Reliable Authentication and Anti-replay Security Protocol for Wireless Sensor Networks

Laura Gheorghe, Răzvan Rughiniş, Răzvan Deaconescu, Nicolae Ţăpuş
Politehnica University of Bucharest,
Bucharest, Romania
{laura.gheorghe, razvan.rughinis, razvan.deaconescu, ntapus}@cs.pub.ro

Abstract—**Wireless Sensor Network provide monitoring services such as environmental, military and medical monitoring. Sensor networks are often deployed in hostile environments and are vulnerable to attacks and failures. Security need to be implemented in order to prevent unauthorized access to the network and malicious attacks. The Authentication and Anti-replay Security Protocol is a combination of two lightweight mechanisms that ensure authentication, anti-replay and intrusion detection: the "Last Hash" method, and the authentication handshake. This paper introduces three reliability enhancements to the first version of the protocol: acknowledgements, re-authentication and a current hash computed with a different key to ensure integrity. Reliable AASP was implemented in TinyOS and tested using TOSSIM. Simulations indicate that Reliable AASP is able to provide a reliable authentication connection between any two communicating nodes, and it meets the critical security requirements: integrity, authentication and freshness.**

Keywords-wireless sensor networks, security, reliability, integrity

I. INTRODUCTION

A Wireless Sensor Network (WSN) consists of a large number of sensor devices characterized by reduced dimension, low cost and low power, which are able to organize themselves into a network by communicating through a wireless medium, collaborating in order to accomplish a common task [1].

WSNs provide monitoring services in different areas, such as industrial, military, public safety, automotive, agriculture, localization, seismic, medical, commercial and emergency situations. Some of the most interesting applications are detecting the enemy units during military monitoring, person locator, disaster detection, and health condition monitoring [2].

Because WSNs have the advantage of being deployable in inhospitable fields, such as battlefields, outer space and deep waters, they are highly recommended in military applications, environmental monitoring, security and surveillance, industrial process control and health care applications [3].

The network design objectives and requirements include: reduced dimension, low cost and low power, scalability, adaptability, reliability, fault-tolerance, security, self-configurability and QoS. Fault tolerance includes capacities for self-testing, self-calibrating, self-repairing and self-recovering [4].

Securing WSNs is essential when they are used in critical applications such as battlefield surveillance and homeland security. This is a challenging task because of several limitations deriving from the wireless channels, resource constraints, and hostile environments.

Because the wireless medium is open, anyone can intercept traffic and inject fake data packets if they have the radio interface configured on the same frequency band.

Traditional security mechanisms cannot be applied to sensor networks because of their severe resource constrains. Sensor nodes do not have the computational capacity to manage public key cryptography or other complex protocols. For this reason, the best choice for WSNs is to use symmetric keys, though they must be used with precaution in order to avoid performance degradation.

Another challenge regarding security in WSNs is their deployment in hostile environments and the fact that they must work unattended. They are vulnerable to physical attacks, such as tampering and node capturing.

The rest of the paper is structured as follows: Section II presents the related work, Section III contains the basic protocol design, Section IV describes the basic protocol issues, Section V presents the reliability improvements brought to the basic protocol, Section VI describes the implementation of the protocol, Section VII presents the experimental results, in Section VIII we discuss the potential problems and solutions, and Section IX presents the conclusions of the paper and some of the future work.

II. RELATED WORKS

Various security solutions were developed for WSNs, and the most important are SPINS, LEAP, TinySec, and SM [5].

SPINS is a set of security protocols that consist of two building blocks: SNEP and μTESLA [6]. Both were implemented to run on top of TinyOS [7].

SNEP is used to provide authentication, integrity, confidentiality and freshness, and μTESLA provides authenticated broadcasts [5]. Authentication and integrity is provided by Messsage Authentication Code (MAC), confidentiality through encryption, and freshness through nonce. μTESLA emulates asymmetry through the delayed disclosure of symmetric keys.

LEAP (Localized Encryption and Authentication Protocol) was designed by Zhu et al. in 2003 and is a key management protocol for WSNs [8].

LEAP was implemented as LEAP+ in TinyOS and first used on Berkley Mica2 motes [9]. LEAP uses four different types of keys in order to provide an adequate level of security to the various messages exchanged in the WSN [5].

TinySec was designed by Karlof et al. in 2004 and is the replacement for SNEP [10]. TinySec is a link layer security architecture that was included in the TinyOS release. It provides authentication and integrity, confidentiality and semantic security. Semantic security is achieved through Initialization Vector (IV) [5].

Security Manager (SM) was proposed by Heo and Hong in 2006 and is a new security method of authenticated key agreement [11]. It uses Public Key Infrastructure (PKI) and Elliptic Curve Cryptography in order to assure security [5].

The existent security solutions are very complex because they aim to meet all the major security requirements: authentication, integrity, confidentiality and freshness. We consider that confidentiality cannot be obtained without a major computational overhead that is not feasible for Wireless Sensor Networks. However, if the application is critical and requires confidentiality, a complex security solution such as SNEP must be used.

Energy consumption is the most critical problem in Wireless Sensor Networks. We aim at developing a lightweight security protocol that is focused only on authentication, integrity and anti-replay, which mitigate the most important threats to sensor networks: packet injection and packet altering. Our protocol introduces a small overhead because it only computes a hash function; therefore it is a lightweight security protocol designed to minimize the energy consumption.

III. AASP

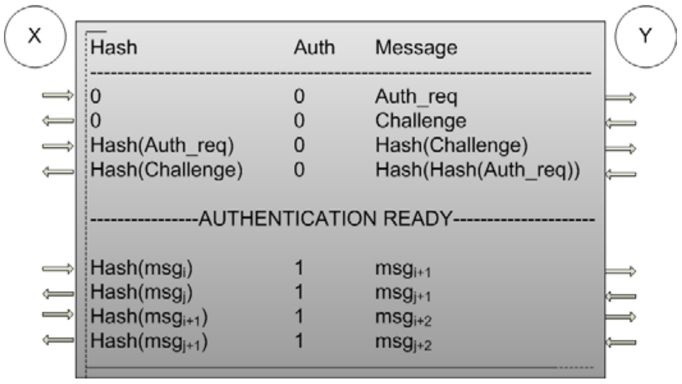
We developed a lightweight security protocol, called Authentication and Anti-replay Security Protocol (AASP), which is able to provide authentication, anti-replay and intrusion prevention [12].

The protocol uses a globally shared key to compute the Message Authentication Code (MAC) in order to provide authentication between nodes.

Anti-replay requirement is assured by the "Last MAC Method", in which the MAC of the last sent message with the same source and destination is sent along with the current message.

An authentication connection must be established between two nodes that want to communicate, using an authentication handshake, as represented in Figure 1.

AASP provides intrusion prevention because it prevents malicious nodes from communicating with the nodes inside the network. Intrusion prevention is achieved using the handshake authentication and the shared key.



Hash	Auth	Message
0	0	Auth_req
0	0	Challenge
Hash(Auth_req)	0	Hash(Challenge)
Hash(Challenge)	0	Hash(Hash(Auth_req))
---------------AUTHENTICATION READY--------------------		
Hash(msg_i)	1	msg_{i+1}
Hash(msg_j)	1	msg_{j+1}
Hash(msg_{i+1})	1	msg_{i+2}
Hash(msg_{j+1})	1	msg_{j+2}

Figure 1. AASP authentication handshake

The security protocol was implemented in TinyOS, an operating system developed especially for Wireless Sensor Networks [7]. AASP was tested using TOSSIM, which is a discrete event simulator for TinyOS sensor networks [13].

IV. AASP ISSUES

As we have stated in [12], several problems can appear in a real-life deployment of this security protocol. This paper aims at finding the most efficient solutions to those problems.

A. *Altered packets in multi-hop networks*

In a multi-hop network, the packet must be routed in order to reach the destination. However, an intermediate node could maliciously alter the payload of the packet, leaving the Hash field unmodified. This would lead to the destination accepting an altered message, which is a serious vulnerability. Our solution is to include another field in the header that will include the MAC of the current message. Therefore, an intermediate node is not able to modify the payload without knowing the secret key, and if it does, the message is dropped at the destination.

B. *Packet loss*

The problem that has the highest probability to appear is the loss of packets. If a single packet is lost, the hash becomes de-synchronized and the subsequent packets are dropped at the destination with an "Incorrect Hash" alert. A similar problem appears in the SPINS suite of security protocols, where the anti-replay protection is provided by incrementing a counter for every sent and received packet. Packet loss causes the disruption of the counter synchronization. A solution would be for the destination to acknowledge packets as they are received and the source to wait for an acknowledgement before sending the next packet. We will evaluate the impact of such a solution as regards energy consumption.

C. *Re-authentication after reboot*

Another problem may appear in the eventuality that one node, designated as X, reboots during an authentication connection between two nodes (X and Y). After rebooting, X could try to re-establish the connection, but Y would reject all packets with the message: "Node already authenticated tries to re-authenticate". This would happen because Y was not announced in any way that node X closed the connection, and continued to act as if it had an authentication connection with it. In addition, we also need a method of terminating an idle connection. The solution for both problems would be to permit the connection to expire after a period of time in which no messages are sent. After the old connection is closed by both sides, a new connection can be established. This method would introduce a certain delay in communication, and we must evaluate the impact of this solution taking into account the probability for a sensor to reboot unexpectedly.

Another solution would be to permit the operation of re-authentication, in which one node can initiate an authentication handshake even if the other node knows it has a connection already established. However, this practice is dangerous because any attacker can interpose between two nodes and tear down their connection just by sending an "Authentication Request".

V. RELIABLE AASP DESIGN

Four main levels of fault tolerance can be implemented for WSNs that correspond to the following layers: hardware and software, network communication and application layers [1]. We are interested in the network communication layer fault tolerance.

In the following sections, we will present three methods that have been integrated into AASP in order to enhance the reliability of the security protocol and the network communication.

A. *Packet loss*

As we have observed in simulations, packet loss is a serious problem. It desynchronizes the anti-replay mechanism because the destination expects another value of the last hash. Subsequent packets are dropped by the destination when a single packet is lost on the way.

The standard procedure when dealing with packet loss relies on the use of acknowledgements. When a packet is received, the destination node is responsible for sending back a packet containing the sequence number of the acknowledged message, and an acknowledgement flag. This way, the source node knows when it is safe to send the next packet.

The source node waits for the acknowledgement for a predefined period of time, after which it resends the packet that was not acknowledged. Not receiving the acknowledgement before timeout, could indicate that either the connection with the destination was closed, or the acknowledgement packet was lost on the way. The connection could be closed when the destination is dead or when it had restarted itself. The source node is not able to differentiate between these two situations, and will treat them in the same way, by resending the lost packet.

During the handshake process, the packets are not acknowledged by separate packets, but because an actual conversation is taking place, the acknowledgements are piggybacked in the reply messages. This cannot be done after the authentication connection is established because, in most cases, the conversation consists in packets sent from the child to the parent in the network hierarchy. A hierarchy is formed so that the collected information would reach the base station. In this case, the parent would have to reply with separate acknowledgement packets.

This method provides reliability to AASP, because it handles the loss of packets through the traditional method of acknowledgements.

B. *Re-authentication*

Re-authentication is the solution for both desynchronized and closed connections. Desynchronization could occur when acknowledgements are not used and packets are lost, or if they are used and the information about the acknowledged packets is corrupted on the source or destination node. As we have previously stated, a connection is considered closed when either node is dead or has restarted.

The connection must be terminated after a specific amount of time in which no message is received from the node on the other side of the connection.

After each received message, a timer is set to fire once after a specific amount of time that is to be determined experimentally. If another message is received during this period of time, the timer is reset. Otherwise, the authentication connection will be terminated by erasing all authentication information regarding the connection with that specific node for which the timer has fired. In addition, a connection termination message will be sent to that node, in order to announce it of the connection tear down. The neighbor could be alive, in the situation of non synchronization or restarted node. In the first situation, the neighbor node will receive the termination message and erase all information about that connection. In the case of a restarted node, all information is already to its default settings and nothing should be done.

After authentication data has been erased at both nodes, a new authentication connection can be established if either node initiates an authentication handshake.

C. *Altered packets*

We can prevent messages from being altered by sending the Message Authentication Code (MAC) of the current message. The destination will compute the hash value of the received message and compare it with the MAC in the header. If the values are different, the packet will be dropped, and an "Altered packet received" message will be generated.

If we want to be sure that the attacker cannot replay older packets, we must have two secret keys in the network: one for the last MAC, and one for the current MAC. This way, the attacker cannot use the current MAC from an intercepted packet as the last MAC in a malicious packet.

We evaluate that the overhead introduced by keeping two secret keys instead of one is insignificant comparable to the advantage of messages being protected from alteration in multi-hop networks.

VI. IMPLEMENTING RELIABLE AASP

A. Packet loss

Acknowledgements are implemented in AASP using a new field in the authentication header, which contains the value "1" if the packet is an acknowledgement and the value "0" if it is a data packet. We call this field the ack flag.

During the authentication handshake, all packets except for the "Authentication Request" have the ack flag set, because each reply acknowledges the previous message exchanged between the two nodes.

After the authentication connection has been established, because a two-way conversation is not expected, acknowledgements must be sent as stand alone packets. These packets contain the sequence number of the packet that is acknowledged, placed in the payload field, and the ack flag that is set. The sequence number will be represented as a new field in the AASP packets. The sequence values start from value "1" after the authentication connection is established.

Each node stores two values for every established connection: the last received acknowledge (last_recv_ack) and the last sent acknowledge (last_sent_ack). After sending an ack, the value of last_sent_ack is updated to the value of the sequence number of the acknowledged packet. When receiving an ack, the value of last_recv_ack is updated to the sequence number received in the payload.

In a hierarchical network topology, when a child sends a message to its parent with a sequence number of x, it starts a timer that expires after a configured period of time. During this period, if the ack is received, the timer is stopped, and the packet with sequence x+1 is sent. However, if no ack is received, the packet with sequence x is resent.

This method has been implemented in the Authentication layer, by maintaining two arrays of integers, representing the last_sent_ack and last_recv_ack for each established connection. For example, last_recv_ack[i] is the sequence number of the last acknowledged packet sent to the neighbor with the identifier i.

An array of timer interfaces is used through parameterized interfaces, an important feature of nesC language. The timer is set to fire once by using *Timer.startOneShot* command and by specifying the period of time as an argument. In the *Timer.fired* event, the packet must be resent to the destination, therefore the packet must be stored locally until an acknowledgement is received. However, if an acknowledgement is received using *Receive.receive* event, the timer for that specific destination is stopped. The next message can now be sent by the main application. The main application execution must be delayed until the next message can be sent.

B. Re-authentication

Re-authentication is possible only after a previous connection is terminated, or both sensor nodes are restarted. The state of a connection is maintained at the Authentication layer by three arrays: auth, req and rd. For a neighbor i, auth[i] is "1" if an authentication connection has been established with that neighbor and "0" if not, req[i] is "1" if an "Authentication Request" has been sent to that neighbor and "0" if not, and rd[i] keeps the value of the challenge used for that neighbor.

At the MAC (Message Authentication Code) layer, the state is kept using two arrays of hashes, both called last_hash, where last_hash[i] is the hash of the last message sent to or received from that neighbor. All this authentication data should be erased during connection termination.

An array of timer interfaces is maintained through parameterized interfaces, as with acknowledge timers. When a packet is received at the Authentication layer, the timer associated with the source node i, is set to fire once after a configured period of time. In the *Timer.fired* event, the information in Authentication layer (auth[i], req[i] and rd[i]) is set to "0". A message is sent to the sensor node i in which auth field is set to a specific termination code called TERM. The MAC layer recognizes the code and erases all authentication data from its level. If the destination is alive and has not restarted, it will reset all authentication data from MAC and Authentication layer. If the destination has restarted, it will ignore the message.

The authentication handshake will be further initiated by the node that transmits data, in most situations, the child in the hierarchy.

C. Altered packets

The method was implemented by adding a new field in the authentication header containing the MAC of the current message that is computed using a different key.

The hash of the payload is computed and put in the current MAC field of the packet, in MacLayerSenderP component, before sending the packet.

In MacLayerReceiverP component, the first operation when receiving a packet is to check if the current MAC contained in the received packet is equal to the hash of the payload. If not, the packet is dropped at this level, or sent to the Authentication Layer with an error code in Auth field. We reserved the code 3 for "Altered packet". In AuthenticationLayerC component, if the Auth field is equal to 3, the packet is dropped, and an "Altered packet received" message is generated.

VII. TESTING RELIABLE AASP

The protocol has been implemented in TinyOS; therefore TOSSIM is the best solution for simulating AASP [13].

We use a section of a real topology to test the first scenario, in which node 3 sends messages to node 1. This scenario is used to test the functionality of acknowledgements, connection termination and re-authentication. Therefore, we did not include the hash of the current message in the simulation output.

Node 3 and 1 communicate using Reliable AASP; the first one sends data packets and the later one replies with acknowledgements. When the packets are lost, no acknowledgement is received and the packet is retransmitted.

When no packet is received by node 3 in the idle period of time, the connection between the nodes must be terminated.

We consider that the authentication handshake takes place and ends successfully with an established authentication connection. After that, in Figure 2 we can observe that node 3 starts sending packets to node 1 periodically. Packets are sent with auth field equal to "1" and ack field equal to "0". The seq field contains the sequence number of the packet. Node 3 responds with an acknowledge packet that has the ack field set to "1" and the payload containing the sequence number of the acknowledged packet. The figure presents two packets that are sent, received and acknowledged.

```
(3): AuthLayer: Packet sent [seq=1 hash=4442 auth=1 ack=0 (3->1)]
(1): AuthLayer: Packet received [seq=1 hash=4442 auth=1 ack=0 (3->1)]
(1): AuthLayer: Packet sent [msg=1 hash=24000 auth=1 ack=1 (1->3)]
(3): AuthLayer: Packet received [msg=1 hash=24000 auth=1 ack=1 (1->3)]
(3): AuthLayer: Packet sent [seq=2 hash=123 auth=1 ack=0 (3->1)]
(1): AuthLayer: Packet received [seq=2 hash=123 auth=1 ack=0 (3->1)]
(1): AuthLayer: Packet sent [msg=2 hash=123 auth=1 ack=1 (1->3)]
(3): AuthLayer: Packet received [msg=2 hash=123 auth=1 ack=1 (1->3)]
```

Figure 2. Normal flow of packets

In Figure 3, the message with sequence number equal to "7" does not reach its destination. The timer set on node 3 expires after the ack period and the node resends the message. After the idle period, the connection timer at the destination expires and the connection is terminated. Node 1 also sends a reset message containing a connection termination code, announcing the destination about the connection tear down. The code used in this implementation is "4" and is placed in the auth field. In this moment both nodes have the authentication data erased from both AASP layers.

```
(3): AuthLayer: Packet sent [seq=7 hash=97 auth=1 ack=0 (3->1)]
(3): AuthLayer: Timeout ack node 1
(3): AuthLayer: Packet sent [seq=7 hash=97 auth=1 ack=0 (3->1)]
(3): AuthLayer: Timeout ack node 1
(3): AuthLayer: Packet sent [seq=7 hash=97 auth=1 ack=0 (3->1)]
(3): AuthLayer: Timeout ack node 1
(1): AuthLayer: Timeout connection node 3
(1): AuthLayer: Reset packet sent [seq=0 hash=0 auth=4 ack=0 (1->3)]
(3): AuthLayer: Packet received [seq=0 hash=0 auth=4 ack=0 (1->3)]
```

Figure 3. Packet loss causing connection timeout

In Figure 4, we can observe the authentication handshake being re-initiated by node 3 and terminating with another authentication connection established.

```
(3): AuthLayer: Packet sent seq=124 hash=0 auth=0.
[..]
(1): AuthLayer: Managed to authenticate myself to node 3
(3): AuthLayer: Managed to authenticate myself to node 1
(3): AuthLayer: Packet sent [seq=7 hash=4442 auth=1 ack=0 (3->1)]
(1): AuthLayer: Packet received [seq=7 hash=4442 auth=1 ack=0 (3->1)]
(1): AuthLayer: Packet sent [seq=7 hash=24000 auth=1 ack=1 (1->3)]
(3): AuthLayer: Packet received [seq=7 hash=24000 auth=1 ack=1 (1->3)]
```

Figure 4. Re-authentication

The packet with sequence number "7" is resent and received successfully at node 1 and acknowledged. The subsequent packets are treated in the same way.

In the second scenario, we test the ability of Reliable AASP to provide integrity, and to detect that the packet has been altered during the routing process by a malicious node, or by errors during transmission.

We use a section of a multi-hop network topology: node 3 wants to send a data packet to node 1, but cannot reach it directly, only through node 2. Therefore it will use node 2 to route data packets to the destination node. In this scenario, node 2 is a malicious node programmed by an attacker to modify the payload of the packets that it must route to destination.

The previous version of the protocol did not provide packet integrity. Therefore packets could have been modified by an intermediate node, while the destination would not have noticed. However, it would have dropped packets starting with the next packet even if they would not have been altered, only because the altered packet would desynchronize the connection.

In the current version of Reliable AASP, the packet includes the hash of the current message, called current hash, which is computed using a secret key different than the one that is used to compute the hash of the last message. This way, even if the intermediate node changes the payload, the packet will be dropped at the destination because the current hash contained in the packet would not match the hash computed at the destination. This does not desynchronize the connection, because the source node waits for an acknowledgement before sending the next message. If the message is dropped by the destination, the acknowledgement is not received, and the message is retransmitted.

In Figure 5, we consider that node 2 does not alter the message. Therefore, we can observe the normal flow of packets. We omitted the authentication handshake process, and the information about the last hash and auth fields.

```
(3): AuthLayer: Packet sent [msg=222 chash=64327  seq=1 ack=0 (3->1)]
(2): RoutingLayer: Packet received [msg=222 chash=64327  seq=1 ack=0 (3->1)]
(2): RoutingLayer: Packet sent [msg=222 chash=64327  seq=1 ack=0 (3->1)]
(1): AuthLayer: Packet received [msg=222 chash=64327  seq=1 ack=0 (3->1)]
```

Figure 5. Non-malicious routing node

The data packet is routed by node 2 and reaches node 1 without being altered. Node 1 computes the hash of the payload using the second secret key, and it verifies if the value is equal to the current hash found in the packet. The values are equal, therefore node 1 generates an acknowledgement packet and sends it to node 3 through node 2.

In Figure 6, we consider that node 2 alters the message received from node 3, and then routes it to node 1. Node 2 cannot re-compute the current hash, because it does not have the secret key. Therefore, it sends the packet with the altered payload and the old current hash. The altered data packet reaches node 1, and is inspected by the MAC layer, which detects that the computed hash is different from the one contained in the packet send it to the Authentication layer with an error code of "3" in the auth field of the packet. The Authentication layer receives the packet, generates an "Altered packet" message and drops the packet. It does not send any acknowledgements, and after the ack period has passed on node 3, it resends the data packet.

```
(3): AuthLayer: Packet sent [msg=222 chash=64327  seq=1 ack=0 (3->1)]
(2): RoutingLayer: Packet received [msg=222 chash=64327  seq=1 ack=0 (3->1)]
(2): RoutingLayer: Packet sent [msg=999 chash=64327  seq=1 ack=0 (3->1)]
(1): MacLayer: Packet received [msg=999 chash=64327  seq=1 auth=1 ack=0  (3->1)]
(1): AuthLayer: Packet received [msg=999 chash=64327  seq=1 auth=3 ack=0  (3->1)]
(1): AuthLayer: Altered packet received. Packet dropped.
(3): AuthLayer: Packet sent [msg=222 chash=64327  seq=1 ack=0 (3->1)]
[…]
```

Figure 6. Malicious routing node

However, if the attacker continues to alter the data packets, it will cause the connection to terminate, because the correct data packet would not be received by the destination. This is considered to be a Denial of Service attack.

VIII. Discussion

As we have stated in the previous section, a node that acts as a router between source and destination nodes can cause a Denial of Service attack by altering all messages that must be routed. The source and destination node would wait the idle time period and close the connection. The best solution would be to modify the routing protocol in order to choose another intermediate node to route the data packets to destination, if the current one does not deliver the packets correctly. Another solution would be to send negative acknowledgements to the source of the data packets in order to reduce the waiting time of the source node before choosing another route to destination. A serious problem would appear if the malicious node is the only way to the destination, meaning that it is the only node that can reach the destination. Therefore, even if we consider that the source and destination are at least two hops away, if the last hop is the only one that can reach the destination, changing the route would not exclude that node, therefore, the communication with that specific destination will be irremediably lost.

Adequate values must be determined experimentally for acknowledge and idle periods. Both values must be computed for the specific deployed network and application. We consider that some requirements must be satisfied in order to find the appropriate values.

In WSNs, we have two types of data messages: periodic and triggered. The periodic messages contain data that is periodically collected from the environment and sent to the base station. The triggered messages are generated in emergency situations and should reach the base station as soon as possible. The idle period of AASP should be greater than the period of data collecting. Otherwise, before every data collecting moment, the connection would expire and another connection must be established introducing overhead because of the authentication handshake. Because triggered messages can be generated anytime, it is impossible to configure the idle period using predictions about these messages.

The acknowledge time period should be greater than the two-way message exchange time between two nodes multiplied by the maximum number of hops between two nodes in the sensor network. This way, if an authentication connection is established between two nodes that have the maximum number of hops between them, the acknowledgement should have time to reach the source of the data packets. Otherwise, the source would send the data packet, and the acknowledgement time period would expire before the acknowledgement would arrive and the data message will be unnecessary retransmitted. The two-way message exchange time period depends on the network traffic. This value can only be determined experimentally after the deployment of the network and application.

Therefore, idle time period depends on the application and its period of collecting and transmitting data. The acknowledge time period depends on the actual topology and the network traffic.

A problem is introduced by the high quantity of acknowledgements required in order to detect packet loss. In the actual implementation, the number of acknowledgements is equal to the number of data packets. However, other methods exist in which an ack packet is used to acknowledge a set of data packets instead of one single packet, but they would require a large number of messages to be stored at the source node before they are acknowledged.

Another problem regarding acknowledgements is that ack packets can also be lost. This would determine the source node to send the packet again. The duplicate would reach the destination and would be dropped because it would have the same sequence number. Duplicate packets are undesirable because they waste bandwidth and consume energy.

Nodes are still vulnerable to flooding with “Authentication requests”, but this attack can be detected using Storm Control Mechanism [14].

IX. Conclusion

Wireless Sensor Networks provide monitoring services in critical domains such as military, security and medical, a lightweight security protocol has to be used in order to secure the communication within the sensor network.

AASP is a lightweight security protocol that was designed to provide security features such as authentication, freshness and intrusion detection. In this paper, we present three reliability enhancements to AASP: acknowledgements, re-authentication and integrity hash.

Acknowledgements are used to detect the loss of packets. When a node sends a data packet, it sets a timer and waits for an acknowledgement for that data packet. If the ack does not arrive until the timer has expired, the packet is resent.

Re-authentication is needed in various situations, from desynchronized connections to restarted nodes. To proceed with the re-authentication, the previous connection must be terminated, otherwise all packets are blocked. The connection is terminated after a specific period of time in which no packet is received by the node.

The initial protocol version did not provide the integrity of the current message. The enhanced protocol version contains an integrity hash placed in the packets. This hash is a Message Authentication Code computed using a second secret key. The hash is recomputed at the destination and compared with the one received in the packet. This way, altered packets can be dropped.

Reliable AASP has been implemented in TinyOS by modifying the previous version of AASP. The protocol was tested in TOSSIM, a simulator for TinyOS applications.

Further work will consist in testing the performance of the protocol in terms of energy and bandwidth consumption. The number of messages exchanged by sensor nodes has doubled because of the acknowledgements. However, acknowledgement packets are relatively small compared to the data packets, and are necessary in order to detect the loss of packets.

Also as a future work, we wish to test our protocol on a real Wireless Sensor Network and compare our simulation results to the ones obtained on the physically deployed sensor network.

REFERENCES

[1] J. Zheng and A. Jamalipour, “Wireless Sensor Networks A Networking Perspective”, John Wiley & Sons, 2009.

[2] C.F. García-Hernández, P.H. Ibargüengoytia-González, J. García-Hernández, and J.A. Pérez-Díaz, “Wireless Sensor Networks and Applications: a Survey”, IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.3, March 2007, pp. 264-273.

[3] D. Trossen and D. Pavel, “Sensor networks, wearable computing, and healthcare Applications”, IEEE Pervasive Computing, vol. 6, no. 2, Apr-June 2007, pp.58-61.

[4] F. Koushanfar, M. Potkonjak, and A. Sangiovanni-Vincentelli, “Fault-tolerance techniques for ad hoc sensor networks”, Proceedings of IEEE Sensors, vol. 2, June 2002, pp. 1491-1496.

[5] D. Boyle and T. Newe, “Securing Wireless Sensor Networks: Security Arhitectures”, Journal of Networks, Vol. 3, No. 1, January 208, pp. 65-77.

[6] A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, and D. Culler, “SPINS: Security Protocols for Sensor Networks”, Wireless Networks, 2002, 8(5), pp. 521-534.

[7] P Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, “TinyOS: An Operating System for Sensor Networks”, Ambient Intelligence, 2005, pp. 115-148.

[8] S. Zhu, S. Setia, and S. Jajodia, “LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks”, CCS ’03, Washington D.C., USA, 27 – 31 October 2003, New York, USA: ACM Press, pp. 62-72.

[9] S. Zhu, S. Setia, and S. Jajodia, “LEAP+: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks”, ACM Transactions on Sensor Networks TOSN, 2006, 2(4), pp. 500-528.

[10] C. Karlof, N. Sastry, and D. Wagner, “TinySec: A Link Layer Security Architecture for Wireless Sensor Networks”, Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, Baltimore, MD, USA, 03 – 05 November 2004, New York, NY, USA: ACM Press, pp. 162 – 175.

[11] J. Heo and C.S. Hong, “Efficient and Authenticated Key Agreement Mechanism in Low-Rate WPAN Environment”, International Symposium on Wireless Pervasive Computing 2006, Phuket, Thailand 16 – 18 January 2006, IEEE 2006, pp. 1-5.

[12] L. Gheorghe, R. Rughiniş, R. Deaconescu, and N. Ţăpuş, “Authentication and Anti-replay Security Protocol for Wireless Sensor Networks”, The Fifth International Conference on Systems and Networks Communications, August 22-27, 2010, pp 7-13.

[13] P. Levis, N. Lee, M. Welsh, and D. Culler, “TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications”, In SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems, 2003, pp. 126-137.

[14] R. Rughiniş and L. Gheorghe, “Storm Control Mechanism for Wireless Sensor Networks”, 9th RoEduNet IEEE International Conference, June 24-26, 2010, pp. 430-435.