ASF: an Attack Simulation Framework for wireless sensor networks

Gianluca Dini
Dipartimento di Ingegneria dell'Informazione
University of Pisa, Pisa, Italy
Email: g.dini@iet.unipi.it

Marco Tiloca
Dipartimento di Ingegneria dell'Informazione
University of Pisa, Pisa, Italy
Email: m.tiloca@iet.unipi.it

***Abstract*—Wireless Sensor Networks are vulnerable to quite a good deal of logical and physical security attacks. However, providing security countermeasures for every possible attack is practically unfeasible for cost and performance reasons. Thus, it is vital to properly rank security attacks in order to establish priorities and then select appropriate countermeasures. In this paper, we present ASF, an attack simulation framework that allows us to describe attacks and quantitatively evaluate their effects on the application and network behavior and performance. ASF helps users to evaluate the impact of an attack, a crucial step in the attacks ranking activity. Also, we present an early prototype of ASF built on top of the popular simulator Castalia. Finally, we show the capabilities of ASF by analysing four attacks against a realistic application scenario.**

I. Introduction

Nowadays, Wireless Sensor Networks (WSNs) are a widely adopted technology in several fields of application. Their consolidated success, both in the academic and industrial world, is mainly due to standards availability, low cost of sensor nodes, and world-wide communities support. Their uses include home automation, plant monitoring, military surveillance, disaster recovery situations, environment monitoring, and many others.

A WSN may be affected by a great number of security threats and attacks. WSNs are subject to *logical attacks* (aka *cyber attacks*), as an adversary simply equipped with a radio receiver-transmitter can easily eavesdrop as well as inject messages. Furthermore, WSNs are also subject to *physical attacks*. Actually, WSNs are often deployed in environments that are open, unattended, and possibly hostile. This allows an adversary to physically attack sensor nodes in order to reprogram, misplace, or, even, break them. All these attacks may lead to unreliable data collection, inaccuracy in controlling processes, or even safety problems.

There are lot of possible attacks against WSNs [9][16][19]. They can have different objectives, be performed at different levels, and result in different effects. Physical attacks comprise node capture [13], and tampering with sensor nodes [6]. On the other hand, logical attacks include the HELLO flooding attack [8], the Sybil attack [12], the sinkhole attack [8], the wormhole attack [21], the blackhole attack [7], and the Distributed Denial of Service (DDoS) attack [20].

However, providing a security countermeasure for every possible attack would result in a huge security overhead, which is difficult to afford and would overwhelm the scarce resources available on sensor nodes [14][15]. Thus, it is vital to properly rank security attacks, in order to evaluate their severity, state protection priorities, and select appropriate countermeasures. That is, it would be useful to quantitatively evaluate the effects of attacks, in order to assign them a priority, and select adequate countermeasures. Most of the times, this information becomes available when it is too late, i.e. once attacks have already occurred. Instead, it would be better to have it even before network deployment takes place.

In this paper, we present ASF, our attack simulation framework for WSNs. Thanks to ASF, users can describe attacks, reproduce their effects, and quantitatively evaluate their impact on the overall functionality and performance. ASF helps users to evaluate the severity of an attack, and therefore constitutes a valuable tool for the attack ranking process. We show that the architecture of ASF is general and independent of the underlying network simulator.

Also, we present an early prototype of ASF built on top of Castalia [2], the popular WSNs simulator. Then, we show the capabilities of ASF by considering a realistic case study and analysing four different attacks. Specifically, we compare the effects of these attacks on system functionality and performance with respect to the case when the system is attack-free.

The rest of this paper is organized as follows. Section II discusses related work. In Section III, we present the ASF framework, and our prototype implemention. Section IV presents the application scenario we refer to, while Section V shows and discusses results of attacks simulations. Finally, we draw our conclusive remarks in Section VI.

II. Related work

Simulative analysis can provide a quantitative idea of the effects of attacks. However, little work has been done on attacks simulation so far. In fact, most of the adopted approaches focus on network simulation and performance evaluation. Also, they consider analytical models or algorithms aimed at *detecting* specific attacks upon their occurrence. Then, simulation is typically used to validate such models.

For instance, in [24], the authors describe a distributed wormhole detection algorithm, and show simulation results in order to prove its low false toleration and false detection rates. In [17], Kaplantzis *et al.* propose an intrusion detection scheme which detects blackhole and selective forwarding attacks.

Also, simulative results are presented to validate such scheme. In [10], the authors propose a scheme based on weighted-trust evaluation, aimed at detecting malicious nodes, and verify its correctness and efficiency by means of simulations. Bonaci *et al.* consider physical node capture attacks, develop a network response strategy, and validate it by means of simulations [18].

In [23], Wang and Bagrodia present *SenSec*, a framework for security evaluation in WSNs. SenSec allows for simulating the occurrence of attacks against the network, by injecting events into real application simulators. They test SenSec on *TïQ* [22], a framework for executing TinyOS applications [5], and use it to evaluate the impact of a number of attacks against routing and time synchronization protocols.

III. Attack simulation framework

The ASF framework provides: i) an *Attack Specification Language* to simply specify attacks; ii) an *Attack Database* to store attacks descriptions; iii) an *Attack Compiler* to convert attacks descriptions into XML configuration files; and iv) an *Attack Simulator* to evaluate effects of attacks. Figure 1 shows an overview of the ASF framework architecture.

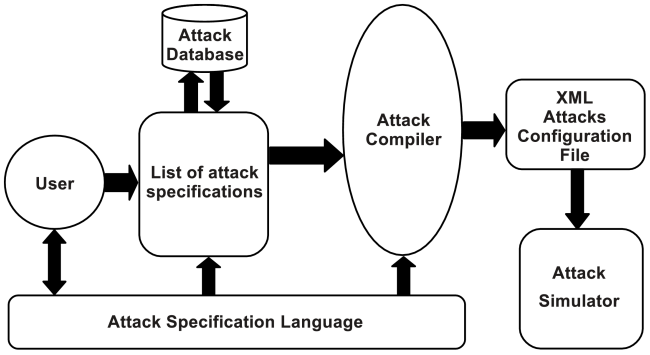

Fig. 1. Overview of the ASF framework architecture.

First, the user specifies the attacks to be evaluated by means of the *Attack Specification Language* (*ASL*), and possibly stores attacks descriptions in the *Attack Database*. Then, the *Attack Compiler* (*AC*) takes such specifications as input, and converts them into an XML *Attacks Configuration File* (*ACF*). The user must specify the name of the specific simulator to be used. By doing so, XML statements in the *ACF* can be correctly parsed and acquired by the adopted simulator. Finally, the *Attack Simulator* takes the *ACF* as input, in order to reproduce attacks and evaluate their effects.

A. Attacks specification

The *Attack Specification Language* (*ASL*) is a collection of *primitives* which allow users to specify attacks to be evaluated. From our framework standpoint, we define an attack as a sequence of *events*, which takes place atomically. That is, the user specifies the sequence of events that compose an attack by means of *ASL* primitives.

We consider two sets of primitives. *Node primitives* allow for altering nodes behavior, thus performing physical attacks. Instead, *packet primitives* allow for performing a number of actions on network packets, thus carrying out logical attacks. The following two *node primitives* are available.

destroy*(nodeID, t)*;
At time *t*, remove node *nodeID* from the network.

move*(nodeID, t, x, y, z)*;
At time *t*, move node *nodeID* to position (*x*;*y*;*z*).

Also, the following six *packet primitives* are available.

drop*(packet)*;
Discard the packet *packet*.

create*(packet, field, content)*;
Create a new packet *packet*, and fill its field *field* with *content*.

clone*(srcPacket, dstPacket)*;
Create a new packet *dstPacket* as a perfect copy of *srcPacket*.

change*(packet, field, newContent)*;
Write *newContent* into the field *field* of packet *packet*.

retrieve*(packet, field, content)*;
Retrieve the content of the field *field* of packet *packet*, and write it into *content*.

put*(packet, dstNodes, direction, delay)*;
After *delay* milliseconds, put packet *packet* either in the transmission or reception buffer of all nodes in the *dstNodes* list. Possible values for *direction* are TX and RX, which refer to the transmission or reception buffer, respectively.

The *ASL* allows for defining three different kinds of attacks, i.e. *physical attacks*, *conditional attacks*, and *unconditional attacks*. In the following, we describe how to specify them.

Physical attacks. They consist in a single event, modeled by either the *destroy()* or *move()* node primitive. As described above, such primitives require to specify the time t at which the event takes place, and the node affected by the attack.

Conditional attacks. This class of attacks considers a number of nodes that intercept packets to be examined, and, potentially, manipulated or even discarded. The user must specify the time t starting from which the attack takes place, and the list of network nodes that perform the attack.

Events occurrence may depend on the evaluation of a conditional statement, namely a *packet filter*. We define a packet filter as a set of simple boolean conditions $c_1, c_2, ..., c_N$ joined by logic operators, i.e. *AND*, *OR*, and *NOT*.

What follows is an example of conditional attack specification. Starting from time 200 s, nodes 1, 2, and 7 intercept all packets travelling through their communication stack. Then, such nodes check if each intercepted packet satisfies the packet filter. In case of a positive match, the attack is actually reproduced, i.e. the specified list of events is executed.

```
from t = 200; nodes = "1 2 7" do {
  if(<packet filter>)
    <List of events>
}
```

Unconditional attacks. Unlike conditional attacks, these attacks are not related to interception of packets by network nodes. However, new packets can be created, and possibly cloned, in order to be naughtly injected into the network. The user must specify the time t starting from which the attack takes place, and the occurrence frequency f according to which the attack has to be repeatedly reproduced over time. For this class of attacks, events occurrence does not depend on the evaluation of conditional statements.

What follows is an example of unconditional attack specification. Starting from time 200 s, the attack occurs repeatedly every 10 s, i.e. the specified list of events is executed.

```
from t = 200; every f = 10 do {
  <List of events>
}
```

In the following, we consider different kinds of attacks, and specify them by means of *ASL* primitives. For the sake of simplicity, we refer to packet fields with a simplified and extremely general notation. However, in a more general case, the user must be aware of which specific network protocols are in use. That is, for each communication layer, she must be aware of packet header structures and fields.

In such a general case, the user relies on a *packet.layer.field* notation, in order to access the field *field* of packet *packet* in the header of layer *layer*. Being available this information, it is possible to access specific header fields, in order to check and alter their content. As a practical example, the OMNeT++ platform [3] and the WSNs simulator Castalia [2] provide a set of objects, namely *descriptors*, aimed at handling packets of a given communication layer and accessing their header fields.

Node removal. Nodes 5 and 10 are removed from the network, at time 200 s and 500 s, respectively.

```
destroy(5,200);
destroy(10,500);
```

Node misplacement. Node 10 is moved to position $(80; 10; 0)$ at time 200 s. Similarly, node 11 is moved to position $(60; 10; 0)$ at time 200 s.

```
move(10,200,80,10,0);
move(11,200,60,10,0);
```

Node reprogram. Starting from time 400 s, nodes 5 and 7 replace the original payload of application data packets sent by node 10 with the minimum possible value. Then, data packets are regularly sent to their scheduled destination.

```
from t = 400; nodes = "5 7" do {
 if(packet.APP.source==10 &&
    packet.APP.type==DATA)
      change(packet,APP.payload,MIN);
}
```

Wormhole attack. Starting from time 200 s, MAC data packets sent by node 3 are intercepted. Then, they are provided to distant nodes 15, 17, and 18, which receive them in their reception buffer after 100 milliseconds. This is basically a wormhole attack, where a subset of packets are captured from a given portion of the network, and forwarded to a number of distant nodes through a low-latency channel.

```
dstList={15,17,18};
from t = 200; nodes = "*" do {
 if(packet.MAC.source==3 &&
    packet.MAC.type==DATA)
      put(packet,dstList,RX,100);
}
```

As mentioned above, attacks specifications can be stored into the *Attack Database*. Of course, the user can retrieve them subsequently, by querying the *Attack Database*. Querying parameters include the lapse of time during which attacks are supposed to be performed, or a list of involved nodes.

Thanks to the *Attack Database*, the user does not have to necessarily define new attacks from scratch. Instead, she can rely on stored specifications, or modify their behavior and severity for different network and application scenarios.

B. ASF network architecture

As depicted in Figure 2, ASF considers every node as composed by a generic *Sensing & Application* module, a *Communication stack* module, and a *Local Filter* module.

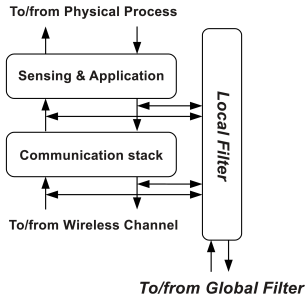

Fig. 2. Sensor node architecture in ASF.

The Sensing & Application module may be composed of different submodules, which model the actual node application as well as physical sensing processes. The Communication stack module may include an arbitrarily complex combination of communication layers, e.g. Routing and MAC. Finally, the Local Filter module intercepts all packets travelling through the communication stack. Thus, it is able to collect packets that the node application has transmitted or is about to receive.

The Local Filter can inspect and alter packets content, add new packets to be transmitted, or even discard them. Also, it can alter the node behavior at different layers, change the node position in space, or even remove the node from the network.

Furthermore, one single Global Filter module is connected with every Local Filter module. That is, Local Filters of different nodes can communicate and cooperate with each other through the Global Filter module, in order to reproduce and evaluate more complex attacks, e.g. a wormhole attack.

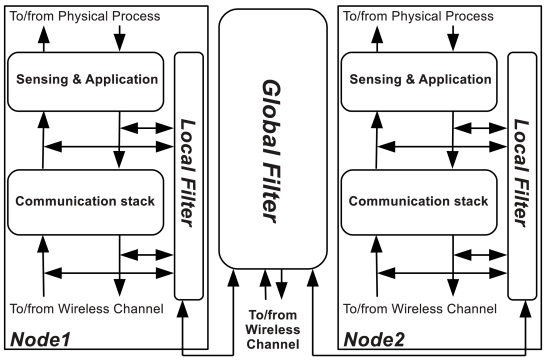

Fig. 3. Overview of ASF network architecture.

Figure 3 shows the overall ASF network architecture, in the presence of two sensor nodes, i.e. *Node1* and *Node2*.

C. Attacks reproduction

An *Attacks Configuration File* (*ACF*) is divided into three sections. The first section regards physical attacks, and contains the list of *node primitives* to be performed. We recall that each *node primitive* specifies the involved node and the time when the attack takes place.

The second section of the *ACF* consists of the list of conditional attacks to be reproduced. For each one of them, the following elements are specified: i) the time starting from which the attack occurs; ii) the set of nodes that perform the attack; iii) the packet filter; and, finally, iv) the list of *packet primitives* to be executed.

Finally, the third section of the *ACF* consists of the list of unconditional attacks to be reproduced. For each one of them, the following elements are specified: i) the time starting from which the attack occurs; ii) the occurrence frequency according to which the attack has to be performed over time; and, finally, iii) the list of *packet primitives* to be executed.

At its startup, the Attack Simulator retrieves the configurations of the attacks to be reproduced from the *ACF*. Then, the following data structures are initialized.

Physical attacks. The simulator considers every node n among the N nodes involved in at least one physical attack. For each node n, the simulator creates a list of physical attacks L_{PA}, containing N_{PA} elements. Such elements represent physical attacks that involve node n, and are ordered in a chronological fashion, according to their occurence time. Finally, the simulator starts N sets of timers, one for each node n. That is, for each node n, it starts N_{PA} timers, one for each attack in L_{PA}, thus scheduling their occurrence.

Conditional attacks. The simulator considers every node n among the N nodes involved in at least one conditional attack. For each node n, the simulator creates a list of conditional attacks L_{CA}, containing N_{CA} elements. Such elements represent conditional attacks performed by node n, and are ordered in a chronological fashion, according to their starting time. Each element includes: i) the packet filter; and ii) the list of events that define the attack. Finally, the simulator prepares N sets of timers, one for each of the above mentioned nodes. Then, for each node n, it starts N_{CA} timers, one for each attack in L_{CA}, thus scheduling the beginning of their occurrence.

Unconditional attacks. The simulator creates a list of unconditional attacks L_{UA}, containing N_{UA} elements. Such elements represent unconditional attacks to be reproduced, and are ordered in a chronological fashion, according to their starting time. Each one of them includes: i) the occurrence frequency according to which the attack is reproduced over time; and ii) the list of events that define the attack. Finally, the simulator starts N_{UA} timers, one for each attack in L_{UA}, thus scheduling their first occurrence.

Then, the Attack Simulator relies on the above mentioned data structures, and reproduces attacks as follows.

Physical attacks. When the timer associated to a physical attack expires, the simulator retrieves the corresponding attack A from the right L_{PA} list, and reproduces it on the involved node. Then, the attack A is removed from the L_{PA} list.

Conditional attacks. When the timer associated to a conditional attack expires, the simulator retrieves the corresponding attack A from the right L_{CA} list, associated to node n. From then on, node n starts to intercept packets by means of its own Local Filter, and filters them according to the packet filter specified in A. Then, for each packet that satisfies the packet filter, node n executes the list of events defined in A. The actual reproduction of conditional attacks may involve the Global Filter, as well as more than one Local Filter module.

Unconditional attacks. When the timer associated to an unconditional attack expires, the simulator retrieves the corresponding attack A from the L_{UA} list. From then on, the list of events defined in A is executed, and repeatedly performed according to the specified occurrence frequency. The Global Filter is responsible for starting the actual reproduction of unconditional attacks.

It is worth remarking that we are interested in simulating and evaluating the *effects* of attacks. That is, we assume that attacks have been entirely and successfully performed, and focus on their effects on the WSN. Unlike described in [23], we do not simulate the actual occurrence of attacks. Instead, we reproduce their final effects on the sensor network.

Finally, it is worth noting that our architectural model is general, and can be further extended. In fact, the Local Filter module is able to deal with arbitrary compositions of communication layers. This makes ASF potentially suitable for any discrete event network simulator.

D. ASF prototype implementation

We implemented a prototype of ASF for the WSNs simulator Castalia [2], based on the OMNeT++ platform [3]. Castalia considers the network as a collection of nodes, which sense values according to a given physical process, and communicate through a commonly shared wireless channel.

In the original architecture of Castalia, nodes are composed of different submodules. Sensor nodes applications interact with the physical process through a sensor manager module, and retrieve physical information from the environment.

Also, nodes are provided with a full communication stack, composed by a Routing, a MAC, and a Radio layer. Thanks to such communication modules, the application sends (receives) packets to (from) the wireless channel. Also, Castalia provides the implementations of different Routing and MAC layers.

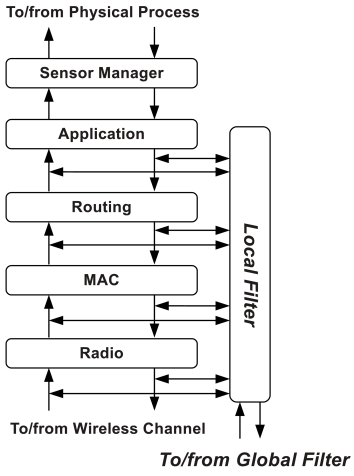

Fig. 4. Enhanced architecture of Castalia node.

Our ASF implementation for Castalia improves the original node architecture by introducing the Local Filter module. As shown in Figure 4, the Local Filter intercepts incoming and outgoing packets travelling through the communication stack, between every pair of layers. Finally, every Local Filter module is connected to the Global Filter module.

IV. Application scenario

We consider a room, whose size is 100 by 20 meters. The room is composed by three different areas, and each one of them hosts a heater, i.e. *S1*, *S2*, and *S3*. Every heater has a constant temperature of 200 °C.

In order to regularly monitor the heat level, eighteen static sensor nodes $n_1, n_2, \ldots, n_{18}$ are positioned within the room in a regular fashion. That is, they form three rows of six nodes each. The network is divided into three different node *clusters*, each one covering a different area, i.e. $\{n_1, n_2, n_7, n_8, n_{13}, n_{14}\}$, $\{n_3, n_4, n_9, n_{10}, n_{15}, n_{16}\}$, and $\{n_5, n_6, n_{11}, n_{12}, n_{17}, n_{18}\}$. Basically, each node is responsible for periodically monitoring the temperature in its cluster.

An additional node is placed in the very center of the room, and acts as *sink*. The sink node collects reports received by other nodes, and computes average temperatures over time. In particular, it computes and stores average temperatures for each one of the three clusters, as well as for the whole room.

Figure 5 depicts the room described above. The heaters are represented by squares, one for each area. The sink node is the triangle above the central heater. Finally, the eighteen black circles represent the sensor nodes deployed in the room.

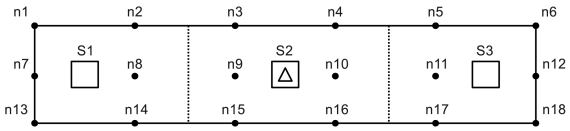

Fig. 5. Application scenario and network topology.

A. Threat model

In such a scenario, an adversary may be interested in compromising the service availability, or altering reports correctness before they are collected by the sink. We consider four possible attacks to the monitoring wireless sensor network.

Removal attack. The adversary removes a number of sensor nodes from the network. By doing so, the computation of average temperatures is clearly altered. However, this attack is relatively simple to detect, since the sink is supposed to not receive temperature reports from removed nodes anymore.

Misplace attack. The adversary moves a number of sensor nodes from their original position to a new one. By properly choosing the new positions, it is possible to alter the computation of average temperatures on the sink node, or even worse. In fact, if a temperature warning threshold is defined, an alarm can be erroneously raised, because of a dangerously fake high temperature. This attack is far more difficult to detect, since the sink assumes that all nodes' original positions are unchanged.

Reprogram attack. The adversary tampers with a number of sensor nodes, and reprograms their behavior. In particular, she can force them to erroneously report either the minimum or the maximum possible temperature value, or even a randomly generated value. In the latter case, the random value is comprised among the minimum and maximum possible value. This attack is quite hard to be detected, although comparisons with other nodes' reports may help to contrast its effectiveness.

Drop attack. The adversary forces a number of sensor nodes to drop a subset of packets, according to specified criteria. For instance, a node can be forced to drop all data packets sent to the sink by a certain group of senders. This attack is quite hard to be detected, since packet loss may be erroneously considered due to packets corruption or medium access issues.

V. Simulative analysis

In this section, we refer to the application scenario described in Section IV, with network nodes sensing temperature in their proximity one time per second. Then, we evaluate the effects of the four attacks presented in Section IV-A by using our prototype implementation of ASF for the Castalia simulator.

In our simulations, we consider the Multipath Rings routing protocol [1], the IEEE 802.15.4 MAC protocol [11], and the CC2420 radio chipset [4]. Results were obtained by means of 10 simulation runs, whose length was 600 seconds each.

The heat propagation model is based on the *Customizable Physical Process* provided by Castalia [1]. We assume that sensor nodes are able to report 0 °C and 1000 °C as minimum and maximum temperature value, respectively.

In the following, we consider the attacks described in Section IV-A. For each one of them, we provide an example of

attack configuration using *ASL* primitives presented in Section III-A, and discuss its impact on the application.

Considered attacks occur at time $t = 200$ s, and may involve three different pairs of nodes. In particular, the adversary manages to compromise either i) not a node; ii) nodes {2,3}; iii) nodes {9,10}; or iv) nodes {12,18}.

With reference to Figure 5, nodes 9 and 10 are supposed to be the most important ones, being very close to the central heater *S2* and the sink node. On the other hand, nodes 12 and 18 are supposed to be less important, being positioned at the extreme right side of the room and far from the sink node.

A. Removal attack

At time $t = 200$ s, the adversary removes either nodes {2,3}, {9,10}, or {12,18} from the network. In case nodes {2,3} are removed, the attack can be described as follows.

```
destroy(2,200);
destroy(3,200);
```

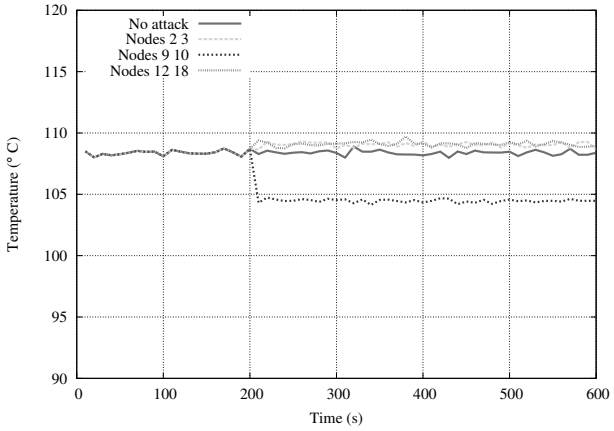

Fig. 6. Average room temperature with nodes removal.

Figure 6 shows variations of the average perceived room temperature, in case different pairs of nodes are removed. It is evident that the average value remains almost the same, and barely changes only if nodes $\{9, 10\}$ are removed.

This result is reasonable, since the Multipath Rings routing protocol introduces redundancy in network connectivity, which results to be pretty robust. Then, messages sent by a given node are likely to be successfully delivered to the sink, through some of the other nodes acting as relay.

Actually affecting temperature monitoring requires to remove either a consistent amount of nodes, or the most relevant ones. Observable effects can be seen if nodes $\{9, 10\}$ are removed, because of their proximity to the sink node, and their major contribution in the packet relaying process.

Figure 7 shows the amount of temperature reports received by the sink from each node. The graph confirms that the amount of received reports can be drastically reduced, depending on the specific pair of nodes removed from the network. Also, if a pair of nodes is removed, reports from other nodes are more likely to be delivered to the sink. This is due to a reduced number of nodes contending to access the medium.

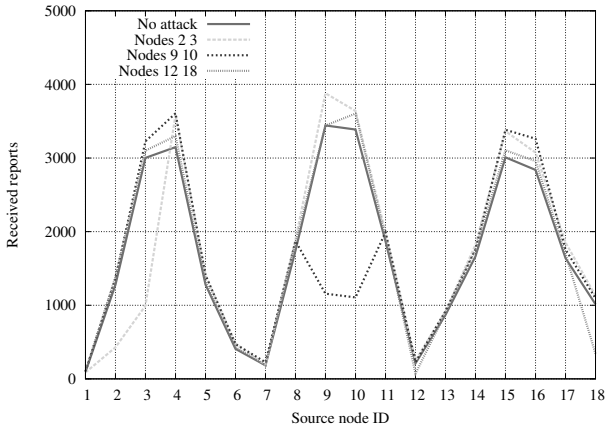

Fig. 7. Reports reception with nodes removal.

B. Misplace attack

At time $t = 200$ s, the adversary moves either nodes {2,3}, {9,10}, or {12,18} to the specified new position. More specifically: i) nodes {2,3} are moved to position (0;10;0); ii) nodes {9,10} are moved to position (100;10;0); and, finally, iii) nodes {12,18} are moved to position (50;10;0). In case nodes {2,3} are moved to position (0;10;0), the attack can be described as follows.

```
move(2,200,0,10,0);
move(3,200,0,10,0);
```

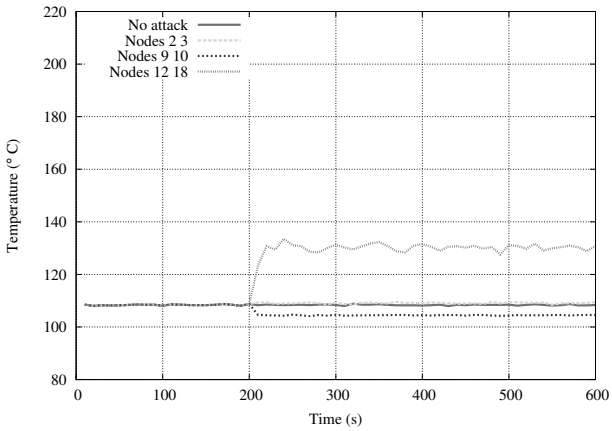

Fig. 8. Average room temperature with nodes misplacing.

Figure 8 shows variations of the average perceived room temperature, in case different pairs of nodes are misplaced. As depicted in the graph, the nodes pair {12,18} is the only one that not negligibly affects the average temperature perceived in the room. This is because such two nodes are supposed to be at the room border and distant from the sink. Then, they are moved to the room center, at the very same position of the sink and the central heater *S2*, thus altering the average perceived temperature of up 20%.

If we focus on the right cluster, the misplace attack appears to be much more effective. As shown in Figure 9, nodes

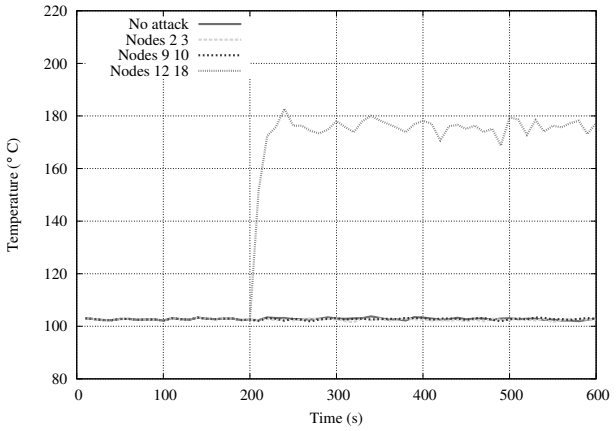

Fig. 9. Average right cluster temperature with nodes misplacing.

{12,18} are still the only ones that consistently affect the average temperature in the right cluster, if moved away from their legitimate positions. However, the average temperature in the right cluster is erroneously perceived as considerably different, i.e. up to 70% higher. Clearly, if a monitoring system took into account also single clusters, this attack might easily succeed in improperly raising an alarm.

C. Reprogram attack

At time $t = 200$ s, the adversary reprograms either nodes {9,10} or {12,18} to change the content of their own application data packets, i.e. their temperature reports, before sending them to the sink. The new content can be either the minimum, the maximum, or a random temperature value. In case nodes {9,10} replace their reported temperature with a random value, the attack can be described as follows.

```
from t = 200; nodes = "9 10" do {
 if(packet.APP.source==SELF &&
    packet.APP.type==DATA)
      change(packet,APP.payload,RAND);
}
```

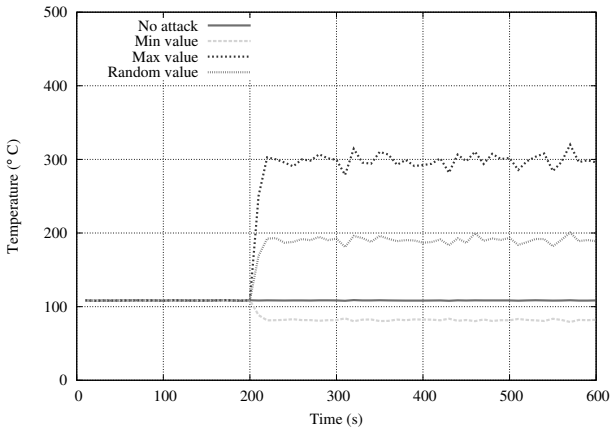

Fig. 10. Average room temperature with nodes $\{9, 10\}$ reprogrammed.

Figure 10 shows variations of the average perceived room temperature, in case nodes $\{9, 10\}$ are reprogrammed. As depicted in the graph, forcing nodes to provide a random temperature value results in erroneously perceiving the room temperature up to 100% higher. Moreover, if the nodes report the maximum possible value, the average perceived room temperature is over 200% higher. Thus, by performing this attack, it is easy to improperly raise an alarm within the room.

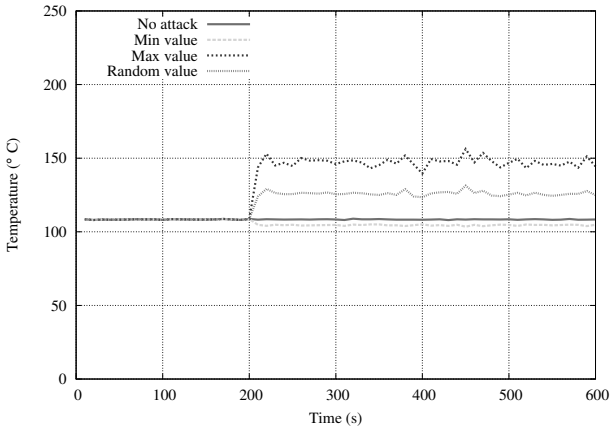

Fig. 11. Average room temperature with nodes $\{12, 18\}$ reprogrammed.

As shown in Figure 11, this attack is less severe in case nodes $\{12, 18\}$ are reprogrammed. In fact, such nodes are located at the right room border, i.e. far from the sink, and not so close to an heater (especially node 18), i.e. they are less influent in the temperature monitoring process. Thus, if nodes are forced to report the maximum possible value, the average perceived room temperature can be up to 40% higher.

D. Drop attack

At time $t = 200$ s, nodes {9,10} start to retain all MAC data packets sent by nodes {1,2,7,8,13,14}. Then, they discard such packets, rather than relaying them to the sink according to the Multipath Rings routing protocol. In case nodes {9,10} are forced to drop MAC data packets from other nodes, the attack can be described as follows. The utility function *belong()* returns *true* in case the list specified as first argument includes the value specified as second argument.

```
srcList={1,2,7,8,13,14};
from t = 200; nodes = "9 10" do {
 if(belong(srcList,packet.MAC.source) &&
    packet.MAC.type==DATA)
      drop(packet);
}
```

Even if nodes {9,10} drop MAC data packets, the amount of reports received by the sink is basically not affected. As a consequence, the average perceived temperature within the room remains the same, even if a *Drop attack* occurs. We performed more simulations, where either nodes {2,3} or {12,18} perform the attack, and obtained similar results.

This is mainly due to the adopted Multipath Rings routing protocol. As discussed for the *Removal attack* in Section V-A,

sensor nodes act as relay at the network layer, thus introducing redundancy in network connectivity. Thus, even if temperature reports from sensor nodes are dropped, they are likely to be eventually delivered to the sink. This means that, in the considered application scenario, two nodes dropping MAC data packets are not sufficient to alter network activity and application performance in a relevant way.

Also, we think that if non trivial routing mechanisms are adopted, an attack which consists only in discarding packets can hardly affect applications and performance. Instead, discarding packets may be effective as part of more complex attacks, together with other basic actions (e.g. altering packet contents, reprogramming nodes, injecting fake packets).

VI. Conclusion

We have presented ASF, our framework for simulative evaluation of attacks in WSNs. ASF provides an Attack Specification Language to specify different kinds of attacks, and an Attack Simulator to quantitatively evaluate their effects. This allows users to evaluate attacks severity, and thus define protection priorities and select appropriate countermeasures.

We have considered a realistic application scenario, and used our ASF prototype for the Castalia simulator to evaluate the effects of four different attacks. Our results show how considered attacks affect the application and network behavior. Also, they suggest to ensure reliability of communications, and provide physical protection to sensor nodes.

Our future work will focus on further improving the ASF framework, in order to specify and evaluate more complex attacks, potentially combined with one another. Also, we will consider a wider range of application scenarios, and evaluate more complex attacks and countermeasures to them.

Acknowledgment

This work has been supported by EU FP7 Network of Excellence CONET (Grant Agreement no. FP7-224053) and EU FP7 Integrated Project PLANET (Grant agreement no. FP7-257649). We would also like to thank Roberta Daidone and Alessandro Pischedda for their valuable help during the implementation phase of our work.

References

[1] "Castalia - A simulator for Wireless Sensor Networks and Body Area Networks, Version 3.2 User's Manual." [Online]. Available: {http://castalia.npc.nicta.com.au/pdfs/Castalia-UserManual.pdf}

[2] "National ICT Australia - Castalia." [Online]. Available: http://castalia.npc.nicta.com.au/

[3] "OMNeT++ Network Simulation Framework." [Online]. Available: http://www.omnetpp.org/

[4] "Texas Instruments CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver." [Online]. Available: {http://www.ti.com/lit/ds/symlink/cc2420.pdf}

[5] "TinyOS Home Page." [Online]. Available: http://www.tinyos.net/

[6] A. Becher, E. Becher, Z. Benenson and M. Dornseif, "Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks," in *Proceeding of the 3rd International Conference on Security in Pervasive Computing (SPC)*, 2006, pp. 104–118.

[7] B. Sun, K. Wu and U. Pooch, "Secure Routing Against Black-hole Attack in Mobile Ad Hoc Networks," in *International Conference on Communications and Computer Networks (CCN'02)*, November 2002.

[8] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: attacks and countermeasures," in *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, May 2003, pp. 113–127.

[9] G. Padmavathi and D. Shanmugapriya, "A Survey of Attacks, Security Mechanisms and Challenges in Wireless Sensor Networks," *International Journal of Computer Science and Information Security*, vol. 4, no. 1 & 2, pp. 117–125, August 2009.

[10] I. M. Atakli, H. Hu, Y. Chen, W. S. Ku and Z. Su, "Malicious node detection in wireless sensor networks using weighted trust evaluation," in *Proceedings of the 2008 Spring simulation multiconference*, ser. SpringSim '08. San Diego, CA, USA: Society for Computer Simulation International, 2008, pp. 836–843.

[11] *IEEE Std. 802.15.4-2006, IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specic requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs)*, Institute of Electrical and Electronics Engineers, Inc., New York, September 2006.

[12] J. Newsome, E. Shi, D. Song and A. Perrig, "The Sybil attack in sensor networks: analysis & defenses," in *Third International Symposium on Information Processing in Sensor Networks, (IPSN 2004)*, April 2004, pp. 259–268.

[13] P. Tague and R. Poovendran, "Modeling node capture attacks in wireless sensor networks," in *46th Annual Allerton Conference on Communication, Control, and Computing*, September 2008, pp. 1221–1224.

[14] R. Daidone, "Experimental evaluations of security impact on IEEE 802.15.4 networks," in *2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2011)*, June 2011, pp. 1–2.

[15] R. Daidone, G. Dini and M. Tiloca, "On experimentally evaluating the impact of security on IEEE 802.15.4 networks," in *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS 2011)*, June 2011, pp. 1–6.

[16] S. Han, E. Chang, L. Gao and T. Dillon, "Taxonomy of Attacks on Wireless Sensor Networks," in *EC2ND 2005*, Blyth, Andrew, Ed. Springer London, 2006, pp. 97–105.

[17] S. Kaplantzis, A. Shilton, N. Mani and Y. A. Sekercioglu, "Detecting Selective Forwarding Attacks in Wireless Sensor Networks using Support Vector Machines," in *Intelligent Sensors, Sensor Networks and Information, 2007 (ISSNIP 2007). 3rd International Conference on*, December 2007, pp. 335–340.

[18] T. Bonaci, L. Bushnell and R. Poovendran, "Node capture attacks in wireless sensor networks: A system theoretic approach," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, December 2010, pp. 6765–6772.

[19] T.-G. Lupu, "Main types of attacks in wireless sensor networks," in *Proceedings of the 9th WSEAS international conference on signal, speech and image processing, and 9th WSEAS international conference on Multimedia, internet & video technologies*, ser. SSIP '09/MIV'09. World Scientific and Engineering Academy and Society (WSEAS), 2009, pp. 180–185.

[20] W. Du, L. Fang and P. Ning, "LAD: Localization Anomaly Detection forWireless Sensor Networks," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, April 2005.

[21] Y. Hu, A. Perrig, D. Johnson, "Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks," in *IEEE INFOCOM 2003*, April 2003.

[22] Y-T. Wang and R. Bagrodia, "Scalable emulation of TinyOS applications in heterogeneous network scenarios," in *IEEE 6th International Conference on Mobile Adhoc and Sensor Systems (MASS 2009)*, October 2009, pp. 140–149.

[23] Y.-T. Wang and R. Bagrodia, "SenSec: A Scalable and Accurate Framework for Wireless Sensor Network Security Evaluation," in *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*, June 2011, pp. 230–239.

[24] Y. Xu, G. Chen, J. Ford and F. Makedon, "Detecting Wormhole Attacks in Wireless Sensor Networks," in *Critical Infrastructure Protection'07*, 2007, pp. 267–279.