This is the published version (version of record) of:


Available from Deakin Research Online:

http://hdl.handle.net/10536/DRO/DU:30007963

Copyright: ©2007 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Using Mobile Agents to Detect and Recover from Node Compromise in Path-based DoS Attacks in Wireless Sensor Networks

Bai Li
Deakin University
baili@deakin.edu.au

Lynn Batten
Deakin University
lmbatten@deakin.edu.au

Abstract

Wireless sensor networks represent a new generation of real-time embedded systems with significantly different communication constraints from the traditional networked systems. With their development, a new attack called a path-based DoS (PDoS) attack has appeared. In a PDoS attack, an adversary, either inside or outside the network, overwhelms sensor nodes by flooding a multi-hop end-to-end communication path with either replayed packets or injected spurious packets. Detection and recovery from PDoS attacks have not been given much attention in the literature. In this article, we propose a solution using mobile agents which can detect PDoS attacks easily and efficiently and recover the compromised nodes.

Keywords: Denial of Service Attack, Detection, Recovery, Sensor Networks

1. Introduction

Wireless sensor networks (WSNs) are increasingly being deployed in mission critical applications, and for monitoring of critical events in areas such as national defense and homeland security applications for prevention and detection of nuclear, chemical and biological threats. Security issues related to deployment of wireless sensor networks in hostile environments are being investigated extensively. WSNs tend to be unstable and have limited resource capacity and so are vulnerable to DoS attacks.

The path-based DoS attack was first described in detail and named by Deng et. al. in [5]. They pointed out that a PDoS attack, by exhausting the batteries of several nodes, has the potential to disable a much wider region than simply a single path due to the tree structure topology of a WSN.

A standard PDoS attack begins with the compromise of member nodes and aggregator nodes which are then used to flood the intermediate and sink nodes with packets along the routing paths. The resulting excessive power consumption can lead to a quick death of a WSN because the nodes are unable to return to sleep mode in order to conserve power.

Few authors have tackled the problem of PDoS attacks on a WSN. In this paper, we introduce a novel method of using mobile agents in the WSN which permits us to detect these attacks and also recover from them. To our knowledge, the only application of mobile agents in sensor networks to appear in the literature is in the paper [19], but these are not used for detection of or recovery from attack. The work in this paper extends recent work of the authors [12] on detection of PDoS attacks in WSNs.

Our methods are described in detail in sections 3, 4 and 5. In section 6, we evaluate the usefulness of our techniques and in section 7, describe work which remains for future papers.

2. Working Model

We assume the WSN has a tree-structure topology, a single base station and four types of nodes as shown in Figure 1. A description of the node and base station functions can be found in [14].

![Figure 1. Tree Structure of a WSN](image)

The work of [19] demonstrates that mobile agents (MAs) are very useful in WSNs because of their ability to reduce network load, encapsulate protocols and are robust and fault-tolerant. We therefore add an MA to the WSN (several may be added in a large system) and assume that there is no communication restriction between nodes and the MA and that the MA has sufficient power to run the algorithms we need. We assume that the base station and MA are trusted.
and can never be compromised by attackers. In addition, in order to be usable in a malicious environment, we assume the node bootloader can not be compromised by the attacker. Our recovery method comprises three parts, as detailed in sections 5.1, 5.2 and 5.3. In all cases, it is sufficient for the WSN to have a single MA. However, the use of several MAs in a large WSN will reduce recovery time.

In order to distinguish between nodes which have been compromised and those which have not, we need to introduce a labeling for each node. We base this labeling on 3-dimensional co-ordinates in space [18], which can be obtained by geographical location using triangulation. For example, a node whose coordinates are 59°Latitude, 29°Longitude and height 2 metres has ID 59:29:2. (If the WSN is operating within a constricted space, a different scale can be used for measuring.) We assume that nodes cannot be positioned in the same geographical location.

In order to implement recovery when a substantial number of nodes has been compromised in a PDoS, we will focus on re-establishing network paths which have been destroyed. In order to do this efficiently, we assume that the MA, during the course of message sending across the network, has identified those nodes which attract most traffic and are therefore critical to maintenance of the network.

3. Detection Stage

Several methods for detecting DoS attacks in general networks have been proposed. One solution [20] is to use a MAC admission control rate limit, so that the network can ignore excessive requests without sending expensive radio transmissions. This limit cannot drop below the expected maximum data rate the network supports, though. However, if the nodes fail in one area, it could lead to high traffic flow in another area, because the sender can select different paths to transmit the packets across thus avoiding dead nodes.

More recently, en-route filtering schemes have been proposed for intermediate nodes to filter false data generated by malicious aggregator nodes as well as to detect intruders engaged in what we have termed PDoS attacks [21] [23]. The basic idea is that the intermediate nodes share some keys with the member nodes in a node group or cluster. Member nodes generate MACs for the reported data using the shared keys, and intermediate nodes can verify the MACs before forwarding packets. In the SEF scheme proposed by Ye et. al., the Bloom filter [21] is used to reduce the size of MACs and ensure their security. The intermediate nodes and member nodes use randomly pre-distributed keys to generate and verify MACs. In this scheme, it is highly likely that the false data will be dropped by one of the intermediate nodes and will not reach the base station. However, there are several problems with the SEF scheme. First, SEF uses a probabilistic approach. It cannot guarantee that every spurious packet will be filtered out on the path. In addition statistically, a spurious packet will be forwarded for a certain number of nodes before it is filtered out. Second, the message overhead of SEF is still large. The size of the Bloom filter is 14 bytes long, which is about half of the data payload of a TinyOS packet.

In the interleaved key scheme proposed by Zhu et. al. [23], member nodes and intermediate nodes set up interleaved keys using randomly pre-distributed keys. These interleaved keys and hop-by-hop authentication ensure that the base station will detect any false packets when no more than a certain number of nodes are compromised. The problem of the interleaved key scheme is that there is no efficient mechanism to authenticate two nodes to each other through multiple hops. In addition, the communication overhead of the pair-wise key establishment for multi-hop nodes is large, and the process is slow.

Jing et. al. [5] developed a method of using one-way hash chains to protect end-to-end communications in WSNs against PDoS attacks. It prevents PDoS attacks from the intermediate nodes or from outside sources capable of launching PDoS attacks, since an adversary cannot generate the next valid OHC number, while replayed old OHC numbers will be dropped. Second, the memory and computational costs of OHC execution are quite lightweight. Third, this scheme tolerates packet losses. But there are also some obvious disadvantages. One of them is it cannot constrain PDoS attacks by the compromised nodes in WSNs, especially the nodes which store the hash function.

All of the work using MACS results in high local computing costs and subsequent death of nodes. The interleaved key scheme has extremely high communication overhead. We therefore adapt the low cost one-way hash chain idea of Jing et. al., but efficiently employ a mobile agent to detect if the sender of the hash has been compromised.

Our detection solution includes a one-way hash chain, a traffic control algorithm, detection of node compromise and node failure and a voting algorithm. We choose the one-way hash chain as it has low computational requirements and is easy to deploy [5]. The one-way hash chain is used to detect any compromised node except for source nodes. We use a traffic control algorithm to detect the misbehavior of source nodes since the adversary could compromise them to access the one-way hash function. We also use message broadcasting from the MA to distinguish between node failure and node compromise, because node failure which is a technical problem rather than an attack could generate higher traffic flow in a local area or along some paths. Finally, a voting algorithm is used to decide whether suspicious nodes have been compromised based on the votes from their neighbour nodes.
3.1. Traffic Control Algorithm

When one node wants to send packets to its neighbour nodes, not only will it include a hash function value, but also it includes its node ID in order to let the MA recognize the source of packets and record them into a traffic table. In Table 1, for each 5 second interval, the number of packets passing through the corresponding node is given in the column. If the traffic in each interval is normal, that is, below a certain given threshold W, the MA will simply delete the contents of the table and refresh it for the next time-frame. If the traffic of certain nodes in the table is abnormal (i.e. is above W), the MA will take further steps to tell whether the node has failed or been compromised.

Table 1. Number of packets for a given node in several time intervals.

<table>
<thead>
<tr>
<th>Node ID</th>
<th>0-5 secs</th>
<th>5-10 secs</th>
<th>10-15 secs</th>
<th>15-20 secs</th>
<th>20-25 secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>21:34:1</td>
<td>22</td>
<td>43</td>
<td>72</td>
<td>32</td>
<td>45</td>
</tr>
<tr>
<td>22:35:0</td>
<td>13</td>
<td>65</td>
<td>37</td>
<td>99</td>
<td>24</td>
</tr>
<tr>
<td>24:30:2</td>
<td>87</td>
<td>55</td>
<td>32</td>
<td>40</td>
<td>14</td>
</tr>
</tbody>
</table>

Our algorithm detects abnormal traffic by choosing a threshold W and counting the number of times the traffic through a specific node exceeds W in a fixed table. If this occurs more than say w times, the MA then considers the node to be compromised or failed, but cannot tell which. Both w and W need to be determined by experiment. Initial choices can be made, followed by adjustments.

The procedure for Node Failure Detection is described in the next section.

3.2. Detection for Node Failure

Both node compromise and node failure can lead to abnormal traffic flow during a short period or in a localized area of a WSN. So we have to eliminate the node failure condition in order to make our Traffic Control algorithm work well. When the MA goes into PROCEDURE NodeFailureDetection, it will simply broadcast a check message to all the nodes in the WSN. The MA counts the number of acknowledgment (ACK) packets received and compares with the number of nodes which do not reply, in order to determine the failed nodes. Responding nodes include their IDs and this allows the MA to determine the IDs of the non-responding nodes. The MA concludes that all non-responding nodes have failed and reports this to a technical support team.

3.3 Voting Algorithm for Node Compromise

In this section, we now determine which of the nodes that did respond to the MA message are in fact compromised. To do this, we use the MA to broadcast a new random message at random times to all the responding nodes exhibiting abnormal traffic patterns. When these nodes receive this message, they will try to send the same message to their neighbor nodes. However, as described in the well-known Byzantine Generals Problem [11], compromised nodes will attempt to transmit conflicting information to other parts of the system. We base our voting algorithm on this fact: if node A receives the same message from node B and the MA, node A will vote that node B is not compromised and send back this voting result along with both IDs to the MA. When the MA has received a certain number of votes against a node, the MA will decide that node has been compromised. The MA now takes steps to recover the compromised node.

We can assume that the WSN has sufficient uncompromised nodes to make such a voting algorithm reliable. On the other hand, in very large-scale WSNs, there could be thousands of nodes, in which case several MAs could be applied in order to implement the detection algorithm. One advantage of using MAs is that they can provide seamless technology to a very big network. So there are no problems in deploying more of them into WSNs.

In addition, node compromise is a fairly slow process, as mentioned in [2] when detection and recovery methods are in place. Thus it is unlikely, that at any given time, more than half the nodes would be compromised.

4. Response Stage

After detection, we need to assess the level of attack. Our response method is based on the number of compromised nodes. The MA will calculate the number of compromised nodes based on the voting results from sensor networks. When the MA receives all the voting results from the nodes, it will apply the response algorithm described below and make a decision on what level of recovery needs to be implemented. We assume that technical interference in the transmission of votes is kept to a minimum.

In determining an appropriate response, the one significant factor is the number of compromised nodes. The MA will choose a number $N_{\text{max,b}}$ below which the situation is deemed to be easy to manage and the compromised nodes quickly recovered. The MA will also choose a number $N_{\text{max}}$ above which the situation is extremely serious and extraordinary steps need to be taken. Between these two values a moderate approach to recovery will be used. In practical situations, these maximum and minimum values would be determined using experimental data.

5. Recovery Stage

Before presenting our three recovery levels in detail, we
examine the literature on node recovery.

For a hardware-based solution, the Mica mote node [7] uses a coprocessor to reprogram the main processor, which ensures that the application can always be recovered. The eXtreme Scale Mote (XSM) [6] uses a timer to ensure that a bootloader eventually regains execution control. The XSM bootloader can detect a golden gesture – three manual resets in quick succession – and revert to a factory image. If the XSM’s timer fires, the node invokes a special management program which waits for commands to be issued but no automatic recovery mechanism exists. Because this type of hardware solution is expensive and hard to implement for different node type, we do not consider it in our setting.

Many software methods have been proposed for node recovery in WSNs. Some of the most recent are described in XNP [9] [4], MOAP [17], Kapur et. al. [15] [22] and Jeong et. al.

5.1 Program-loaded Recovery

In each of the recovery stages, the speed of recovery is crucial in order to prevent the attack from moving to a higher level.

The program-loaded recovery approach is implemented when only a few nodes have been compromised. In this case, we have the MA reload the program into each compromised node. The aim is to reconstitute compromised nodes as quickly as possible.

Our method is based on Jeong et. al.’s work in that we compare the original block image with the current image and reload the difference. We do not reload if the images are identical. Our model only runs the “DOWNLOAD” command after the MA finds that there is a difference between two blocks (the malicious and original one). Due to the limitation on processing capacity, the nodes can not self-program the chips. At the same time, too much energy consumption could make the nodes die earlier than necessary. This recovery method is most suitable when only a relatively small number of nodes has been compromised because the block image comparison and download components are time consuming.

In detail, the process is as follows. The MA connects each compromised node (previously identified) with wired cable, downloads the current code and compares this code block by block with the original program of which it has a stored copy. (This is only practical for a very small number of nodes; to deal with larger numbers, more MAs can be deployed.) Since the node has already been identified as compromised, we assume that there will be differences in one or more blocks. If fewer than half the blocks differ, the MA reloads the difference into the node. If half or more of the blocks differ, the MA reloads the full original code. At the same time, the MA analyzes the differences and produces appropriate security patches for the code (e.g. short-term monitoring, message sending limitation); these are downloaded with the code. The security patches are designed to prevent the attack from escalating and they will be passed from node to node as the nodes communicate.

A copy of the original program of each of the four node types is stored in the MA block by block (for instance, one block may be 128bits). The MA keeps two hash tables: one is a checksum hash table used to compare the current program and malicious program easily; the other is an address hash table which records the location of the code block.

Figure 2 shows code divided into fixed block sizes and indicates the use of the checksum hash table in determining differences between blocks. The checksum pair is inserted into the hash table for look-up.

Recovery time for each node in this method is defined as the sum of the time of comparison for difference, the time to send a transmission command, the time to download and the reset time. Normally we do not count the development time for any security patch developed.

5.2. Blacklist Recovery

In this situation, a medium number of nodes has been compromised. If the network is small, we can employ the program-loaded recovery method. In a larger network, in addition to reprogramming compromised nodes as described in section 5.1, we also prevent these nodes from compromising other nodes in the network. A key part of our approach is to use the paths along which the attacker is applying the PDoS.

Blacklist recovery involves the development of a list of compromised node IDs (the blacklist). The MA implements the program-loaded recovery method and at the same time, it builds and maintains a blacklist of compromised nodes as these is detected using the detection method of section 3.3.
A node ID is deleted from the blacklist once the node has been recovered either by the technical team or by the MA. The MA then uses the blacklist to prevent compromised nodes from compromising additional nodes by sending the updated blacklist to all neighbours of nodes either added to or deleted from the list and follows this up with an acknowledgement request to ensure it was received. When a node receives a message from another node, it checks that node’s ID and if the ID is on the blacklist, does not accept the message.

Blacklist entries either expire or are deleted when the MA has detected normal traffic flow for several time intervals. A timer can be set up for expiration after an entry has been created in the blacklist table.

In anticipation of a greater number of nodes being compromised than can be suitably handled by this method, the MA also keeps a whitelisted list of node IDs which have never been suspect. The whitelist will be used in case it is necessary to move to the next recovery stage.

The recovery time for each compromised node in this method can be simply defined as the sum of the time of blacklist generation related to verification of that node, the time of checking that the node is compromised, the time for repairing it and the reset time.

5.3. Node-redundant Recovery

In this situation, the number of compromised nodes has moved above the threshold and we expect that very few paths are still available for operation in the WSN. Regaining control of random compromised nodes may not be sufficient to re-introduce critical pathways through the network. A quick and efficient recovery of network flow is required in this situation.

Our method is based on the redundant recovery concept described in [10]. As mentioned in our working model description, the MA has identified a number of nodes as being crucial to the maintenance of traffic flow across the WSN. In recovery, the MA then first backs up the latest process image of any of these crucial nodes which have become compromised, and continues to give these nodes top priority for recovery.

The technical officers replace crucial compromised nodes if the power of the nodes is almost gone. After reparation and maintenance, the MA reloads the latest process image to the nodes. At the same time, the MA closes down all nodes along fully compromised paths which cannot be maintained.

Recovery time for each node in this method can be simply defined as the total sum of the time for checking a compromised node, the time to repair a compromised node, the time to reset and the time for new node deployment.

When we calculate how many paths go through a node, we use the number of incoming paths and of outgoing paths. Let \( P_{\text{total}} \) be the number of paths going through each node. Let \( P_{\text{incoming}} \) be the number of incoming paths for each node. Let \( P_{\text{outgoing}} \) be the number of outgoing paths for each node. The following equation then holds:

\[
P_{\text{total}} = P_{\text{incoming}} \times P_{\text{outgoing}}
\]

This formula determines the level of importance of each node in a WSN and can be used by the MA to determine crucial nodes.

In determining priority of recovery after the crucial nodes have been recovered, the MA can use the number of paths through each node as determined in the above equation.

6. Evaluation

6.1. Detection

The simulation model was built around the cluster based ad hoc wireless network infrastructure described in [3]. The cluster infrastructure uses polling as the channel access scheme within each cluster, with code separation across clusters. The clusterhead polls each of the local nodes to allocate the channel. It was based on the Maisie/PARSEC simulation platform developed at UCLA [1]. For our simulation model we generate 50 nodes in a 200x200m square area. The radio transmission range is 40 meters and the data rate of the wireless link is 2Mbps. The data traffic is generated by constant bit rate with an inter-arrival time of 25ms. For each node there are two transmission queues, one for control packets and one for data packets. The control queue is used for control packets such as route requests or recovery requests and it always has higher priority than the data queue. We also set up a timer if a packet has not reached the destination in 10 seconds. The packet length for data packets is 10k bits with 500 bits for the header message. The queue length for data packets is 50 for all nodes. The timer which is used to resend the message if there is no reply is set to 50ms.

Figure 3 compares the time to detect attacks between our method, the en-route filtering method and the SEF method.

![Figure 3. Detection Time under Different Attack Rates](image-url)
We measure the detection time in seconds and the attack rate in SYN/sec.

As indicated, our method is significantly better than the others when the attack rate is between 17 SYN/sec and 28 SYN/sec.

![Energy Comparison on Detection](Image)

Figure 4. Energy Comparison on Detection

Figure 4 shows the energy comparison between each detection method. Our method uses substantially less energy than the other methods.

6.2. Recovery

6.2.1. Program-loaded Recovery

To evaluate the performance of the program-loaded recovery method, we use the BcastM.nc file from the TinyOS sensor system as a test scenario. This file is used to forward command messages that it receives to other nodes in the network. This is not a very long file but is sufficient to implement our experiment. Various changes can be made on the program code in order to set up the cases.

We count the recovery time of the test set for each node. We do not consider the impact from the change of code. We take the following two cases as a test scenario:

- **Case 1:** This is the case with the minimum amount of change on program code. We only modify constants in the program.
- **Case 2:** This is the case with a major amount of change on program code. We add a few lines of code into the program and change the structure of some code.

To evaluate the performance of this recovery method, we estimated and measured the recovery time and power consumption. We do not use reset time as it is always a fixed time period.

Let $T_{\text{comparison}}$ be the time for comparison, $T_{\text{command transmission}}$ be the time for command transmission, $T_{\text{difference download}}$ be the time for download and $E_{\text{measured}}$ be the energy consumption measured.

<table>
<thead>
<tr>
<th>Method</th>
<th>$T_{\text{comparison}}$</th>
<th>$T_{\text{command transmission}}$</th>
<th>$T_{\text{difference download}}$</th>
<th>$E_{\text{measured}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our Method</td>
<td>259</td>
<td>189</td>
<td>47</td>
<td>78</td>
</tr>
<tr>
<td>Incremental Programming</td>
<td>274</td>
<td>241</td>
<td>49</td>
<td>134</td>
</tr>
</tbody>
</table>

Table 3 compares our method with incremental programming [8]. Our recovery time is much less, especially for case 1. We use the MA to do the comparison work saving processing time on nodes. Our copy time is also lower than that in [8].

6.2.2. Blacklist Recovery

The goal of this experiment is to understand the costs of our recovery mechanism, and see if it is feasible for sensor network recovery. We evaluate this recovery method by testing the time for blacklist generation, $T_{\text{blacklist generation}}$, because the times for checking and repairing, $T_{\text{check}}$ and $T_{\text{repairing}}$ can involve such things as technician support. $T_{\text{blacklist generation}}$ includes the read time from the node array, blacklist algorithm running time and write time to record into the blacklist. In our experiment, we set up blacklisting lower thresholds of 0%, 20%, 40%, 60%, 80%, 90% in a 10-node WLA determining how many nodes have been compromised. The objective is to see how well the method works at each of these lower threshold levels. We test the reliability of network links in each case as follows. One node, A, is given top priority for maintenance and is assumed to be always reliable (never on the blacklist). Each node in the WSN generates one packet every two seconds. Node A broadcasts a message to detect compromised nodes. At the same time, we measure the power consumption for blacklist broadcast. Whether it is in an acceptable range is also important to our recovery system. We test it by using an MICA2 mote connected to a multimeter in order to obtain the power consumption after broadcasting the blacklist to its neighbour nodes. Figure 6 shows the power consumption on different blacklist thresholds.

Upon analyzing the experimental data, we found that on average 99.6% of traffic going through compromised nodes had been blocked after 20 trial experiments. This method can protect the nodes from being compromised efficiently as shown in Figure 7. But it could also lead to abnormal local traffic flow. We found that a lower blacklist threshold of less than 50% of nodes will be recoverable in reasonable time.
Packet Delivery Rate at different blacklisting thresholds

![Packet Delivery Rate at Different Blacklisting Thresholds](image)

Figure 6. Packet Delivery Rate at Different Blacklisting Thresholds

Energy Consumption at Different Blacklisting Threshold

![Energy Consumption at Different Blacklisting Threshold](image)

Figure 7. Power Consumption on Blacklist Recovery

With a single MA. This is based on levels of power consumption and recovery time after 30 trial experiments. Otherwise, the network needs to employ more than one MA to complete the recovery.

As we see from simulation results in Figure 7, at low blacklisting thresholds, the network only uses an extremely low power to send and receive the data packets. As the number of blacklisted nodes increases, there are more active periods on the working nodes, thus consuming more energy. However compared with the total power consumption of a WSN, this is much less compared with the power consumption on the normal function of nodes [16].

6.2.3. Node-redundant Recovery

In the simulation, we did not test the recovery time on deploying backup nodes into the current wireless sensor network. If there is no available path in a WSN, we can try to set up at least one available path to keep the network working. Based on the algorithm we used to find the most important node for each round, we can measure the time to find that node by running the algorithm on one node to get the time of calculating the number of paths. We simulated the situation that an aggregator node is set up with its direct neighbor nodes, and measured the number of paths going through this node during the available path retrieval. There are in total 20 nodes which include one aggregator node, 13 member nodes and 6 neighbor nodes. This means we have 13 incoming paths and 6 outgoing paths. We can calculate the time of running the algorithm on the member nodes and intermediate nodes. We also set up the network with 34 nodes including 24 member nodes and 10 intermediate nodes, 48 nodes including 33 nodes and 11 intermediate nodes, and 69 nodes including 47 member nodes and 22 nodes.

From the experiment, we found that times are 13.3 ms, 23.1 ms, 32.9 ms and 47.6 ms respectively for the 20 node, 34 node, 48 node and 69 node cases to locate at least one available path in the WSN. We simulated the available path setup time with sensor networks of different density. We see that the setup time of the available path linearly increases as the size of the sensor networks increases. In addition, we take into consideration the travel time of the MA, the processing time of the MA and the number of MAs used in this recovery. See Figure 8.

We found that the time needed to locate an available path depends on the number of compromised nodes on that path. More nodes on the path lead to more retrieval time. Therefore, the recovery time depends on the number of nodes on that path needed to be retrieved. It is a good idea to deploy the new nodes for urgent recovery when most of the nodes are compromised.

![Time to set up available paths based on different number of nodes](image)

Figure 8. Time to set up available paths based on different number of nodes
7. Conclusions and Future Work

We have presented a new detection and recovery method for node compromise in PDoS attacks in a WSN and evaluated our results, based on small WSNs, including comparison with other work. Our work demonstrates the feasibility of full recovery methods when few nodes have been compromised and prevention of further disintegration when many nodes have been compromised.

In future work, we will implement these detection and recovery methods in large-scale WSNs and extend the role of the MA in the detection and recovery procedures.

8. References