Fault Tolerance of Embryonic Algorithms in Mobile Networks

David Lowe[1], Amir Mujkanovic[1], Daniele Miorandi[2], and Lidia Yamamoto[3]

[1] Centre for Real-Time Information Networks
University of Technology Sydney, Australia
david.lowe@uts.edu.au, Amir.Mujkanovic@student.uts.edu.au
[2] CREATE-NET, v. alla Cascata 56/D, 38123, Povo, Trento, IT
daniele.miorandi@create-net.org
[3] Computer Science Department, University of Basel, Switzerland
Lidia.Yamamoto@unibas.ch

Abstract. In previous work the authors have described an approach for building distributed self–healing systems – referred to as EmbryoWare – that, in analogy to Embryonics in hardware, is inspired by cellular development and differentiation processes. The approach uses "artificial stem cells" that autonomously differentiate into the node types needed to obtain the desired system–level behaviour. Each node has a genome that contains the full service specification, as well as rules for the differentiation process. This approach has inherent self-healing behaviours that naturally give rise to fault tolerance. Previous evaluations of this fault tolerance have however focused on individual node failures. A more systemic fault modality arises when the nodes become mobile, leading to regular changes in the network topology and hence the potential introduction of local node type faults. In this paper we evaluate the extent to which the existing fault tolerance copes with the class of faults arising from node mobility and associated network topology changes. We present simulation results that demonstrate a significant relationship between network stability, node speed, and node sensing rates.

1 Introduction

In this paper, we consider the issue of fault-tolerance in self–healing distributed networks that incorporate mobile devices and hence rapidly changing network topologies. Inspired by related work on Embryonics [1, 2], in our earlier work [3] we proposed *EmbryoWare*, an "embryonic software" architecture for robust and self-healing distributed systems. Like Embryonics, the EmbryoWare approach is based on the assumption that each node in the system contains a *genome* that includes a complete specification of the service to be performed, as well as a set of differentiation rules meant to ensure that each node differentiates into the node type required to provide required overall system–level behaviour. A particular feature of both Embryonics and EmbryoWare is that there is no

distinction between the fault[4] handling behavior and the normal behavior of a node. The ability of a node to restore from a faulty to a normal state is a side-effect of the system's normal process of differentiating into the locally correct node type. Therefore, no special fault-handling routines are needed, which can make the system potentially more robust to unforeseen disruptions.

In [3] we examined the general behaviour and performance of the EmbryoWare approach and demonstrated its validity as well as its inherent robustness and self–healing ability. That previous work however focused on individual node failures with a uniform probability distribution of failures occurring in any node. There does exists the likelihood of more complex patterns of node failure. One of the more significant of these occurs when we have mobile nodes, leading to regular changes in the network topology. When the topology changes the local neighbourhood for nodes is affected. Given that nodes differentiate into different types based, in part, on the sensed information from nodes in their local neighbourhood, when this neighbourhood changes it can mean that the node types are no longer correct. This can be interpreted as the introduction of faults into the system. An example of this situation would be an ad hoc network of mobile devices (such as cell phones) that form a distributed processing network. As devices move, they establish and then lose temporary connections, and hence the network topology is constantly changing. This has implications for ensuring the validity of the system–level functionalities – particularly where the correct behaviour of each node is dependent upon the behaviours in its neighbourhood.

In this paper we evaluate the fault–tolerance behaviour of EmbryoWare under mobility, by measuring the extent to which the patterns in EmbryoWare can be maintained in a valid state in spite of mobility. In particular, we are interested in the relationships between the rate of fault generation (which will correspond to the speed of the nodes and hence the rate of change in the network topology) and those factors that affect the rate at which faults are addressed. In essence we are considering how quickly the nodes in an embryonic system can re-differentiate to ensure that the individual nodes are in a valid state.

In section 2 we discuss the background to our approach and related work. Then in section 3 we provide a brief overview of the basic EmbryoWare architecture and the changes we have made to incorporate node mobility into our simulations. We then describe our analysis approach and results in section 4. Finally, in section 5 we describe our conclusions and future work.

2 Background

The motivation for our work comes from the increasing utilisation of distributed services, i.e. services whose outcomes depend on the interaction of different components possibly running on different processors. Distributed services typically

[4] We refer to a fault as any circumstance in which a node is not operating in a steady state but rather a state in which subsequent sensing is likely to lead to a differentiation of the node type. This should be distinguished from a node failure, where the node has failed to operate correctly due to some other operational reason.

require complex design with regard to the distribution and coordination of the system components. They are also prone to errors related to possible faults in one (or more) of the nodes where the components execute. This is particularly significant for applications that reside on open, uncontrolled, rapidly evolving and large–scale environments, where the resources used for providing the service may not be on dedicated servers (as the case in many grid or cloud computing applications) but rather utilise spare resources, such as those present in user's desktops or even mobile devices. (Examples of such scenarios are the various projects making use of the BOINC or similar platforms[5].)

Other examples of distributed applications where each node takes on specific functionality include: peer-to-peer file sharing; distributed databases and network file systems; distributed simulation engines and multiplayer games; pervasive computing [4] and amorphous computing [5]. With all of these applications there is a clear need to employ mechanisms that enhance robustness and reliability, ensuring the system's ability to detect faults and recover automatically, restoring system–level functionalities in the shortest possible time.

In this work, we deal with problems arising when the topology changes due to nodes mobility. While as of today the vast majority of distributed services are meant to run over static nodes, the increasing penetration of powerful mobile devices (smartphones) has the potential of boosting the adoption of similar approaches in the mobile computing field. Even when the devices themselves are not mobile there still exists the potential for changes to the network topology due to approaches such as intelligent routing. We report the following example of applications, which help in better positioning our work.

Example 1 Wireless Grid Computing: *One example of the kind of applications our framework applies to is the so–called wireless grid computing [6–8]. This applies the same principles underpinning grid computing research to mobile phones. Sharing the load for performing heavyweight computational tasks across a plurality of devices can provide advantages in terms of completion time and load balancing. The possibility that the network topology can change dynamically introduces an additional level of complexity with respect to grid computing scenarios, due to the need to ensure that tasks will get completed even in the presence of disconnections.*

Example 2 Distributed Sensing Platforms: *Current state-of-the-art smartphones are sensor–rich. They typically include at least a camera (video and image sensor), a microphone (audio sensor) and short–range communication capabilities (such as Bluetooth and WiFi). Smartphones carried around by users could therefore be used as a distributed wireless sensing platform [9, 10]. Such a platform could be used to gather environmental information. An example is the distributed search engine considered in [11].*

Example 3 Mobile Data Sharing: *as smartphones are commonly equipped with some form of short–range wireless communications, they could be used to*

[5] `http://boinc.berkeley.edu/`

exchange data and content in a peer–to–peer fashion [12–14]. Going beyond pure flooding–based strategies (à la Gnutella) requires the introduction of a distributed indexing/caching services, which should be able to ensure some system–level performance (related, e.g., to the ability of locating and retrieving given content) even in the presence of device mobility.

We are particularly interested in distributed services whereby the desired system–level behaviour (or: system–level configuration, meaning the mapping of devices to 'types', where different node types carry out different behaviours) can be expressed in terms of *spatial constraints* between the nodes and their types. An example could be "A node of type `A` has to be no more than two hops away from a node of type `B`" or "Any node of type `C` shall have no more than two 3–hop neighbours of type `D`".

Robustness in distributed computing systems is a well–studied topic. Classical fault–tolerance techniques include the use of redundancy (letting multiple nodes perform the same job) and/or the definition of a set of rules triggering a system reconfiguration after a fault has been detected [15]. In many cases however it is not feasible to pre–engineer all possible failure patterns and the consequent self-healing actions to be taken for restoring global functionalities. In previous work by two of the authors [16], we considered the potential for using bottom-up approaches inspired by embryology to the automated creation and evolution of software. In these approaches, complexity emerges from interactions among simpler units. It was argued that this approach can also inherently introduce self–healing as one of the constituent properties without the need to introduce separate fault–handling behaviours. The ability of a node to restore from a faulty to a normal state is a side-effect of the system's normal process of differentiating into the locally correct node type.

3 EmbryoWare Architecture

EmbryoWare [3] applies concepts inspired by cellular development to the design of self–healing distributed software systems, leveraging off previous research conducted in the evolvable hardware domain. Such approaches, which gave rise to the *embryonics* research field [1, 2], are based on the use of "artificial stem cells" [17, 18], in the form of totipotent entities that can differentiate – based on sensing of the state of neighbouring cells – into any component needed to obtain the desired system–level behaviour. In general, we define an *embryonic system* as a system composed of networked entities that:

1. Are able to sense the state (or: type) expressed by neighbouring entities, i.e., those immediate neighbours with which direct communication is possible, or those entities for which information is provided by immediate neighbours;
2. Are able to differentiate their behaviour into a given type, depending on the type expressed by neighbouring entities and according to a set of well-defined rules;

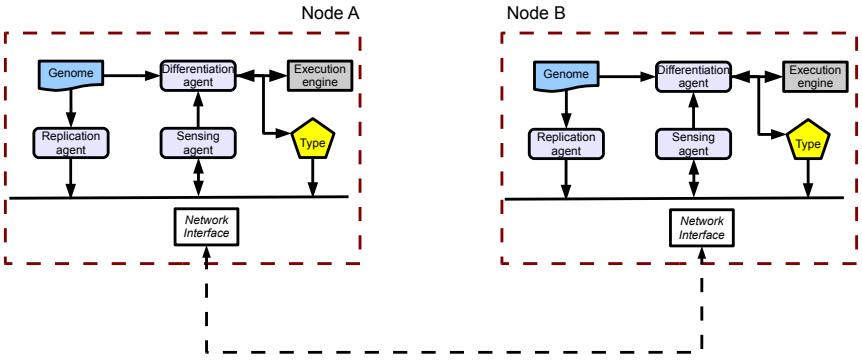

Fig. 1: EmbryoWare Architecture, showing two neighbouring nodes.

3. Are able to replicate to neighbouring entities (i) the definition of all types (ii) the set of differentiation rules.

Our specific architecture is shown in Figure 1 for the case of two neighbouring nodes. Nodes are organised in a network, and each node contains the following components:

- *Genome*: defines behaviour of the system as a whole, and determines the type to be expressed based on local context (i.e., neighbour cell types).
- *Sensing agent*: component that periodically communicates with neighbours regarding their current type. We consider in this work pull sensing, in which each node periodically polls its neighbours to inquire about their currently expressed type (as distinct from push sensing, in which each node 'pushes' information on its type to its neighbours).
- *Replication agent*: component that periodically polls the neighbours about the presence of a genome; if a genome is not present then the current genome is copied to the "empty" cell.
- *Differentiation agent*: component that periodically decides, based on the cell's current type and the knowledge about the types of the neighbouring cells, which functions should be performed by the node.

In our earlier work we discussed some possible design choices and considered the overall system performance – including the impact of network characteristics such as latency and dropped data packets [3]. However, whilst the algorithms themselves are independent on the network topology, we did not measure the impact of mobile nodes, and hence of a changing network topology. When the topology changes the local neighbourhood for nodes is affected, and this can mean that the node types are no longer correct. This can be interpreted as the introduction of faults into the system, and hence have significant implications for the ongoing validity of the system.

3.1 Case Study: Coordinated Data Sensing and Logging

The following example scenario will be used throughout this paper: a number of mobile wireless sensor devices are deployed over an area for the purpose of environmental monitoring. Each device collects sensor information from its surroundings, and the data collected must be logged within the local neighbourhood (to minimise longer range communication overheads). This means that each monitoring node should be within only a few hops of a logging node. In this case study we set this distance to two hops. When a monitoring node, through sensing its neighbourhood, discovers that it is not within two hops of a logger, then it will probabilistically differentiate into a logger. The differentiation behaviours are given in Algorithm 1 and the pattern that results is illustrated in Figure 2.

It is worth remarking that this specific example could be regarded as a clustering problem, in which cluster heads need to be at a maximum distance of four hops. Similar problems have received attention in the ad hoc network community, in particular related to the problem of computing the connected dominating set (CDS) [19]. This problem could be addressed in a traditional way by, e.g., first computing the CDS of the original network and then computing the CDS on the resultant overlay. However we believe that the EmbryoWare solution is much simpler, more compact, and able to handle faults in an intrinsic way. A comparison with existing cluster construction algorithms is a good topic for future work.

- *Stem* cell:
  with probability P_{TtoM}: $Type \leftarrow Monitor$
- *Monitor* cell:
  no 2-hop logger $\Rightarrow$ with probability P_{MtoL}: $Type \leftarrow Logger$
  with probability P_{MtoT}: $Type \leftarrow Stem$
- *Logger* cell:
  2-hop logger $\Rightarrow$ with probability P_{MLtoT}: $Type \leftarrow Stem$
  with probability P_{LtoT}: $Type \leftarrow Stem$

Algorithm 1: Differentiation behaviour for Genome for simple environment logging application.

In the subsequent sections, we will evaluate the impact on the system validity (i.e. the ability to return to a correct state from a state that include faults), in the case of a time–varying network topology due to nodes mobility, of different choices for the sensing period, i.e., the time elapsed between consecutive polls of a nodes neighbour. Furthermore, we will consider two options related to the timing of when a node becomes aware of a change in the topology. The baseline behaviour would be that nodes operate completely independently except for the periodic sensing. In our earlier work, with a fixed topology, this sensing only gave information on the current type of neighbouring nodes. With mobile nodes becoming a possibility, the sensing will give not only information on nodes

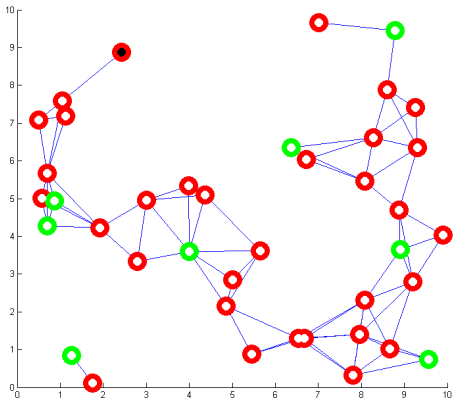

Fig. 2: Example differentiated node pattern: The red (darker) circles represent monitoring nodes and the green (lighter) circles are logging nodes. Nodes with a black centre are currently in an invalid state.

types but also node *connections* – i.e. the local neighbourhood topology. This means that if the topology changes due to node movement (or failure) then each node will only become aware of that, and respond to it through appropriate differentiation, after its next sensing operation. We refer to this sensing behavior as **connection unaware**. The alternative to this is if the node maintains a continuous awareness of its connections to other nodes (through relevant lower-level communications mechanisms, such as the loss or gain of a carrier signal and/or reception of appropriate beacon messages) then it could become aware of a changed topology much sooner than the next sensing cycle. In this situation it would be able to react much more quickly. We call this mode of operation **connection aware**. The implications of these two different sensing behaviours will be analysed in the following section.

4 Performance Evaluation Under Node Mobility

We now evaluate the impact of mobility on the fault-tolerance properties of the scenario described in Section 3.1. Initially, the overall system may be in a valid state (i.e. all monitoring nodes within 2 hops of a logger). However, as nodes move, and the topology changes, the validity of the system can be affected. Consider the cluster of monitoring (red) nodes around $(1, 7)$ in Figure 2. If these nodes were to move upwards then they would become isolated from the associated logging node at $(1, 5)$, and hence they would be in an fault state. This fault would persist until the nodes were able to sense the lack of a neighbourhood logger, and one of the nodes in this cluster differentiated into a logger.

A key performance characteristic to evaluate the system's self-healing ability is the percentage of time that the system is in an invalid state (i.e. a fault in the system is persisting). Two factors will affect this: the frequency with which faults arise, and the speed with which they are then corrected. The former should be predominantly related to the rate of change in the topology, and hence the speed at which the nodes are moving. The latter will be related to the speed with which the fault is detected, and hence the sensing behaviour.

Understanding the relationship between system validity, node speed, and sensing behaviour is important insofar as it allows us to appropriately tune the behaviours of the system. Sensing the state of neighbours (or, as discussed above, the existence of network connections) incurs both processing and bandwidth overheads. If we have a sensing behaviour that performs more rapid sensing than is necessary, then we are wasting resources.

To evaluate the extent to which each of these factors plays a role we extended the Matlab simulations from our previous work in order to incorporate node mobility. The basic algorithms for implementing the embryonic behaviours are outlined in [3]. These were modified in several ways. Firstly, the nodes have been made mobile. They have an initial location and a random (uniform distribution) velocity that only changes when the the node reaches the edge of the containing area (a lossless reflection). All nodes continuously move, with connections existing between nodes only when they are within a specified range of each other. The node network shown in Figure 2 was generated using $N = 40$ nodes initially randomly distributed in a $10m \times 10m$ area, with nodes being connected when they within $2m$ of each other.

We then undertook two main fault-tolerance evaluations – using each of the two primary sensing behaviours described above. To evaluate the fault-tolerance, we varied the maximum node velocity over the range $0...2m/s$, and the sensing period over the range $0.05...0.8secs$. For each pair of velocity and sensing period values, we ran 10 simulation passes, with each pass running for an initial period to allow node replication to occur, and then a $60sec$ evaluation period where we measured the proportion of time during which no fault was present in the network.

4.1 Connection aware versus connection unaware sensing

The first set of analyses were carried out for the two sensing behaviours described previously. Figure 3 graphs the overall system fault rate (i.e. the percentage of time for which the system contains at least one faulty node, i.e. a node is not within 2 hops of a logging node and hence needs to differentiate to return to a valid node type) against node speed and sensing period for the two cases discussed above (i.e. where the nodes do, and do not, retain awareness of the existence and loss of network connections).

As can be seen from these results, in both cases there is a noticeable, though expected, increase in the percentage time that the system contains at least faulty node as the node mobility increases. Of interest is that this increase is gradual and relatively linear, and there does not appear to be a point at which the ability

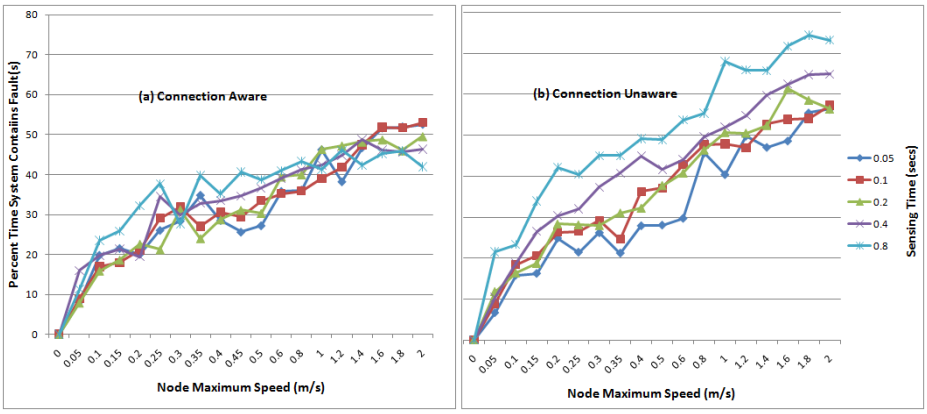

Fig. 3: Fault recovery: Results showing the percentage of time that the system contains at least one faulty node for varying node maximum speed and varying sensing times: (a) where the nodes retain awareness of the existence or failure of network connections; and (b) where the nodes do not monitor the state of the network connections. (Generated by Matlab files GenomeTester_v3j.m and GenomeTester_v3k.m)

of the system to recover collapses. This is an important observation insofar as the implications for varying node speeds that can be tolerated.

Somewhat more surprising is the result with regard to variations in the sensing period. In the "connection aware" case, variations in the sensing period appear to have only marginal effect on the fault recovery. This can be explained as follows: when a connection between two nodes is broken because of node movement, the direct neighbouring nodes will become aware of this immediately and any information that either node obtained from the other node is removed from its list of sensed data. This means that the node differentiation can then occur immediately, rather than needing to wait for the next sensing period. The details of the implementation of this are given in Algorithm 2. The only occasions when an immediate re-differentiation does not occur is where the directly impacted nodes are still in a valid state, and it is nodes further away in the neighbourhood that are the only ones that enter a faulty state. In this case the re-differentiation that corrects the fault must wait for a sensing cycle to occur. Overall, this particular behaviour leads to a more rapid response to changes in the network topology and a relative Independence of the sensing period, but does require that all nodes retain constant awareness of their connectivity to nearby nodes (often this would be available through the presence of a carrier signal) with the associated resource overheads that this implies.

In the "connection free" case, there is a slightly stronger relationship with the sensing period. As can be seen, as the sensing period gets longer, the percentage of time that the system contains faulty nodes increases. We can understand this relationship more clearly by looking not only at the time that the whole system is valid (i.e. no nodes at all that are in a faulty state), but at the average

```
for all i ∈ Nodes do
    if Node movement leaves region then
        reverse Node velocity
    update Node location
for all i, j ∈ Nodes do
    calculate distance(i, j)
    if distance(i, j)< commRange then
        connected(i, j)=true
for all i, j ∈ Nodes do
    if !connected(i, j) then
        delete Node i sensed data obtained from Node j
```

Algorithm 2: Node movement

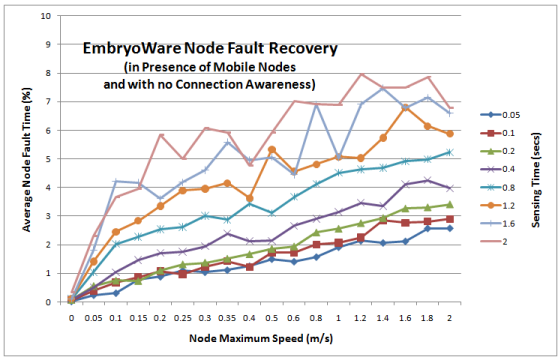

Fig. 4: Connection unaware sensing: Results showing the average percentage of time that a node is faulty for varying node maximum speed and varying sensing times, where the nodes do not monitor the state of the network connections. (Generated by Matlab file GenomeTester_v3k.m)

validity of each individual node. Figure 4 shows these results. As can be seen, there is a much more significant relationship to the sensing period. Several other observations arise from this data. Firstly, it appears that there is a baseline fault rate that even extremely rapid sensing cannot improve – for example, with the system configuration used in these simulations[6], at a node maximum speed of $0.25m/s$, it does not appear possible to reduce the average percentage of time that nodes are in a fault state below 1% irrespective of how quickly the sensing occurs. We believe that this is an artifact of the algorithmic sequencing in our simulation – though even if this is the case, similar behaviours would be likely to emerge in real-time code executing on live mobile devices.

A second observation arising from the data shown in Figure 4 is the increasing volatility of the average node fault rate as the sensing period increases. The processes being evaluated are inherently stochastic, both in terms of the speed and associated movement of the nodes (and hence the changes to network topology), and in terms of the node differentiation decisions. At low sensing periods the baseline fault rate (as discussed above) tends to dominate the behaviour.

[6] Relevant factors in the configuration are likely to be area size, number of nodes and hence node density, and the probabilities that affect the differentiation behaviours.

At slower sensing rates however the delay in returning to a valid state from a fault state appears to be significantly more variable. This may be an issue that needs to be taken into account with applications that cannot afford extended periods of unavailability of individual nodes – though it is worth acknowledging that embryonic systems are designed explicitly so that they do not rely on the behaviour, or indeed even availability, of individual nodes.

5 Conclusions and Further Work

In this paper we report performance measurements with regard to the fault tolerance of a distributed processing architecture, based on embryonic principles, where the nodes in the system are mobile. The node mobility inherently leads to constant changes in the network topology for the system, and hence changes in the local neighbourhood for individual nodes. This in turn can lead to those nodes being temporarily in a fault state. This fault state in inherently rectified by the self-healing differentiation processes in the nodes – but this process does take time.

We have evaluated the relationship between node speed, node sensing period, and fault recovery. Interestingly, we found that rather than reaching a "knee" in the performance curve where above a certain node speed the system performance collapsed and became unable to recover from the increasing number of faults, the relationship between node speed and fault recovery was relatively linear. This is likely to be an important finding in terms of dynamic adaptation of the sensing periods in the nodes in ensuring that the performance remained above a specified level.

We also have shown that the fault recovery performance becomes much less dependant upon the sensing period if nodes are able to continuously monitor the existence (or loss) of the network connections. This monitoring is unlikely to be feasible in systems involving, for example, sensor networks where the communication is intentionally very sporadic in order to minimise resource utilisation (i.e. most commonly power and/or bandwidth). However in other domains where the connection is maintained (or at least there is a constant carrier) this finding will be significant in that it indicates a much lower sensing rate, and hence lower processing and bandwidth overheads, will be tolerable.

One aspect that we have not considered, and which is a fruitful source for future investigation, is the possibility of replacing (or even supplementing) state sensing with pro-active state broadcasting. In this scenario when a node changes its state it would broadcast its changed state to its neighbour. This may circumvent the need for monitoring of the connection (as described in the previous paragraph) as a simpler way of making the performance less dependant on the sensing period. However this could also introduce excessive messages when mobility is high, and a compromise would have to be found.

Our measurements are performed over a particular case study: the logging scenario. Ideally, one would like to know the general fault-tolerance properties of the EmbryoWare approach. For this purpose, as a future work, it would be in-

teresting to evaluate several different cases, and see whether they share common fault handling patterns.

References

1. Ortega-Sanchez, C., Mange, D., Smith, S., Tyrrell, A.: Embryonics: a bio-inspired cellular architecture with fault-tolerant properties. Genetic Programming and Evolvable Machines **1**(3) (2000) 187–215
2. Tempesti, G., Mange, D., Stauffer, A.: Bio-inspired computing architectures: the *embryionics* approach. In: Proc. of IEEE CAMP. (2005)
3. Miorandi, D., Lowe, D., Yamamoto, L.: Embryonic models for self-healing distributed services. In: Proc. ICST Bionetics, Avignon, France (2009)
4. Saha, D., Mukherjee, A.: Pervasive computing: A paradigm for the 21st century. Computer **36**(3) (2003) 25–31
5. Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy, G., Thomas F. Knight, J., Nagpal, R., Rauch, E., Sussman, G.J., Weiss, R.: Amorphous computing. Communications of the ACM **43**(5) (2000) 74–82
6. McKnight, L.W., Howison, J., Bradner, S.: Wireless grids — distributed resource sharing by mobile, nomadic, and fixed devices. IEEE Internet Computing **8** (2004)
7. Ahuja, S.P., Myers, J.R.: A survey on wireless grid computing. J. Supercomput. **37** (2006) 321
8. Palmer, N., Kemp, R., Kielmann, T., Bal, H.: Ibis for mobility: solving challenges of mobile computing using grid techniques. In: Proc. of HotMobile. (2009) 1–6
9. Akyildiz, I.F., Melodia, T., , Chowdhury, K.R.: A survey on wireless multimedia sensor networks. Computer Networks (2006) 921960
10. Campbell, A., Eisenman, S., Lane, N., Miluzzo, E., Peterson, R., Lu, H., Zheng, X., Musolesi, M., Fodor, K., Ahn, G.S.: The rise of people-centric sensing. IEEE Internet Computing **12** (July/August 2008) 1–21
11. Yan, T., Ganesan, D., Manmatha, R.: Distributed image search in camera sensor networks. In: Proc. of ACM SenSys. (2008)
12. Ding, G., Bhargava, B.: Peer-to-peer file-sharing over mobile ad hoc networks. In: Proc. of IEEE PerCom Workshops. (2004) 104–108
13. Marossy, K., Csucs, G., Bakos, B., Farkas, L., Nurminen, J.: Peer-to-peer content sharing in wireless networks. In: Proc. of IEEE PIMRC. Volume 1. (2004) 109–114
14. Kelényi, I., Csúcs, G., Forstner, B., Charaf, H.: Peer-to-peer file sharing for mobile devices. In Fitzek, F.H.P., Reichert, F., eds.: Mobile Phone Programming — Application to Wireless Networking. Springer Netherlands (2007) 311324
15. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed systems: concepts and design. Addison-Wesley Longman (2005)
16. Miorandi, D., Yamamoto, L., De Pellegrini, F.: A survey of evolutionary and embryogenic approaches to autonomic networking. Computer Networks (2009) In press, doi:10.1016/j.comnet.2009.08.021.
17. Mange, D., Stauffer, A., Tempesti, G.: Embryonics: a microscopic view of the molecular architecture. In: Proc. of ICES, Lausanne, CH (1998) 185–195
18. Prodan, L., Tempesti, G., Mange, D., Stauffer, A.: Embryonics: artificial stem cells. In: Proc. of ALife VIII. (2002) 101–105
19. Wan, P.J., Alzoubi, K.M., Frieder, O.: Distributed construction of connected dominating set in wireless ad hoc networks. Mobile Networks and Applications **9**(2) (Apr. 2004) 141–149