEMMA 4.23 as well as use the fact that in the communication graphs provided by the tree family from COROLLARY 4.20, for all values of $N \in \mathbb{N}$ we have $diameter(C_N) = O(\log n_t), d_N = 3$ and that the schedule-length of s_N is $l_N = 4$. □

As a consequence, all representatives and replacements of S_I know all the update items available after $O(\log n_t + \tilde{b})$ time slots. Thus all nodes in any set S_I are aware of all crashed and new nodes at the time when the algorithm started (and also of some crashes/joins that happened during the algorithm's execution, but not necessarily all of those). N channels are used (one for each set).

4.5. Stop if Burst too Large (Step 5)

In this step, the sets determine whether the algorithm failed due to too large a burst – that is more than $\tilde{b}$ nodes joined or crashed (within time $c\tilde{b}$). To distinguish sets that do not have any information to forward from sets of which all members crashed, we let each set S_I send "I'm here!" in its scheduled time slots without new information to be sent.

Theorem 4.24. *If $b > \tilde{b}$ then $O(\log n_t + \tilde{b})$ time slots after the dissemination step all nodes have the same information: Either they have noticed that the burst is too large and stopped the execution or all have the same information on network changes.*

Proof. Set S_1 knows with high probability if too many nodes tried to join and forwarded this information in the dissemination step. Thus all nodes are aware of this event at the end of step 4 of the FBMA $A_{\tilde{b}}$ if it occurs: if the decision of S_1 is wrong, the algorithm still works properly, it just takes longer until all nodes which join the network are included, however, all nodes receive the same information.

If one or more sets S_I completely crash before or during step 4, its neighbors immediately know that more than $\tilde{b}$ nodes crashed and the algorithm might fail (e.g. the communication graph might be disconnected and not all nodes have been delivered the same information). The neighbors of S_I then broadcast this information through the communication graph with highest priority. Even if further sets crash completely and the failure message originated by the neighbors of S_I does not reach all sets, the neighbors of the other crashed sets start propagating such a message through the network as well. After $O(\log n_t + \tilde{b})$ phases (one send or receive operation constitutes a phase, described in lines 3-10 for sender and lines 2-7 for receiver of Algorithm 5) all representatives are informed if one or more sets did not receive all the information: If no set crashed then after $\log n_t + \tilde{b}$ phases all sets have all the update items. If a set crashes before all sets have this information, then $\log n_t + \tilde{b}$ phases later all

sets are informed of a failure, no matter how many sets crash now. If a set crashes afterward, the update information has reached all sets already and thus all surviving sets can continue with this information.

The last possibility of an adversary to disturb the self-monitoring process consists in letting more than $\tilde{b}$ nodes crash even though all sets survive. By extending the dissemination phase by a constant number of time slots, we can ensure that all sets notice if more than $\tilde{b}$ update items have been disseminated and conclude that the adversary exceeds the bound of $\tilde{b}$. Therefore, also in this case a potential failure of the algorithm is known to all nodes after the dissemination step. Thus, the algorithm guarantees that all sets have the same set of update items at the end of a successful round if it did not stop the execution. □

4.6. Participating with a fragmentary ID Table

Note that the joiners can already participate in the algorithm without knowing the whole ID table: When a new node v is detected by the network, the node that is the oldest in the network according to the timestamp (ties are broken by ID) tells v the smallest ID of a node w in the network that is larger than v's ID. This is possible since the oldest node is guaranteed to have a good ID table. Joiner v now assumes to have this position in the ID table. After the dissemination step has finished, node v determines the number $c^<$ of crashed nodes with IDs smaller than v and subtracts $c^<$ from its assumed position. Then v counts $j^<$, the number of nodes that joined the network with an ID smaller than itself and adds $j^<$ to its assumed position. Thus there is only one node in the network assigned to a position in the ID table after updating the ID tables based on the information gathered in the dissemination step. Knowing this position in the ID table allows the joiner to participate in all the necessary algorithms: partition / crash and join detection / information dissemination. In the next section we describe how the new nodes can fill their ID table with entries for the existing nodes.

4.7. Learning the ID Table

In order to allow new nodes to learn the IDs of the nodes that are already in the network, the existing nodes alternately transmit their IDs and the time slot when they arrived on channel 1. This process can be interleaved with the execution of the monitoring algorithm, i.e., odd time slots can be used for the monitoring algorithm while even time slots are reserved for getting to know all existing nodes. The sequence in which the nodes announce their presence is ordered by the timestamp of their arrival (in case of ties by their ID) as stored in the ID table. Let us consider joiner j which attached to S_1 in time slot t. Let the number of nodes currently in the network be n_t (this number includes the nodes that have joined in previous rounds and are not yet announcing their ID at time t). In the following node j listens to the announcements in even time slots and enters the information obtained into its ID table. As soon as a joiner j receives such an announcement for the second time, it starts announcing itself

at the appropriate point in time. To make sure that j learns those nodes that joined shortly before j and have not inserted themselves into the announcement procedure during the first traversal of the sequence, joiner j keeps listening until it hears an announcement for the third time. Observe, that there is always a node that announces itself three times as long as we set c to be larger than 6 (since then not all nodes can crash during this period of time). This guarantees, that joiner j learns all the IDs of nodes in the network and this procedure is completed in $O(n_t + b)$ time slots. At the same time it is informed about nodes that crashed or joined during the time it learned the ID table and can update it continuously. Now j's ID table contains all IDs and is therefore a good ID table. The future time slots it has to announce itself can be computed internally based on the information in the ID table.

5. Lower Bound and Trade-Offs

Until now the algorithm is asymptotically optimal up to an additive $O(\log n_t)$ term due to being linear in b for $b = \Omega(\log n_t)$ as we need at least b time to disseminate b information items. The following lemma shows that also the additive $O(\log n_t)$ term is necessary.

Lemma 5.1. *In order to detect which nodes crashed without mistakingly declare an active node as missing any (randomized) algorithm using k or fewer channels has time complexity $\Omega(\frac{n_t}{k})$.*

Proof. We prove the lower bound for algorithms solving the simplified problem of detecting which node crashed in the case that exactly one node v chosen uniformly at random fails. Showing a lower bound for this case implies a lower bound for the self-monitoring problem, since this reflects the case $b = 1$, which needs to be covered.

Up to a constant factor, the only way to be sure a node u has not crashed is to receive a message sent by u. Thus if u does not send a "hello" message in the time slot assigned to it, one can assume it crashed. As there is no mechanism available for collision detection, we can use techniques where several nodes use the same channel simultaneously only in a very limited way, especially since there is exactly one node that crashed: It is possible to obtain a speed-up of two by asking pairs of nodes to send a message in each time slot. Since we know that exactly one node crashed, there is exactly one node that can transmit a message successfully (the messages of other pairs collide) and we can derive which node did not send. However it is not possible to get a higher speed-up – assume more than two nodes are sending on the same channel, then no matter whether one of them crashed or not there will be a collision and we cannot derive which node crashed. This also implies that there is no better solution in a randomized setting.

Thus (when ignoring constant factors) any algorithm must ensure that for each node u_i there is a time slot and a channel to send its "hello" message successfully without collisions. Let X be the random variable denoting how

many nodes an algorithm did not listen to before finding the crashed node v (i.e., not receiving a message from v in the time slot assigned to it). Since the crashed node v was chosen uniformly at random, any algorithm does not listen to $\mathbb{E}[X] = n_t/2$ nodes in expectation.

Using the Markov inequality stating that $\Pr[X \geq \lambda \cdot \mathbb{E}[X]] \leq 1/\lambda$ we derive $\Pr[X \geq \frac{n_t}{2} \cdot \frac{3}{2}] \leq 2/3$ which implies that there is no Monte-Carlo algorithm working correctly with probability greater than $2/3$ and that needs to check fewer than $n_t - 3 \cdot n_t/4 = n_t/4$ nodes.

The above does not take into account the fact that we should not mistakingly declare an active node as missing. Remark that this would only increase the lower bound and thus we do not take care of this anymore. Since on each of the k channels we can only check one node at a time, we need $\Omega(\frac{n_t}{k})$ time to check $\Omega(n_t)$ nodes. □

The previous analysis implies that the time complexity of our algorithm is asymptotically optimal for all values of b, since the monitoring algorithm presented in this paper uses only $O(n_t/\log n_t)$ channels and any algorithm needs $\Omega(\log n_t)$ time in this case.

Now that we have a lower bound depending on the channels, we are interested in a trade-off for the time complexity upper bound that arises if the number of channels and the energy available are limited. In the presentation of our Algorithm 1 we need one channel for each of the sets (in step 2 the sets use a separate channel each for communication within the set, in step 4 and 5 the same channels are used for inter-set communication, in step 3 only one channel is used). So far we assumed that there are as many channels available as there are sets, i.e., the number of channels used is $\Theta(\max_{time\ t}(n_t/\log(n_t)))$. This is due to the fact that each iteration of the monitoring algorithm starts with $\tilde{b}_1 = \log n_t$ and thus needs $O(n_t/\log n_t)$ channels, one for each of the $N = \lceil n_t/(2\log n_t + 2)\rceil - 1$ sets. Since in typical wireless networks the number of channels is fixed, we show now how to modify our algorithms to work for networks with k channels.

Lemma 5.2. *For $(c, \tilde{b})$-adversaries, the* FBMA *$A_{\tilde{b}}$ (*Algorithm 1*) implies a trade-off between the time complexity and the number of channels necessary for known $\tilde{b}$, i.e., the time complexity is $O(\tilde{b} + \log n_t + \frac{n_t}{k})$. For $\tilde{b} > \log n_t$ this trade-off is asymptotically optimal.*

If only k channels are available and the burst size b is unknown, the FBMA *family $\{A_{\tilde{b}}\}_{\tilde{b}\in\mathbb{N}}$ implies a time complexity of $O\left(b + \log\left(\frac{b}{\log n_t}\right)\frac{n_t}{k} + \frac{n_t}{k} + \log n_t\right)$ for a monitoring algorithm.*

Proof. Each time slot of step 2 and 3 of the algorithm is spread across $\lceil N/k\rceil$ time slots: set S_I is assigned to channel $I \mod k$ and executes the corresponding action in the $\lceil I/k\rceil^{th}$ time slot. In all other time slots the nodes of the set sleep. The schedule for step 4 is stretched analogously and adapts the assignment of channels and time slots for transmission and reception. Apart from time slots when a transmission or reception of the set is scheduled all nodes

of the set sleep. Thus each round of $A_{\tilde{b}}$ takes a factor $\lceil N/k \rceil$ longer. Since one round of the algorithm with N channels is completed in $O(\max(\tilde{b}, \log n_t))$ and $N \in O(n_t / \max(\tilde{b}, \log n_t))$, the time complexity with k channels is thus $O(\tilde{b} + \log n_t + n_t/k)$. This is asymptotically optimal, because of the capacity of a k-channel medium. Every node needs an opportunity to successfully send an "I am still here" message, which cannot be achieved in fewer than $\Omega(n_t/k)$ time slots. If fewer time slots are available, some nodes are mistakenly assumed to have crashed. For the dissemination of the information at least $\Omega(\tilde{b})$ time slots are necessary, because nodes can only receive messages on one channel at a time.

We can derive the result for unknown burst size analogously by modifying the proof of THEOREM 4.1. If $b < \log n_t$, only one round is executed, which leads to a runtime of $O(\log n_t + n_t/k)$. If $b \geq \log n_t$, as we have just shown the time for one execution of the loop rises from $O(b_i)$ to $O\left(b_i + n_t/k\right)$ and thus

$$\begin{aligned}
\sum_{i=1}^{\log\left(\frac{b}{\log n_t}\right)} O\left(b_i + \frac{n_t}{k}\right) &= \sum_{i=1}^{\log\left(\frac{b}{\log n_t}\right)} O\left(2^{i+1} \log n_t + \frac{n_t}{k}\right) \\
&= O\left(2^{\log \frac{b}{\log n_t}} \log n_t + \log\left(\frac{b}{\log n_t}\right) \frac{n_t}{k}\right) \\
&= O\left(b + \log\left(\frac{b}{\log n_t}\right) \frac{n_t}{k}\right).
\end{aligned}$$

Hence, in the case of a bounded number of channels k and unknown burst size b we obtain an overall runtime of $O\left(b + \log\left(\frac{b}{\log n_t}\right) \cdot \frac{n_t}{k} + \frac{n_t}{k} + \log n_t\right)$. □

Apart from its resilience against c-adversaries, we measure an algorithm's quality by its *energy usage*. Note that all our algorithms are energy-balanced in the sense that the highest energy consumption among all nodes is only a constant factor larger than the lowest energy consumption. So far, we focused on algorithms that detect missing or new nodes and disseminate this information in as few time slots as possible. To this end, the nodes are either in state `transmit` or `receive` in a constant fraction of the time slot. We now analyze scenarios requiring that the radio communication system of each node is on during at most e time slots in every interval of $T \geq \max(e, c \cdot b)$ consecutive time slots. In other words a delay of a factor T/e is tolerated in exchange for prolonged network lifetime. We define the energy consumption e_t of an algorithm associated with time slot t to be the maximum energy consumption of all nodes $m \in M$ over the next T time slots. Additionally, let $\sigma_i(m) \in \{\texttt{transmit}, \texttt{receive}, \texttt{sleep}\}$ denote the state of the node m in time slot i. Then e_t is formally given by

$$e_t := \max_{m \in M} |\{i \mid \sigma_i(m) \in \{\texttt{transmit}, \texttt{receive}\}, t \leq i < t + T\}|.$$

A lower bound on the energy consumption for a burst of size b is $\Omega(b)$, since every node has to listen to a channel during at least $\Omega(b)$ time slots to be informed about all events. Thus algorithms with a maximum energy consumption $O(b)$

are optimal. We now show how to modify our algorithms for optimal energy consumption.

Theorem 5.3. *By adding $\lceil T/2 - c \cdot b/2 \rceil$ time slots at the end of each round of the* FBMA $A_{\tilde{b}}$ *we can construct monitoring algorithms with optimal energy complexity for $T \geq c \cdot b$.*

Proof. One round of the FBMA $A_{\tilde{b}}$ consists of $\lceil \tilde{b} \cdot c/2 \rceil$ time slots to guarantee that every node of the network is updated at most two rounds after an event happened. In the time slots added at the end of each round, all nodes sleep. Thus the additional time slots ensure that all update items have reached all nodes in T time slots while at the same time the energy consumption is kept at $c \cdot b$. This implies that we can modify the monitoring algorithm as constructed in THEOREM 4.1 as well to guarantee that time and energy complexity are optimal. □

6. Conclusion

This paper presents an algorithm that notifies the participants of a wireless network of crashed and new members. This algorithm can be used as service to build applications that rely on the nodes to have a consistent view of the network. Despite the fact that we considered nodes without the possibility to distinguish collisions from noise on the channel, the algorithm reacts to events fast and continuously adapts to the number of nodes that fail or arrive. Apart from the procedure for joining nodes, which is inherently randomized, the algorithm is deterministic and ensures that the participants exchange few messages in order to save energy. Open problems for future work include to further explore multihop networks, to extend the algorithm to cope with adversaries that prevent communication on certain channels (jamming) or to investigate the possible speed up if larger messages can be transmitted.

[1] A. Bakshi and V.K. Prasanna. Energy-Efficient Communication in Multi-Channel Single-Hop Sensor Networks. In *International conference on parallel and distributed simulation (ICPADS)*, page 403-414, 2004.

[2] R. Bar-Yehuda, O. Goldreich, and A. Itai. Efficient Emulation of Single-Hop Radio Network with Collision Detection on Multi-Hop Radio Network with No Collision Detection. *Distributed Computing*, volume 5(2), pages 67-71, 1991.

[3] M. Bienkowski, M. Klonowski, M. Korzeniowski, and D. R. Kowalski. Dynamic Sharing of a Multiple Access Channel. In *27th Int. Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 83-94, 2010.

[4] I. Caragiannis, C. Galdi, and C. Kaklamanis. Basic computations in wireless networks. In *Proceedings of the 16th Annual International Symposium on Algorithms and Computation (ISAAC)*, Lecture Notes in Computer Science, volume 3827, pages 533-542, 2005.

[5] B. S. Chlebus, L. Gasieniec, A. Gibbons, A. Pelc, and W. Rytter. Deterministic broadcasting in ad hoc radio networks. *Distributed Computing*, volume 15(1), pages 27-38, 2002.

[6] B. S. Chlebus, D. R. Kowalski, and A. Lingas. Performing work in broadcast networks. *Distributed Computing*, volume 18(6), pages 435-451, 2006.

[7] B. S. Chlebus, D. R. Kowalski, and M. A. Rokicki. Maximum throughput of multiple access channels in adversarial environments. *Distributed Computing*, volume 22(2), pages 93-116, 2009.

[8] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys (CSUR)*, volume 33(4), pages 427-469, 2001.

[9] D. R. Kowalski Chryssis Georgiou, S. Gilbert. Meeting the deadline: On the complexity of fault-tolerant continuous gossip. In *Proceeding of the 29th ACM SIGACT-SIGOPS Symposium on Principles of distributed computing (PODC)*, pages 247-256, 2010.

[10] A. E. F. Clementi, A. Monti, and R. Silvestri. Distributed broadcast in radio networks of unknown topology. *Theoretical Computer Science*, volume 302(1-3), pages 337-364, 2003.

[11] S. Dolev, S. Gilbert, R. Guerraoui, and C. Newport. Secure communication over radio channels. In *Proceeding of the 27th ACM SIGACT-SIGOPS Symposium on Principles of distributed computing (PODC)*, pages 105-114, 2008.

[12] S. Gilbert, R. Guerraoui, D. Kowalski, and C. Newport. Interference-resilient information exchange. In *Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 2249-2257, 2009.

[13] T. Jurdzinski, M. Kutylowski, and J. Zatopianski. Energy-efficient size approximation of radio networks with no collision detection. In *8th Annual International Conference on Computing and Combinatorics (COCOON)*, Lecture Notes in Computer Science, volume 2387, pages 279-289. Springer, 2002.

[14] J. Kabarowski, M. Kutylowski, and W. Rutkowski. Adversary immune size approximation of single-hop radio networks. *Theory and Applications of Models of Computation (TAMC)*, Lecture Notes in Computer Science, volume 3959, pages 148-158, 2006.

[15] M. Kik. Merging and Merge-Sort in a Single Hop Radio Network. In *Theory and Practice of Computer Science (SOFSEM)*, Lecture Notes in Computer Science, volume 3831, pages 341-349, 2006.

[16] M. Klonowski, M. Kutyłowski, and J. Zatopianski. Energy Efficient Alert in Single-Hop Networks of Extremely Weak Devices. In *Workshop on Algorithmic Aspects of Wireless Sensor Networks (ALGOSENSORS)*, Lecture Notes in Computer Science, volume 5304, pages 139-150, 2009.

[17] D. R. Kowalski and A. Pelc. Leader Election in Ad Hoc Radio Networks: A Keen Ear Helps. In *International Conference on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, volume 5556, pages 521-533, 2009.

[18] M. Kutyłowski and D. Letkiewicz. Computing Average Value in Ad Hoc Networks. *Mathematical Foundations of Computer Science (MFCS)*, Lecture Notes in Computer Science, volume 2747, pages 511-520, 2003.

[19] M. Kutyłowski and W. Rutkowski. Adversary Immune Leader Election in Ad Hoc Radio Networks. In *11th Annual European Symposium on Algorithms (ESA)*, Lecture Notes in Computer Science, volume 2832, pages 397-408, 2003.

[20] C. Lavault, J. F. Marckert, and V. Ravelomanana. Quasi-optimal energy-efficient leader election algorithms in radio networks. *Information and Computation*, volume 205(5), pages 679-693, 2007.

[21] K. Nakano and S. Olariu. Energy-efficient initialization protocols for radio networks with no collision detection. *IEEE Transactions on Parallel and Distributed Systems*, volume 11(4), pages 851-863, 2000.

[22] M. Mitzenmacher and E. Upfal. Probability and computing: Randomized algorithms and probabilistic analysis. Cambridge University Press, 2005.

[23] S. Rhea, D. Geels, T. Roscoe and J. Kubiatowicz. Handling churn in a DHT. In *USENIX Annual Technical Conference*, pages 127-140, 2004.

[24] M. Singh and V. K. Prasanna. Optimal energy-balanced algorithm for selection in a single hop sensor network. In *IEEE International Workshop on Sensor Network Protocols and Application (SNPA)*, pages 9-18, 2003.

[25] M. Singh and V. K. Prasanna. Energy-optimal and energy-balanced sorting in a single-hop wireless sensor network. In *IEEE International Conference on Pervasive Computing and Communications (PERCOM)*, pages 50-59, 2003.

[26] G. Stachowiak, T. Jurdzinski. Probabilistic Algorithms for the Wakeup Problem in Single-Hop Radio Networks. *Theory of Computing Systems*, volume 38(3), pages 347-367, 2005.