Brief Announcement: Self-Monitoring in Dynamic Wireless Networks

Stephan Holzer[1] Yvonne Anne Pignolet[2] Jasmin Smula[1] Roger Wattenhofer[1]
[1]ETH Zurich, Switzerland, [2]IBM Research, Switzerland
{stholzer, smulaj, wattenhofer}@tik.ee.ethz.ch, yvo@zurich.ibm.com

ABSTRACT

Wireless networks often experience a significant amount of churn, the arrival and departure of nodes. We propose a distributed algorithm that detects churn and is resilient to a worst-case adversary. The nodes of the network are notified about changes quickly, in asymptotically optimal time up to an additive logarithmic overhead.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms, Reliability, Theory

1. INTRODUCTION

In traditional (wired) distributed systems the *group membership problem* has been studied thoroughly (we refer to [2] 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. Imagine for example a bunch of wireless sensors, distributed in an area, to observe that area. From time to time some of the nodes will fail, maybe because they run out of energy, maybe because they are maliciously destroyed. On the other hand, sometimes more sensors are added. Despite this churn, 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 use the name self-monitoring instead of group membership.

We present an algorithm for the self-monitoring problem that is efficient in several ways, in an adversarial setting. Since energy as well as channels are scarce resources for wireless devices, we evaluate a trade-off between energy and delay / runtime for single- and multi-channel networks in the full paper.

2. MODEL AND DEFINITIONS

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, every node can communicate with every other node directly (single-hop) without collision detection as in the radio network model [3]. New nodes may join the network at any time, and nodes can leave or crash without notice. We exclude Byzantine behavior and assume that as soon as a node crashes, it does not send any messages anymore. Thus the number of nodes in the network varies over time. 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. Further, we assume that the number of properly divided communication channels is sufficiently large. The nodes have sufficient memory and computational power to store an *ID table* containing all IDs of currently participating nodes. n_t denotes the number of entries in the ID table at time t.

At any time, an adversary may select arbitrary nodes to crash, or it may let new nodes join the network. However, the adversary may not modify or destroy messages. Since messages have bounded size, nodes can learn at most a constant number of identifiers per message. As each node can only receive at most one message per time slot, any algorithm needs at least $c_{\min}$ time units on average (for some constant $c_{\min}$) to learn about *one* crash or join. 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. [1] for the maximum tolerable average message rate in a dynamic broadcast setting). Therefore we define an adversary and monitoring algorithm as follows: denote by α the number of crashes/joins that happen in a maximal burst and by $\tilde{\alpha}$ the maximal burst-size that an algorithm tolerates. Both α and $\tilde{\alpha}$ can vary over the time.

Definition 1 (c-Adversary, $(c, \tilde{\alpha})$-Adversary). *We call an adversary a* c-adversary *if it lets nodes join and crash arbitrarily as long as it does not crash the whole network at any time and on average the adversary joins/crashes 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 letting the algorithm fail. A* $(c, \tilde{\alpha})$-adversary *is a c-adversary whose churn is bounded by a constant $\tilde{\alpha}$ during every period of $c \cdot \tilde{\alpha}$ time slots.*

3. MONITORING ALGORITHM

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 $c > c_{\min}$. 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 message size to learn newly joining (or crashed) nodes is bounded. The algorithm can also tolerate churn while trying to recover from previous bursts; again the only limit

PODC'10, July 25–28, 2010, Zurich, Switzerland.
ACM 978-1-60558-888-9/10/07.

is the learning rate of $r_{\max}$ IDs per time unit. 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 churn takes time, depending on the bursts. If there is a burst of b joins or crashes, an optimal algorithm needs at least $b \cdot c_{\min}$ time until the corresponding information at all nodes is up-to-date. Similarly, our algorithm needs time $b \cdot c$. If bursts happen while recovering from previous bursts, delays will take longer due to the constant learning rate. Up to a logarithmic additive term, the learning delay of the algorithm is asymptotically optimal: the algorithm handles the maximum average rate of churn any algorithm can tolerate in this communication model.

Our algorithm is randomized. However, randomness is only required for detecting new nodes since this part cannot be done in a deterministic fashion. All other parts of the algorithm are deterministic, which might be of interest in a setting where only updates on crashed nodes is needed and no nodes join the network. Furthermore, beside an additive logarithmic term, our algorithm exchanges only a constant factor more messages than the number of messages required by an optimal algorithm.

THEOREM 1. *We construct a* Monitoring Algorithm *that can handle c-adversaries (for a constant c) with logarithmic additive overhead: $O(\alpha + \log n)$ time slots after an event all nodes have updated their ID tables.*

Proof Idea: We construct a family of "fixed burst" monitoring algorithms $\{A_{\tilde{\alpha}}\}_{\tilde{\alpha}\in\mathbb{N}}$ that tolerate churn bursts of size $\tilde{\alpha}$ and might fail if the churn is larger. That is: $A_{\tilde{\alpha}}$ can handle a $(c, \tilde{\alpha})$-adversary. From this family we build a monitoring algorithm for handling arbitrary churn without knowing its size α beforehand by estimating the burst size α to be $\tilde{\alpha}$, starting with $\tilde{\alpha} = \log n_t$, and doubling the estimated size $\tilde{\alpha}$ each time $A_{\tilde{\alpha}}$ fails. When the correct order of burst size is reached, $A_{\tilde{\alpha}}$ works fine. This algorithm can adjust to bursts of arbitrary size and never fails for a c-adversary.

In the remainder of this article we describe the "fixed burst"-monitoring algorithms (FBMA) $A_{\tilde{\alpha}}$ in the family mentioned in the proof idea above. Each $A_{\tilde{\alpha}}$ requires the following INVARIANT 1 at all times.

INVARIANT 1. *All nodes that have been in the network for $\Theta(n_t)$ time slots have the same view of the network, i.e., their* ID table *always contains the same entries. Nodes that joined more recently know their position 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 sorted *ID table* at all nodes.

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

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

In brief, FBMA repeats the following six steps to maintain up-to-date information in the *ID tables* of the nodes. The algorithm is fully distributed and does not need a central entity to control its execution. The six steps ensure that INVARIANT 1 holds at the beginning of each loop.

Step 1 – partition nodes into sets: Nodes are divided into $N \in O(1 + n_t/\tilde{\alpha})$ sets $V := \{S_1, \ldots, S_N\}$. Based on the information in their *ID table*, the nodes can determine to which set $S_I \in V$ they belong by following a deterministic procedure. Each set appoints nodes as representatives of the set and their replacements in case they crash. Every node has the necessary information in its *ID table.* Time $O(1)$.

Step 2 – detect crashed nodes in each set on separate channels in parallel: 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 a list of so-called update items U_I (information to disseminate). Time $O(\min(\tilde{\alpha}, n_t))$.

Step 3 – detect joined nodes: New nodes listen to learn the tolerated burst size $\tilde{\alpha}$ and when to try joining. They send requests to join to S_1 with probability $1/\tilde{\alpha}$. In expectation at least one node can join in constant time if the estimate $\tilde{\alpha}$ is in $O(\alpha)$. Detected joiners are added to U_1 together with a note that they joined. After $O(\tilde{\alpha} + \log n_t)$ time slots S_1 decides whether the estimate $\tilde{\alpha}$ needs to be doubled due to too many joiners. Its decision is correct whp (with high probability: with probability greater than $1 - n_t^{-\gamma}$ for each constant γ). Time $O(\tilde{\alpha} + \log n_t)$.

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 $\tilde{\alpha}$ replacements to take over its job. No collisions occur due to the schedule. Time $O(\tilde{\alpha} + \log n_t)$.

Step 5 – stop if burst too large: If the adversary is too strong, information on some of the sets is missing, or more than $\tilde{\alpha}$ nodes crashed or tried to join a single set. In this case, all nodes are notified and the execution of the algorithm stops. Time $O(\tilde{\alpha} + \log n_t)$.

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 1 holds. Time $O(1)$.

4. REFERENCES

[1] B. Chlebus, D. Kowalski, and M. Rokicki. Maximum throughput of multiple access channels in adversarial environments. *Distributed Computing*, 22(2):93–116, 2009.

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

[3] T. Jurdzinski, M. Kutylowski, and J. Zatopianski. Energy-efficient size approximation of radio networks with no collision detection. In *COCOON 2002.*