2012 IEEE 11th International Symposium on Network Computing and Applications

Local Algorithms for Robust Mission Realization in Large-Scale Disconnected Networks

Michel Charpentier, Radim Bartos and Ying Li
Department of Computer Science, University of New Hampshire, Durham, NH 03824
Email: {charpov,rbartos,yws2}@cs.unh.edu

***Abstract*—Robustness and reliability in large-scale sensor networks present a triple challenge. First, the large scale (possibly compounded by the use of unreliable communication) makes failures—transient and permanent—unavoidable. Second, the difficulty of deploying such systems, especially in challenging environments, makes it next to impossible to replace failed nodes or rearrange surviving nodes after each failure. Third, the limited resources and capabilities of most sensor nodes prohibit elaborate, global/centralized approaches, especially in view of the potentially large number of nodes in the network. In this paper, we propose simple mechanisms to alleviate the effect of node failure in large-scale disconnected networks and show that beneficial global network properties can emerge from local rules achievable by simple nodes.**

I. Context and Problem Definition

As distributed systems increase in scale (number of nodes) and use unreliable communication such as wireless, new issues emerge that were not prominent in more traditional systems. We have started to explore a model tailored to large-scale systems of coordinated agents deployed in challenging environments, of which sensor networks are a notable example.

A key problem, which this paper focuses on, is one of reliability and robustness. The main challenge stems from two characteristics of such networks: Firstly, the scale and unreliable communication make individual failures—transient and permanent—unavoidable; secondly, the cost and complexity of deployment make it impossible to restructure the system after each failure. These constraints result in systems that must continue to perform their primary mission in spite of individual failures but cannot rely on a central management to react in a comprehensive way. As an example, consider sensors dropped by airplane in a jungle or embedded into some substratum like road pavement. These sensors *will* fail and won't be repaired or replaced after each failure. Deployed entities thus need to mitigate failures without the freedom to restructure the entire network. The difficulty is compounded by the fact that these entities tend to be limited in resources such as energy, computational power or memory.

This paper defines several operations in the context of a model of coordinated interactions. These operations are triggered by nodes to alleviate the effect of failures and are characterized by their *locality* and their *simplicity*. They aim to answer the question: Are there simple steps that nodes can take, after they detect a failure and in coordination with their immediate neighbors, from which the health of the entire network will globally improve?

II. A Model of Disconnected Networks

To properly describe and evaluate our algorithms, we have defined a model of interaction and cooperation [1]. In this section, we only introduce aspects of the model relevant to the paper.

Our model is one of agents, such as sensor nodes, deployed for a particular mission that involves data acquisition, distributed computation and actuation is response to environmental events. The primary focus of these agents is the realization of their mission, for which communication and interaction is necessary at some times. In other words, ours is a *mission* network, not a communication network.

As an illustration, consider the system described in [2], in which the wings of an airplane are equipped with numerous air pressure sensors and synthetic jet actuators in order to reduce aerodynamic drag. These sensors/actuators are simple agents with limited computing capabilities. They are embedded in the wing, laid out on a regular grid and constitute a network that uses both wired and wireless communication. This system is implemented hierarchically and is not centered around a single sink. Actuators react locally to changes in air pressure using a fast control loop, while also centralizing data for further adjustments using a slower loop.

Similarly, agents in our model work independently and only interact when needed, for instance to confirm their observations with neighbors or to propagate information. This form of "disconnected network" presents multiple benefits, especially in terms of communication overhead and energy consumption (see [1] for details). In this model, agents need to take proactive steps to enable communication, like moving closer to a neighbor, aiming an orientable antenna, waking up from power-saving modes, etc. By doing so, they form temporary groups that perform joint computational steps. We refer to such steps as agents *attending a meeting*. The behavior of an agent consists of an alternation of local tasks and group interactions at meetings.

978-0-7695-4773-2/12 $26.00 © 2012 IEEE
DOI 10.1109/NCA.2012.48

When a mission is implemented, agents are connected through a partnership graph. It should be emphasized that even though this graph is a graph of agents, it is not a connectivity graph in the usual sense. In particular, the fact that two (or more) agents are linked through a partnership does not mean that they share a communication link. It only means that these agents interact in the distributed computations of the system by attending joint meetings.

For these interactions to take place, partnership graphs are annotated with meeting schedules. They specify the points in time at which meetings take place. Every partnership—every edge in a partnership graph—is associated with an infinite sequence of meeting times, allowing agents to interact in a loosely choreographed way. (In this paper, we focus on periodic schedules for simplicity.) As a result, our framework is markedly more synchronous than many other models. However, the necessary synchronization is not an issue in practice since repeated interactions allow agents to apply clock-synchronization protocols efficiently.

During the course of its lifetime, the network will suffer from intermittent failures (e.g., agents miss their meetings) and permanent failures (e.g., agents run out of power or are destroyed by external events). The objective of this paper is to devise simple steps that agents can take to mitigate the effects of such failures on the mission.

III. Proposed Algorithms for Robustness

In the context of this paper, agents are (static or mobile) sensor nodes. As failures occur, these nodes need to reconfigure the partnership graph of the mission without central coordination and within the constraints imposed by their locations, communication ranges, motion capabilities, etc.

The solution we propose is based on simple algorithms, which nodes execute during meetings in order to react to perceived damage to the system. Complex transformations of the partnership graph could deliver better repair but are also difficult to implement without a global knowledge that the scale of the systems under consideration may make impossible to achieve. Furthermore, complex transformations also make the system more fragile and less able to sustain further failures. In particular, major changes to the partnership graph—in its topology or its schedule annotations—need to be propagated to all nodes involved before the next failure.

In this paper, we strive to keep dynamic transformations as simple as possible so they remain feasible by basic sensor nodes and do not leave the system in a transitory fragile state while changes are propagated. The transformations we propose here have very limited needs for propagation: Nodes unaware of a previous transformation can still react to further damage to the system as if the previous transformation never occurred.

Our algorithms are based on the notion of *crossing*, by which we mean that a node switches roles with a (possibly missing) neighboring node. If nodes are mobile, this could

```
atMeetingWith(p)
  cross
```

(a) *cross-always*

```
atMeetingWith(p)
  if p is missing
    cross
  else
    bounce
```

(b) *cross-bounce*

Figure 1. Two variants of a *crossing* protocol.

be achieved by actually "crossing" a physical boundary into another node's territory. Static nodes can achieve virtual crossings in a variety of ways: by extending communication ranges, switching to a different communication channel, reorienting a directional antenna, etc. In some cases, crossings may be partial in the sense that a node is not able to fully replace its missing partner. For instance, the sensing range of a node may be impossible to extend or may only be extended at the cost of less accurate measurements.

Using crossing as an elementary operation, nodes can apply different distributed algorithms in response to a failure. These algorithms may require different capabilities on the parts of the nodes and may have different performances in terms of repair quality. In order to explore the potential of crossing-based repair strategies, this paper focuses attention on the two algorithms in Fig. 1.

In variant (a), nodes systematically cross at meetings: When two nodes attend a meeting they switch roles and if one node is missing, the other takes its place. In variant (b) of the crossing algorithms, a node crosses at a meeting only if its partner is missing; otherwise, it "*bounces*" off the other node. The idea is that, if nodes are able to interact, they know their partners are active and there is no need for repair.

As an illustration, consider the case of a tiny network on a 3×3 grid: The center node has four neighbors (N, W, S and E), edge nodes have three neighbors and corner nodes have two neighbors; each node participates to pairwise meetings at regular intervals. The schedule used in this example has nodes visiting their neighbors in clockwise ($NESW$) or counterclockwise ($NWSE$) fashion, each node using a direction opposite of that of its neighbors.

Fig. 2 shows the behavior of the nine nodes in a perfectly healthy network. Nodes are represented as large dots and the series of their meetings form a trajectory—possibly virtual, as in the case of static sensors—represented as a line. The *cross-always* variant of the algorithm results in three trajectories or equal length, each with three nodes, while the *cross-bounce* variant has nine trajectories, each with one node. In general, on an $n\times m$ network, *cross-bounce* results in $n \cdot m$ individual trajectories and *cross-always* in g trajectories of equal length, where $g = \mathsf{gcd}(n, m)$. This is because in *cross-always*, nodes only bounce off the edges

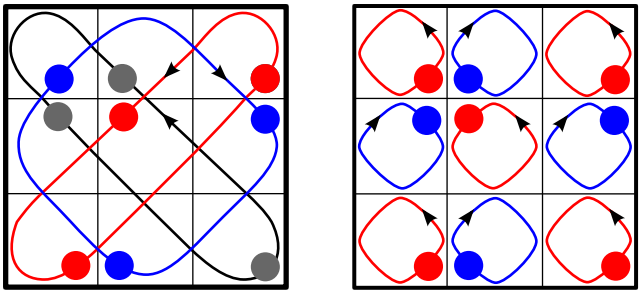

Figure 2. Node behavior under the *cross-always* (left) and *cross-bounce* (right) protocols.

of the network and thus travel on longer trajectories, while in the other variant, they bounce off each other and cannot escape their local paths.

To demonstrate the various healing properties of the algorithms, we examine the following failure scenario: The central node is permanently disabled and stops attending all of its meetings. This node was interacting in pairwise meetings with four neighbors, each of which becoming aware of the failure independently and at a different time.

Fig. 3(a) shows the behavior, after the failure of the center node, of a network in which all the nodes follow the *cross-always* variant of the protocol. In comparison to Fig. 2, one of the three trajectories now contains two nodes only. This results in the individual tasks of the nine original nodes being performed at rates that lay between 75% and 100% of what they were before the failure. In some sense, a "perfect repair" would have all nine tasks covered by the eight nodes at a $^8/_9 \approx 89\%$ rate. We use this idea of a perfect repair to evaluate and compare repair strategies in the next section.

In Fig. 3(b), the surviving nodes apply the *cross-bounce* variant of the protocol. Six nodes end up participating in the repair: The missing node's four neighbors and two of the corner nodes. The trajectories are fairly complex because nodes continue to bounce off each other. In particular, neighbors of the missing nodes can bounce off a node that has crossed and is playing the part of the failed node. Conversely, two of the corner nodes end up crossing because, even though their neighbors have not failed, they sometimes sacrifice their own meetings to participate in the repair.

It is worth observing that, although the trajectories are markedly different, the resulting task coverage is the same for *cross-always* and *cross-bounce*. Indeed, we use the *cross-always* variant mainly as a tool in the study of the *cross-bounce* variant. *Cross-always* has no real benefit over *cross-bounce* and has prohibitive costs if crossing is implemented in a way that requires a lot of resources. However, it also has simpler meeting trajectories that can be used to predict the performance of the *cross-bounce* variant.

Another important point to note is the fact that some crossings occur as a result of healthy nodes missing their own meetings because they are busy doing repair work. This can produce a form of domino effect: A node that steps in for a missing node is later perceived as having failed by its other neighbors, which triggers their stepping in and their being perceived as failed by their neighbors, and so forth. Although it has the potential of spreading the repair effort among all nodes, this phenomenon can be problematic. Firstly, however they are implemented, "crossings" likely have a non-negligible cost on the system (increase power usage, loss in accuracy, etc.). It is therefore desirable to limit their numbers and try to achieve good quality repairs with as few crossings as possible. Secondly, this domino effect makes nodes step in for their neighbors' neighbors, their neighbors' neighbors' neighbors, etc. This might be acceptable if crossings are performed through physical motion, but it can be impossible to achieve with static nodes. For instance, a node may be able to increase its transmission power to contact a neighbor's neighbor but not a neighbor's neighbor's neighbor.

To alleviate this problem, the *cross-bounce* protocol can be extended into a richer (but still simple) variant that uses limits and markers. Limits prevent nodes from crossing beyond a certain distance in the partnership graph; markers are used by a node to inform its neighbors that it is temporarily unavailable because it is performing the work of another node. The idea is to have nodes bounce off a marker when their partner is missing. This way, markers introduce a loose form of coordination that the raw version of the crossing protocol lacks. Since nodes may fail while working on behalf of other nodes, "fading" markers that spontaneously disappear over time can be used. This way, markers left by a failed node will not permanently impair future repairs. How markers are actually implemented is dependent on the type of nodes and is beyond the scope of this paper. There are two predominant strategies for the realization of markers: Use actual physical markers by acting on the environment (e.g., a node detects wireless transmissions from a neighbor to the neighbor's neighbors and infer from it a marker) or rely on virtual markers as an abstraction for distributed agreements among nodes that an edge of the partnership graph is "marked".

Limits and markers introduce tradeoffs between repair quality and cost: With more crossing or with nodes "travelling" farther from home, more nodes can be involved in the repair, thus improving repair quality at the cost of more crossings and more power usage. We have run experiments to evaluate these tradeoffs by measuring the impact of limits and markers. From lack of space, these are omitted in this short paper.

So far, the discussion has focused on permanent node failures. However, simple crossing and bouncing rules can also be defined to deal with recoveries from transient failures. The consequence of transient failures, either missed meetings due to communication errors or nodes resuming

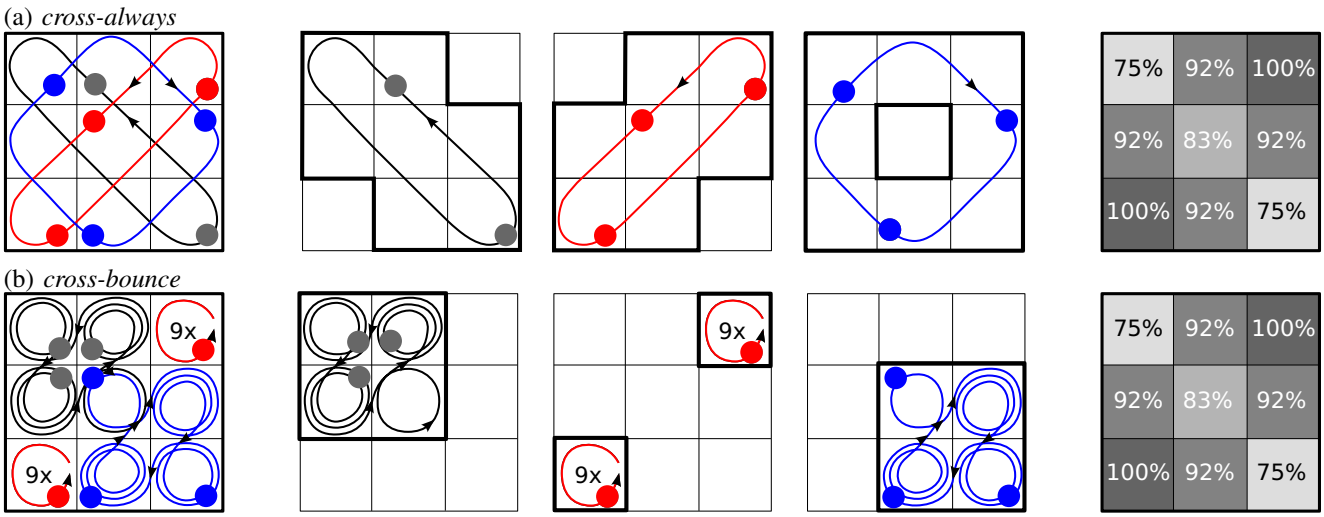

Figure 3. Single failure handled by 8 agents running a *cross-always* or *cross-bounce* protocol.

execution after temporary failure, is that a node may mistakenly cross and assume the role of a healthy node. Fortunately, such a situation can be quickly detected and resolved when both nodes—the healthy node and its replacement—run into each other at their next meeting. The property of our model that enables such a quick recovery is also the one that makes crossing possible in the first place, namely the simple, deterministic meeting schedules known to all agents.

As the meeting trajectories of Fig. 3 demonstrate, crossing protocols can result in fairly complex trajectories and different nodes can end up with different types of trajectories with different cycle lengths. This makes an analytical study of the protocols quite challenging. In order to guide us in such a study, we have implemented a simulator of the disconnected networks model. We have used this simulator to carry experiments on larger networks (hundreds to thousands of nodes) to measure the quality of network repair under various failure scenarios, such as a uniform distribution to represent independent node failures or a randomly shaped "hole" of failed nodes to represent the impact of an environmental event. In particular, we have studied the conditions under which enough nodes can be involved in a repair so that high failure rates can be handled with minimum crossing and travelling.

IV. Evaluation

As the meeting trajectories of Fig. 3 demonstrate, crossing protocols can result in fairly complex trajectories and different nodes can end up with different types of trajectories with different cycle lengths. Indeed, there are examples of systems with a few hundred nodes in which some trajectories are over several tens of thousands steps. This makes an analytical study of the protocols quite challenging.

In order to guide us in such a study, we have implemented a simulator of the disconnected networks model. We have begun to implement experiments that aim to measure the effectiveness of the proposed algorithms on large networks under various failure scenarios. In particular, we are especially interested in two measures: the average number of crossings per step (cost) and the quality of task coverage, including sensing (benefit). These can be used to study tradeoffs in many variants of the crossing algorithms.

As part of our evaluation, we are implementing different models of node failure, including uniform distribution to represent independent node failures and random shapes to represent agent loss due to external events. Preliminary results show interesting properties of different crossing variants, both expected and surprising. Due to space limitations, a detailed discussion is be presented here.

Our simulator can record relevant data during a simulation and also produces color animations that can be used to get a better intuition of the overall behavior of an algorithm on a large grid of nodes. As an illustration, Fig. 4 shows the first few frames of an animation of the *cross-bounce* algorithm after a subset of nodes in the center of the grid have been permanently damaged. It shows how nodes step in for missing nodes and the fast repair that involves many nodes at the cost of many crossing. A variant of the algorithm that limits node participation using travel distance limits and/or fading markers would produce a noticeably different animation (see the discussion above on domino effect, distance limits and markers).

V. Related Work

The need to rethink the suitability of traditional networks has been observed in the context of sensor networks [3]. In this paper, we consider networks of nodes that form

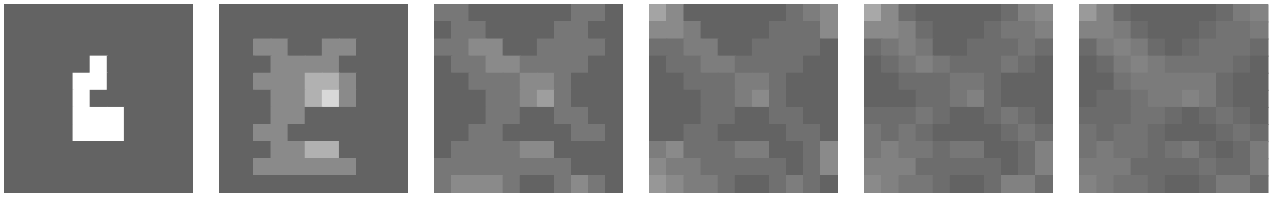

Figure 4. Changes in sensing rate after a permanent failure of 10 agents on a 11×11 grid and subsequent repair operation.

cooperating systems despite the lack of a standard communication infrastructure. The idea of such *disconnected networks* has been proposed and used in various contexts, differentiated by assumptions placed on how opportunities to interact occur, from stochastic properties [4] to weak fairness constraints [5] to full determinism [6], [7]. Our partnership-based model allows simpler nodes to achieve system transformations potentially more robust than those previously studied [8].

The emergence of mobility and wireless ad hoc networks has led to variations and extensions of traditional distributed computing models and algorithms [9]–[11]. In our approach, agents use these algorithms to implement joint operations at meetings, such as consensus or leader election with unreliable (wireless, acoustic) communication.

In the context of sensor networks, methods that address reliability typically utilize broader coordination among nodes while solving complex optimization problems, and assume deployment of additional sensors, e.g., [12]. This paper considers a network of very simple nodes, incapable of complex computation, deployed on a massive scale with no global repair mechanism. Furthermore, our approach places minimal requirements on the programming capabilities of the nodes in an attempt to remain compatible with most systems and languages.

VI. Conclusions and Future Work

In this paper, we define simple mechanisms that sensor nodes can use to mitigate the effects of failures in large-scale networks. The proposed algorithms rely on local interactions and take into account the limited capabilities of typical sensor nodes. This work is part of a larger effort to study the benefits (and limitations) of local algorithms in disconnected networks. In particular we aim to explore new ways to achieve deployment, sensing, computational tasks, adaptation and repair in large-scale networks of agents with tight resource constraints. We believe that better robustness and scalability can be achieved through deliberate relinquishment of global, all-connected, mechanisms in favor of local interactions with immediate neighbors in a disconnected network.

References

[1] M. Charpentier, R. Bartoš, and Y. Li, "Interaction patterns for resilient intermittently-connected static sensor networks," in *Proc. of IEEE Conference on Military Communications (MILCOM'10)*, San Jose, CA, October 2010, pp. 1004–1009.

[2] K. Bur, P. Omiyi, and Y. Yang, "Wireless sensor and actuator networks: Enabling the nervous system of the active aircraft," *Communications Magazine, IEEE*, vol. 48, no. 7, pp. 118 – 125, july 2010.

[3] D. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle, and J. Zhao, "Towards a sensor network architecture: Lowering the waistline," in *HotOS X: Tenth Workshop on Hot Topics in Operating Systems*, Jun. 2005.

[4] M. Grossglauser and D. Tse, "Mobility increases the capacity of adhoc wireless networks," *IEEE/ACM Transactions on Networking*, vol. 10, no. 4, pp. 477–486, Aug. 2002.

[5] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, and R. Peralta, "Computation in networks of passively mobile finite-state sensors," *Distributed Computing*, vol. 18, no. 4, pp. 235–253, Mar. 2006.

[6] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, "Delay-tolerant networking: an approach to interplanetary Internet," *IEEE Comm. Mag.*, vol. 41, no. 6, pp. 128 – 136, Jun. 2003.

[7] X. Zhang, J. Kurose, B. Levine, D. Towsley, and H. Zhang, "Study of a bus-based disruption tolerant network: Mobility modeling and impact on routing," in *13th ACM International Conference on Mobile Computing and Networking (MOBICOM)*, Montreal, Canada, 2007.

[8] G. Wang, G. Cao, T. L. Porta, and W. Zhang, "Sensor relocation in mobile sensor networks," in *IEEE INFOCOM*, 2005, pp. 1101–1112.

[9] S. Dolev, "Self-stabilizing and self-orgenizing mobile networks," in *DIAL M-POMC '08: Proceedings of the fifth international workshop on Foundations of mobile computing*, 2008, pp. 25–26.

[10] T. P. Hayes, J. Saia, and A. Trehan, "The forgiving graph: a distributed data structure for low stretch under adversarial attack," in *PODC '09: Proceedings of the 28th ACM symposium on Principles of distributed computing*, 2009, pp. 121–130.

[11] W. Wu, J. Cao, J. Yang, and M. Raynal, "Design and performance evaluation of efficient consensus protocols for mobile ad hoc networks," *IEEE Trans. Comput.*, vol. 56, no. 8, pp. 1055–1070, 2007.

[12] J. L. Bredin, E. D. Demaine, M. T. Hajiaghayi, and D. Rus, "Deploying sensor networks with guaranteed fault tolerance," *IEEE/ACM Trans. Netw.*, vol. 18, pp. 216–228, February 2010.