An Algorithm for Reconnecting Wireless Sensor Network Partitions

Gianluca Dini, Marco Pelagatti, and Ida Maria Savino

Dept. of Ingegneria della Informazione
University of Pisa
Via Diotisalvi 2, 56100 Pisa, Italy
{g.dini,m.pelagatti,i.savino}@iet.unipi.it

Abstract. In a Wireless Sensor Network, sensor nodes may fail for several reasons and the network may split into two or more disconnected partitions. This may deteriorate or even nullify the usefulness and effectiveness of the network. Therefore, repairing partitions is a priority. In this paper we present a method to repair network partitions by using mobile nodes. By reasoning upon the degree of connectivity with neighbours, a mobile node finds the proper position where to stop in order to re-establish connectivity. Factors influencing the method performance are singled out and criteria for their selection are discussed. Simulations show that the proposed method is effective and efficient notwithstanding packet loss.

1 Introduction

Networked Embedded Systems play an increasingly important role and affect many aspects of our lives. New applications are being developed in areas such as health-care, industrial automation, smart building and rescue operations. The European Integrated Project "Reconfigurable Ubiquitous Networked Embedded Systems" (RUNES) [1] brought together 21 industrial and academic partners with the aim of enabling the creation of large scale, distributed, heterogeneous networked embedded systems that inter-operate and adapt to their environments.

To illustrate the potential of the networked embedded systems, the project selected a disaster relief scenario, in which a fire occurs within a tunnel, much as happened in the Mont Blanc tunnel in 1999 [5]. The RUNES work in general and the disaster relief scenario in particular offer a number of interesting and challenging problems. In the rest of the paper we focus on the following.

A set of nodes with wireless communication capabilities are deployed inside the tunnel for monitoring purposes. As soon as an emergency situation occurs, for example an accident involving many cars, the nodes need to transmit data regarding the tunnel conditions to a base station responsible for tunnel control. In such a scenario, accurate and comprehensive information must be provided to the base station so that correct counter measures can be taken. It is of fundamental importance that the network would maintain connectivity, so that the flow of critical data to the base station is guaranteed. However, the network could be partitioned because of a malfunction of the nodes, caused by a fire, or because the presence of obstacles that deteriorate or even nullifies metrics of the Quality of Service. In such a critical situation, restoring network connectivity is a priority.

The problem of network partitioning in WSN is not entirely new even though so far has received limited attention [16]. Chong and Kumar raise the problem of partitions with a security focus [7]. So do Wood and Stankovic with respect to denial of service [17]. In [6], Cerpa and Estrin propose methods to self-configuring WSNs topologies. Although they mention the problem of network partitions as an important one, however, they leave such methods to future work. Finally, Shrivastava *et al.* propose a low overhead scheme to detect network partitioning, "cuts" in their parlance, but they do not propose any method to repair them [16].

With respect to Shrivastava *et al.*'s work, in this paper we focus on the complementary problem of restoring network connectivity. With reference to the tunnel scenario, we propose a method that uses autonomous mobile nodes. Once the base station determines the network partitioning, one or more mobile nodes are sent inside the tunnel. A mobile node is equipped with a radio transmitter-receiver so that it can communicate with the sensor nodes. Furthermore, it maintains connectivity with the base station through the wireless sensor network. By reasoning upon the degree of connectivity with neighbours, a mobile node navigates inside the tunnel until it reaches the optimal position to re-establish connectivity.

The paper is organized as follows. In Section 2 we state the system model. In Section 3 we define the problem. In Section 4 we present the algorithm the mobile nodes locally perform to restore the connectivity and we single out the main factors affecting the algorithm. In Section 5 we present a performance analysis based on simulations. Finally, in Section 6 we draw final conclusions.

2 System Model

According to the tunnel-disaster-scenario, the wireless sensor network is composed of a powerful base station and a set of low-end sensor nodes. Base station and sensor nodes have wireless capabilities and communicate through a wireless, multi-hop, ad-hoc network. We assume the resulting wireless network runs a routing algorithm that is able to cope with the failure of a "small" number of nodes by finding alternative routes [10,14,15]. However, in the case of a disaster, the number of failed nodes is "too large" and the network breaks into two or more disconnected partitions.

We assume the existence of a *Partition Detection System* (PDS), running on the base station and able to both detect the presence of network partitions and provide a rough estimation of their positions. More precisely, we assume the Shrivastava *et al.*'s partition detection system [16] that works as follows. The base station knows the position of a small subset of sensor nodes that are called the *sentinels*. Each sentinel communicates with the base station at regular

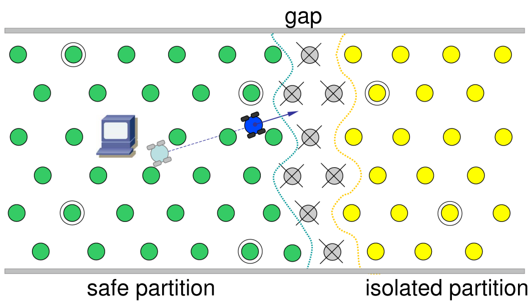


Fig. 1. The tunnel disaster-relief scenario. Double-circled nodes represent sentinels.

time intervals. Intuitively, the failure of the base station to communicate with a given sentinel is the proof that a partition containing that sentinel has formed (Figure 1). Furthermore, the sentinel position provides a rough estimation of the partition position.

Notice that even in structured environments such as a tunnel, the base station could not be able to define where exactly the failure occurs. Let us suppose that the base station knows the location of each node. So, the base station could broadcast a discovery message and then define the partition border on the basis of the not-responding nodes. Nevertheless, this solution is not suitable because too expensive in terms of time and communication overhead.

Our system includes mobile nodes (robots), that are used for repairing the network partitions. Upon detecting the presence of a partition and roughly estimating its position, the base station sends a mobile node to that position. The mobile node navigates inside the tunnel until it reaches the target position. Each mobile node is equipped with the same communication capabilities as sensor nodes so that it can communicate with sensor nodes in its neighbourhood and with the base station through the wireless sensor network. In this way, mobile nodes can reach positions that are far from the base station despite their limited radio range. We assume that the speed of a mobile node is such that its neighbourhood remains practically constant during a network round-trip time.

3 Problem Definition

Let us assume that the WSN splits in two partitions: a *safe partition*, containing the base station, and an *isolated partition*. The two partitions are separated by a *gap* of failed nodes (Figure 1). For brevity, we call *safe nodes* the nodes belonging to the safe partition and *isolated nodes* those belonging to the isolated one.

The PDS detects the network partitioning and knows that the inter-partition gap intersects the path leading to the not-responding sentinel, but the PDS is not able to exactly determine where the intersection actually occurs and how wide the inter-partition gap is. Hence, the mobile node has to determine itself the proper place where to stop according to the following conditions: 1) the mobile node is in contact with both the safe and the isolated partition; or, 2) the mobile node is in contact with the safe partition and any further movement makes the mobile node lose connectivity with that partition.

Condition 1 occurs when the safe and the isolated partition are so close to each other that a single mobile node is sufficient to reconnect them despite its limited communication range.

Condition 2 occurs when the inter-partition gap is too wide and a single mobile node is not sufficient to reconnect them. In this case the mobile node has to get as close as possible to the isolated partition while remaining always connected to the safe partition. This means the mobile node has to realise when it is about to lose connectivity with the safe partition and, consequently, stop before this event takes place.

4 Algorithm for Repairing Network Partitioning

As soon as the PDS has detected the network partitioning, the base station broadcasts a fresh number, called *epoch*, to identify the partitions. Safe nodes, that are connected with the base station, can receive the new epoch. In contrast, isolated nodes keep holding the old epoch. Notice that in real environments communication channels are not stable because of several factors including the distance between nodes, environment conditions, noise, and interference. So, a safe node could not be able to receive the new epoch.

As soon as the base station has distinguished the safe partition from the isolated one by broadcasting the epoch, it sends the mobile nodes towards the isolated partition. While navigating, each mobile node detects the partition boundary by monitoring its connectivity degree with the safe partition, i.e., the number of safe nodes it can communicate with. In fact, when the number of neighbours is below a certain threshold, it is likely that the mobile node is close to the partition boundary (Figure 2). However, a simple threshold-based approach may not be sufficient because of communication instability. In our model, we have a good communication link between a mobile node and a fixed node only if the probability for the former to receive a message is greater than a certain threshold

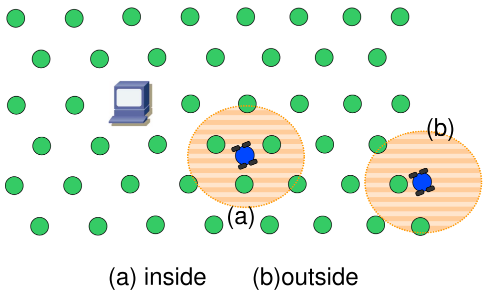


Fig. 2. Number of neighbour nodes vs. mobile node position

$\mathcal{P}_g$. So, the mobile node considers a fixed node as its neighbour if they have a good communication link.

During this phase, referred to as *Monitoring Phase*, the mobile node broadcasts an Hello message. If a fixed node replies with a Reply message containing the current epoch, the mobile node assumes that the fixed node certainly belongs to the safe partition. The mobile node counts how many Replys it receives from safe neighbours. As long as the mobile node is in the safe partition, the number of Replys it receives for each Hello does not suffer strong variations. So, if the mobile node detects a decrease in the number of received Reply messages, it could be about to lose connection with the safe partition.

It is worthwhile to notice that the mobile node may receive a Reply message carrying an old epoch even from a safe node. In fact, a safe node could fail to receive the new epoch because of packet loss. So, if the mobile node has not detected any decrease in its neighbourhood during the Monitoring Phase, it can ignore the message containing the old epoch.

If the mobile node detects a decrease in its neighbourhood during the Monitoring Phase, it enters into the *Verification Phase*. In this phase, the mobile node verifies whether it is about to lose connection with the safe partition by broadcasting a burst of Hello messages. If this is not the case, the mobile node returns into the Monitoring Phase. Otherwise, it verifies whether it has received a Reply message containing the old epoch. If it is the case, the mobile node assumes that it has reached an isolated node so that the partitions have been bridged. Otherwise, the mobile node assumes that the gap is too wide and any further movement makes him lose the connectivity with the safe partition.

When the mobile node reaches the position to bridge the partitions or, at least, reduce the partition gap, it may stop and behave as an ordinary, fixed sensor node, so participating to routing and sensing, if it has sensing capabilities, and cooperating with other mobile nodes for network repairing. Alternatively, the mobile node may carry fixed sensor nodes and deploy one of them in the final position. These alternative choices influence the mobile node complexity from an electro-mechanical standpoint.

The mobile nodes could be equipped with an hardware device for the self-localization. In case of inter-partition gap too wide, the mobile node could broadcast its own position to the other mobile nodes so that they can use this information to adjust their final position. Nevertheless, each mobile node has to perform the algorithm to verify its connectivity degree with the safe partition.

Furthermore, repairing the network is not the only form of cooperation required to mobile nodes. Mobile nodes have also to cooperate to avoid crashing into one another or into obstacles. Of course, this influences the actual path of the mobile node to reach the final position. It should be noted that the actual path the node takes is independent of the problem addressed in this paper, namely finding the proper positions where to place nodes (mobile or not) to repair network partitions. For this reason we shall not consider these issues any further. Interested readers can refer to [2,4].

4.1 Algorithm for Finding the Proper Position

The Monitoring and the Verification Phase are implemented by the Monitor and Verify functions, respectively.

```
 1: function Monitor(epoch)
 2: begin
 3: $\Delta T_H = 0; replySet = \emptyset;$ round=random number;
 4: repeat/* round */
 5:     wait($\Delta T_H$);
 6:     broadcast(⟨HELLO, epoch, round⟩);
 7:     setTimeOut($\Delta T_R$);
 8:     repeat
 9:         reply = receive();
10:         if (getRound(reply) == round) then
11:             replySet = replySet ∪ reply;
12:         end if
13:     until timeout strikes
14:     $n_R$ = size(replySet);
15:     round++;
16:     $\Delta T_H$ = newInterval($n_R$);
17: until $n_R > N_{PS}$
18: end
```

Fig. 3. The Monitor function

A conceptual implementation of the Monitor function is in Figure 3. The Monitor function is organized as a sequence of rounds that starts every ΔT_H seconds. The sequence terminates when the mobile node is in the gap and about to lose connection with the safe partition. In every round, the Monitor function broadcasts an HELLO packet (line 6) and receives the corresponding REPLY packets for ΔT_R seconds from fixed nodes (lines 7-9,13). An HELLO packet has two fields: (i) an *epoch* field that specifies the epoch known to the mobile node; and (ii) a *round* field that specifies the round (of that epoch) in which the mobile node has transmitted the packet. A REPLY packet has three fields: (i) an *identifier* field that specify the fixed node identifier; (ii) an *epoch* field that specifies the epoch known by the fixed node; (iii) a *round* field that specifies the round of the HELLO packet to which the REPLY packet replies. We say that a REPLY packet is a *valid* reply for a given HELLO packet if the former carries the same round and the same epoch as the latter.

The Monitor function counts the number n_R of valid REPLY packets coming from safe nodes (line 14). If n_R is greater than a given threshold N_{PS}, Monitor assumes that the mobile node is still in the safe partition and calculates the new value for the interval ΔT_H on the basis of the n_R itself (line 16). Notice that such a computation does not take into account ΔT_R, that is usually negligible with respect to ΔT_H. On the contrary, if $n_R < N_{PS}$, the Monitor function assumes that it is about to lose connectivity with the safe partition. So, the function ends and the mobile node calls the Verify function to ascertain whether it is actually in the gap between partitions.

```
 1: function Verify(epoch) $\rightarrow$ [FalseAlarm, IsolatedNodeFound, InTheGap ]
 2: begin
 3: replySet = $\emptyset$; round = random number; neighbours = 0
 4: while ((round $< B$)) do
 5:     broadcast($\langle$HELLO, epoch, round$\rangle$))
 6:     setTimeOut($\Delta T_R$);
 7:     repeat
 8:         reply = receive();
 9:         if ((getEpoch(reply)==epoch) and (getRound (reply)==round)) then
10:             replySet = replySet $\cup$ reply;
11:         end if
12:         if (size(select(replySet,getId(reply)))==$B_{min}$) then
13:             neighbours++;
14:         end if
15:     until timeout strikes
16:     if (neighbours $\geq N_C$) then
17:         return FalseAlarm;
18:     end if
19:     round++;
20: end while
21: for reply in replySet do
22:     if (getEpoch( reply ) $\neq$ epoch) then
23:         return IsolatedNodeFound;
24:     end if
25: end for
26: return InTheGap;
27: end
```

Fig. 4. The Verify function

The conceptual implementation of the Verify function is in Figure 4. The function estimates the connectivity degree of the mobile node with the safe partition, i.e. how many good links the mobile node has with safe nodes. The Verify function evaluates the neighbourhood by broadcasting a burst of B HELLO messages (lines 4–20). If a fixed safe node replies with at least B_{min} valid messages, it is considered a neighbour (lines 9–14). If the number of neighbours exceeds a given threshold N_C, the mobile node has still enough neighbours. So, the function returns FalseAlarm (lines 16–17) and the mobile node returns to the Monitor function.

If the number of neighbours is less than N_C, the function verifies whether it has received a REPLY message from an isolated node. So, if it has received a REPLY packet containing the old epoch, the function returns the value IsolatedNodeFound (lines 21–25). Otherwise, the function assumes that the mobile node is about to disconnect from the safe partition and returns the InTheGap value (lines 26).

4.2 Algorithm Parameters

In order the algorithm performs as expected, the following parameters should be properly chosen: the *Connectivity threshold* N_C, the *Phase-Switch threshold* N_{PS}, the *Signalling interval* ΔT_H, the *Response interval* ΔT_R, and the *Burst parameters* B and B_{min}.

Let us define R_g the maximum distance covered by good links. We assume most of neighbouring nodes lie into a circle with radius R_g, whereas nodes placed outside suffer link instability. Given the radio communication range R, the parameter R_g could be defined as $R_g = \mathcal{P}_g \times R$, or calculated via experiments. Furthermore, let us define $\overline{N}_{neigh}$ the expected number of neighbours when the mobile node is in the safe partition. Given the initial node distribution δ_0, $\overline{N}_{neigh}$ can be defined as $\overline{N}_{neigh} = \lfloor \delta_0 \pi R_g^2 \rfloor$ or via experiments. Note that the nodes are subjected to failure and the network density could get not uniform. More in detail, in some areas the density could be less than the initial value δ_0.

The Connectivity Threshold N_C
The *Connectivity threshold* N_C is the minimum number of neighbours in the safe partition the mobile node has to be in contact with during the Verification Phase.

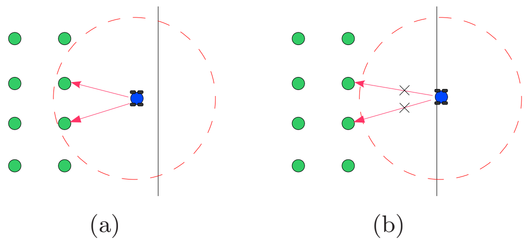


Fig. 5. A small N_C may result in disconnections

The Connectivity threshold N_C influences the algorithm performance because a too small value may cause disconnections from the safe partition. Let us consider the scenario depicted in Figure 5 where the mobile node is in contact with two neighbours. If $N_C = 1$ the mobile node keeps moving and loses the connectivity with the safe partition. Actually, the next time it checks for connections, it has already left the partition. For this reason, $N_C \geq 2$ is in general preferable. Nevertheless, a high N_C value could cause a premature stopping of the mobile node in the safe partition. So, N_C has to be below $\overline{N}_{neigh}/2$ that is the expected number of neighbours when the mobile node is crossing the border.

The Phase-Switch Threshold N_{PS}

The *Phase-Switch threshold* N_{PS} is the expected number of REPLYs the mobile node receives as it enters the gap between partitions. So, if the number of replies is lower than N_{PS}, the mobile node switches from the Monitoring Phase to the Verification phase. A high value of N_{PS} implies frequent switchings to the Verification Phase: many bursts are sent in order to verify the mobile node's connectivity, thus causing an excessive amount of messages. On the contrary, choosing a too low threshold reduces phase switchings and may cause disconnections from the safe partition if the Verification Phase is executed when the node has already crossed the border of the safe partition.

The mobile node uses this parameter to perform a first, rough detection of the partition boundary. When the mobile node crosses the border, half of the communication range lies outside the safe partition. Hence a reasonable value is $N_{PS} = \overline{N}_{neigh}/2$. Furthermore, during the Monitoring Phase, the mobile node could receive messages from poor link nodes. In order to avoid disconnections from the safe partition, the Phase-Switch threshold has to be greater than the Connectivity one. That is, $N_{PS} > N_C$.

The Signalling Interval ΔT_H

The *Signalling interval* ΔT_H is the time between two consecutive HELLO messages broadcast during the Monitoring Phase. It affects both the probability of disconnecting from the safe partition and the total number of messages exchanged during the repairing process. In order to stop in time, a mobile node needs to sample the number of its neighbours with a high frequency. Hence, a short ΔT_H is required. However, using a small interval implies an excessive amount of messages. For this reason, we opted for an adaptive calculation of ΔT_H in order to use shorter intervals only in the most critical region (i.e., when the mobile node is entering the gap between partitions).

As the mobile node gets near the safe partition boundary, the number of fixed nodes around it decreases because only a portion of its communication range lies within the safe partition. The signalling interval is calculated taking into account the worst case, when the partition border is perpendicular to the direction $\hat{v}$ of the mobile node (Figure 6).

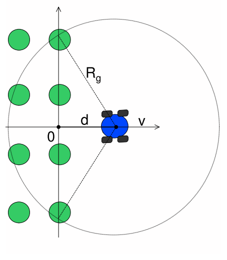


Fig. 6. Area containing neighbours

$$n_{cur}(d) = \begin{cases} \left\lfloor \frac{\overline{N}_{neigh}}{\pi} \left(\arccos \frac{d}{R_g} - \frac{d}{R_g} \sqrt{\frac{R_g^2 - d^2}{R_g^2}} \right) \right\rfloor & |d| \le R_g \\ \overline{N}_{neigh} & d \le -R_g \end{cases} \tag{1}$$

Let $n_{cur}(d)$ be the number of nodes within the R_g range when the mobile node is at a distance d from the safe partition border. A negative distance from the border means that the mobile node is still in the safe partition and has not crossed the border. With reference to Figure 6, $n_{cur}(d)$ is given by the Equation 1.

Since the number of neighbours assumes only discrete values in the range $[0, \overline{N}_{neigh}]$, the mobile node can construct a translation table containing the pairs $\langle n, \tilde{d}(n) \rangle$, where n is the number of neighbours and $\tilde{d}(n)$ is the maximum value so that $n_{cur}(\tilde{d}(n)) = n$. By using the translation table, the mobile node can estimate the maximum distance that can be covered without losing connectivity with the safe partition. More in detail, given the number of Replys n_R, the mobile node estimates its current distance from the border, $\tilde{d}(n_R)$. Note that the mobile node could receive more Reply messages than $\overline{N}_{neigh}$ because the communication radius R is greater than R_g. In this case, the distance from the partition border is approximated with $\tilde{d}(\overline{N}_{neigh}) = -R_g$.

The next position is defined on the basis of how many nodes it has to be connected with. The mobile node has to be connected with almost N_C nodes, thus it has to stop at distance $\tilde{d}(N_C)$ from the border. Hence, the mobile node has to cover a distance $\tilde{D}$ so that $\tilde{D} \le |\tilde{d}(n_R) - \tilde{d}(N_C)|$.

Table 1. Translation table ($\overline{N}_{neigh} = 7$)

n	2	3	4	5	6	7
$\frac{\tilde{d}(n)}{R_g}$	0.42	0.14	−0.11	−0.34	−0.56	−1

For example, with reference to Table 1, if the mobile node receives $n_R = 4$ replies and $N_C = 2$, the estimated distance that has to be covered is $\tilde{D} \le |\tilde{d}(4) - \tilde{d}(2)| = |-0.11 - 0.42| R_g = 0.53 R_g$.

Given the current position $\boldsymbol{x}_{cur}$ and the direction $\hat{v}$, the mobile node can calculate the next position $\boldsymbol{x}_{nxt} = \boldsymbol{x}_{cur} + \hat{v}\tilde{D}$. So, the Signalling interval ΔT_H is the time that the mobile node needs to reach the next position $\boldsymbol{x}_{nxt}$ starting from $\boldsymbol{x}_{cur}$. It depends on the motion algorithm, i.e., obstacle advoidance algorithm, collision advoidance algorithm and so on. Let us suppose that the mobile node moves in a straight line at constant speed V. In this case, the Signalling interval is $\Delta T_H = \frac{\|\boldsymbol{x}_{nxt} - \boldsymbol{x}_{cur}\|}{V} = \frac{\tilde{D}}{V}$.

The Response Interval ΔT_R

The *Response time* ΔT_R is the time the mobile node waits for the Reply messages. Since $\overline{N}_{neigh}$ is the expected number of replies when the node is in the safe partition, ΔT_R has to include at least $\overline{N}_{neigh}$ packet transmission intervals.

Furthermore, ΔT_R has to take into account the probability of collisions resulting from concurrent broadcasting. Usually, the MAC protocols use Random Backoff Scheme in order to reduce collisions. Each broadcast is delayed by a time period, called backoff time period. The protocols define how to select appropriately this backoff time. We define $\Delta T_{BO}(\overline{N}_{neigh})$ the expected backoff time period in presence of $\overline{N}_{neigh}$ nodes. So, the Response interval ΔT_R is defined as follows:

$$\Delta T_R = \overline{N}_{neigh} \Delta T_1 + \Delta T_{BO}(\overline{N}_{neigh})$$

where ΔT_1 is the time for transmitting a packet. Usually the Response interval ΔT_R is negligible with respect to the Signalling interval ΔT_H. In fact, the interval ΔT_R is a communication time, whereas ΔT_H is the time the mobile node needs to cover a given distance.

The Burst Parameters B and B_{min}

The parameter B is the number of Hello messages the mobile node broadcasts during the Verification Phase. This burst of messages is used to find how many neighbours in the safe partition the mobile node is in contact with. The parameter B_{min} is the minimum number of Reply messages the mobile node has to receive from a specific node in order to consider it as a neighbour.

As already specified, two nodes are neighbours if the probability of receiving a Reply in response of a Hello message is greater than or equal to a given $\mathcal{P}_g$. In order to evaluate this probability, a mobile node sends B Hello messages and records how many Replys it receives from each fixed node. Let us suppose that r_j is the number of replies broadcast by the node j. This counter gives us a rough estimation of the reception probability: if the r_j/B ratio is greater than $\mathcal{P}_g$, we assume to have a good link.

The parameter B could be defined via experiments on the basis of $\overline{N}_{neigh}$. That is, B is the minimum burst size so that the expected number of replying nodes is $\overline{N}_{neigh}$. Furthermore, given B and $\mathcal{P}_g$, the parameter B_{min} is chosen so that $B_{min} \geq \mathcal{P}_g B$.

5 Simulation Results

We implemented our algorithm over TinyOS [13] using the *nes*C programming language [9] and carried out performance analysis through simulation using TOSSIM simulator [11,12]. The simulated environment is a wireless sensor network where nodes are uniformly distributed to form a grid. Grid spacing is 8 feet and the radio communication range is 12 feet. Furthermore, we suppose that every safe node holds the current epoch. The mobile node moves in a straight line with a constant speed V=2 feet/s.

We used TOSSIM's *empirical* radio model, which defines packet loss rates based on measurements made by Woo et al. on RFM radio [8]. For our experiments, we defined the reception probability for discriminating between good and poor links to $\mathcal{P}_g = 0.8$. Before testing the algorithm, we performed the preliminary calibration of parameters $\overline{N}_{neigh}$, R_g, and B. The mobile nodes broadcasts

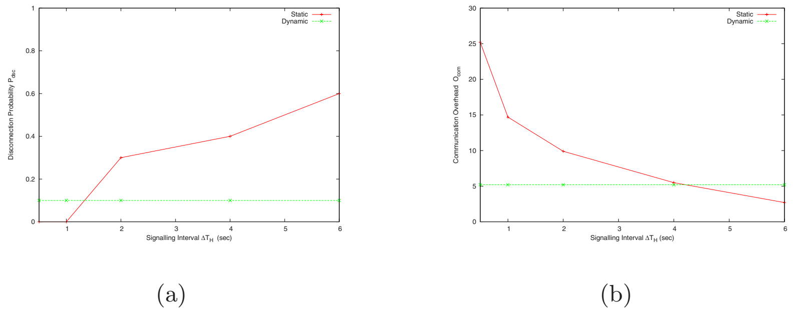


Fig. 7. $\mathcal{P}_{dsc}$ and O_{com} vs. static and dynamic ΔT_H (sec)

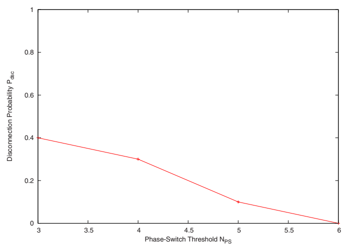
a burst of HELLO messages when it is within the safe partition. On the basis of the replying nodes and their position, the mobile node sets the parameters as follows: $\overline{N}_{neigh} = 10$, $R_g = 10$ feet, and B=10. Thus, given $\mathcal{P}_g$ =0.8 we have B_{min}=8.

We evaluate the algorithm performance in terms of communication overhead O_{com} and disconnection probability $\mathcal{P}_{dsc}$. More in detail, the communication overhead O_{com} is the average number of HELLO messages that the mobile node broadcasts. The average is computed over ten repetitions of the experiment. For each repetition, the starting point of the mobile node is randomly selected. The disconnection probability $\mathcal{P}_{dsc}$ is defined as the ratio between the number of simulations resulted in a connection loss (no good links between the mobile node and the safe partition) and the total number of runs.

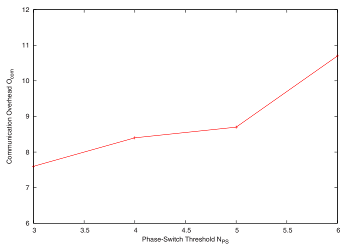
First, we examined the algorithm behaviour both whether the signalling interval ΔT_H is statically chosen and is adaptively calculated. In the case of a static predefined ΔT_H, the disconnection probability increases as the signalling interval exceeds a certain value. However, a short interval means a higher communication overhead. In fact, as shown in Figure 7, a static ΔT_H in the $[0.5, 2]$ range limits the disconnection probability in the $[0, 0.3]$ range. Nevertheless, in this case the communication overhead ranges from 9.9 to 25.2. On the other hand, the adaptive calculation of ΔT_H limits the disconnection probability to 0.1 and the communication overhead to 5.2. It is important to observe that an adaptive selection of ΔT_H is particularly effective when the safe partition is much larger than the communication range because the mobile node strongly reduces the amount of communication overhead it produces while traversing the safe partition.

Furthermore, we evaluated how the Phase-Switch threshold N_{PS} influences both the disconnection probability and the communication overhead. Figure 8(b) shows how small N_{PS} values reduce the communication overhead. In particular, they reduce the chance of false alarms, and thus unneeded bursts. Nevertheless, small N_{PS} values may cause an higher probability of disconnection from the safe partition (Figure 8(a)).

Our third experiment concerns how the connectivity threshold N_C affects the final position reached by the mobile node. We ran simulations with different



(a)



(b)

Fig. 8. $\mathcal{P}_{dsc}$ and O_{com} vs. N_{PS}

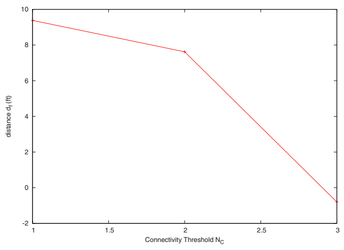


Fig. 9. Final distance d_f vs. N_C

values of N_C and measured the average distance d_f from the mobile node to the safe partition boundary. A negative value of d_f means that the mobile node is still in the safe partition and has not crossed the border. Results are shown in Figure 9. We notice that lower thresholds lead to higher distances (the node advances into the gap between partitions). On the other hand, higher thresholds make the mobile node stop early. For example, in case of $N_C = 3$, the mobile node does not cross the partition border ($d_f = -0.8ft$).

6 Conclusions

With reference to a WSN, we have presented a method for repairing network partitions based on mobile nodes. The paper has the following merits. First of all, it treats an important problem that, so far, has received limited attention. Furthermore, the paper suggests a method that is based on a few mobile nodes that move through the network reducing the communication overhead. The paper presents the main factors influencing the algorithm behaviour and performance and discusses their selection criteria. By simulation, the paper shows that the proposed method is effective in terms of disconnection probability and efficient in terms of communication overhead. Future steps consist in deploying an early prototype on the multi-agent platform we have been developing [3].

Acknowledgements

This work has been supported by the Commission of the European Communities under the Sixth Framework Programme Project IST-004536 "Reconfigurable Ubiquitous Networked Embedded Systems"(RUNES). Authors also wish to thank Daniele Quercia, a PhD student at the University College of London, who contributed to improve presentation.

References

1. Reconfigurable Ubiquitous Networked Embedded System (RUNES). European Integrated Project, FP6-IST-004536, `http://www.ist-runes.org`
2. Dini, G., La Porta, S., Pallottino, L., Savino, I.M., Bicchi, A., Danesi, A., Schiavi, R.: A safe and secure component-based platform for heterogeneous multi-robot systems. IEEE Robotics and Automation Magazine (to appear)
3. Pallottino, L., Savino, I.M., Schiavi, R., Dini, G., Danesi, A., Fagiolini, A., Bicchi, A.: A scalable platform for safe and secure decentralized traffic management of multi-agent mobile systems. In: REALWSN 2006. Proceedings of the ACM Workshop on Real-World Wireless Sensor Networks, Uppsala, Sweden (June 19 2006)
4. Alriksson, P., Nordh, J., Årzén, K.H., Bicchi, A., Danesi, A., Schiavi, R., Pallottino, L.: Component-based approach to the design of networked control systems. In: ECC 2007. Proceedings of European Control Conference, Kos, Greece (July 2–5, 2007)
5. Årzén, K.H., Bicchi, A., Dini, G., Hailes, S., Johansson, K.H., Lygeros, J., Tzes, A.: A component-based approach to the design of networked control systems. European Journal of Control 13, 261–279 (2007)
6. Cerpa, A., Estrin, D.: Ascent: Adaptive self-configuring sensor networks topologies. IEEE Transactions on Mobile Computing 3(3), 272–285 (2004)
7. Chong, C.-Y., Kumar, S.P.: Sensor networks: Evolution, opportunities and challenges. Proceedings of the IEEE 91, 1247–1256 (2003)
8. Ganesan, D., Krishnamachari, B., Woo, A., Culler, D., Estrin, D., Wicker, S.: An empirical study of epidemic algorithms in large scale multihop wireless networks (2002)
9. Gay, D., Levis, P., von Behren, R., Walsh, M., Brewer, E., Culler, D.: The *nes*C language: A holistic approach to network embeddedd systems. In: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, San Diego, California, USA, pp. 1–11 (June 9–11, 2003)
10. Johnson, D.B., Maltz, D.A.: Dynamic source routing in ad hoc wireless networks. In: Imielinski, Korth (eds.) Mobile Computing, vol. 353, Kluwer Academic Publishers, Dordrecht (1996)
11. Levis, P., Lee, N.: TOSSIM: A simulator for TinyOS networks (Novemeber 14 2003)
12. Levis, P., Lee, N., Welsh, M., Culler, D.E.: TOSSIM: accurate and scalable simulation of entire TinyOS applications. In: SenSys, pp. 126–137 (2003)
13. Levis, P., Madden, S., Gay, D., Polastre, J., Szewczyk, R., Woo, A., Brewer, E., Culler, D.: The emergence of networking abstractions and techniques in TinyOS. In: NSDI 2004. First Symposium on networked system design and implementation, San Francisco, California, USA, pp. 1–14 (2004)

14. Perkins, C., Bhagwat, P.: Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In: ACM SIGCOMM 1994 Conference on Communications Architectures, Protocols and Applications, pp. 234–244 (1994)
15. Perkins, C.E., Belding-Royer, E.M.: Ad-hoc on-demand distance vector routing. In: WMCSA, pp. 90–100. IEEE Computer Society, Los Alamitos (1999)
16. Shrivastava, N., Suri, S., Tóth, C.D.: Detecting cuts in sensor networks. In: IPSN, pp. 210–217. IEEE, Los Alamitos (2005)
17. Wood, A.D., Stankovic, J.A.: Denial of service in sensor networks. IEEE Computer (2002)