June 2007

Aalborg University

Department of Electronic Systems
Networking Security Group

Fredrik Bajers Vej 7 DK-9220 Aalborg Ø Telephone +45 96 35 87 00

Title: Self-organizing protocol for reliability and security in wireless sensor networks

Theme: Network Planning and Management

Period: February 1st, 2007 – June 20th, 2007

Project group: 07gr1084

Group members:

Fernando García Diez

Celia de Dios Velasco

Supervisors:

Neeli R. Prasad
(Aalborg University)

Ana García Armada
(Universidad Carlos III de Madrid)

Abstract

During this project we will propose a new security protocol with reliability mechanisms. This is designed to be used in self-organizing wireless sensor networks (WSN), taking into account the strong limitations that the sensor nodes have, mainly consisting in energy usage and limited memory storage constraints.
The proposed protocol is described that provides security mechanisms in the communication from node to node. Everything is managed distributedly by the sensor nodes without depending on a central base station, providing more flexibility and scalability.

Main report: 68 pages

Appendix: 4 pages

Finished: June 6th, 2007

Preface

This report has been written as a project for the 10th semester by the group 1084 during the second semester of 2007. It is primarily addressed to students and staff of the Department of Electronic Systems at Aalborg University, and anyone interested in the security and reliability of the wireless sensor networks.

The report is divided in two parts, main report and appendix. The main report is divided into the next several chapters, where primary concepts concerning the project are explained:

- Chapter 1 gives an introduction to the reader and sets the framework for project focus.
- Chapter 2 presents an overview of the background theory related to the topic of the project. It describes the wireless network models that will be used in the rest of the project.
- Chapter 3 introduces a more detailed analysis of the project describing the proposed protocol and explaining the considerations of the project.
- Chapter 4 presents the simulator used to test the proposed protocol. It also shows the results obtained in the simulations.
- Chapter 5 summarizes the work done in the project and takes conclusions from the results obtained.
- Chapter 6 exposes some possible future work.
- Chapter 7 provides the references to literature used in this project.

The appendix part includes the implemented simulator guide and the datasheet from the MICA2 mote.

A CD-ROM is enclosed. It contains the report in PDF-file format, the documentation used, as well as the source code in C of the simulator used to evaluate the proposed protocol.

We would like to thank our supervisor, Neeli R. Prasad, for all the help given to us during the project. Also to Jose Gutierrez Lopez, from the department of electronic systems, for all the guidance through the elaboration of the project. We want to mention also Jette Damkjær, secretary of the department, for all her help with the administrative issues, as well as all the people from the Luna Kollegium.

Fernando would like to thank to all his family, specially parents and sisters, and to Ana, for all of their love and support during this semester that I have not been with them. Also to Celia for bearing with me during these months and to all the friends from the Carlos III university, because we have passed through all the studies together and learned a lot of things that were used in this project.

Celia would like to thank to Fernando, for all his help and patience during these months in the project. Also to Claudia for her continuous support since she arrived to Aalborg. Celia dedicates this project to her parents and to her sister for all their encouragement and support, because they are always there no matter the distance.

Aalborg University, June 6th, 2007

Fernando Garcia Diez

Celia de Dios Velasco

Index

List of tables

List of figures

1 Introduction

A Wireless Sensor Network consists in several autonomous sensor nodes, generally in large numbers, distributed over a broad area, taking measurements of real world variables and in most of the cases sending this measured data to a central base station where all of these measurements are organized so they can be accessed easily. It is then a way of getting the networks world in contact with the real world.

The closest ”ancestors” of WSNs are the ad hoc networks, but they have very different characteristics. WSNs are unlike adhoc networks in the sense that WSNs are very limited in resources, they are much more densely deployed, and they are prone to node failures. Also the number of nodes in WSNs is several orders higher than that of ad hoc networks, and network topology is constantly changing. [1].

These networks are formed by hundreds of nodes deployed very often in areas with difficult access, and that makes it nearly impossible to access to each individual node to replace its battery, thus, individual nodes must be small devices with very strict energy control in order to maximize the network lifetime. The main energy-consumer in this nodes is the radio-transmission of data to the nodes' neighbors, and this fact together with that there are hundreds of nodes makes centralized algorithms not very suitable for this kind of applications. Instead, it would be much more preferable to look for a decentralized option where every node's configuration would depend just on the information collected by itself.

Self-organization in this context would be the capability of the whole network to configure itself without relying on a central station or depending from a network manager that would connect to every node in the network to change its settings. On the contrary, every node should be able to collect information from its neighbors to configure itself in a way that can benefit not only the node selfish interests, but that can also optimize the performance of the network as a whole unit.

Concerning reliability and security in WSNs, this requirements can be very application-dependent. In some applications reliability may not be of any importance, like those used to monitor the temperature, where a single packet drop will not affect seriously the performance of the application, but in other applications like those used in emergency situations, it is very important that all nodes are sharing the same information, that the important event messages are received by the sink, and that the sink can ask for particular information about an area of interest without any message loss. It is in this kind of situations where our work will be focused on.

1.1 Thesis motivation

In the last years the interest about transmitting information from mobile nodes in wireless environments, and in particular about Wireless Sensor Networks (WSNs), has been growing as new applications for this kind of networks appear and they imply new design challenges.

There is a need to monitor environments where secure and reliable communications are necessary, for example in hospital or airport environments will require that the

information transmission is reliable and secure. If for any reason one of the nodes stopped working, the information should still reach the sink reliably and securely.

Although several protocols have been proposed for securing communications over WSNs ([2], [3]), it is still a challenge to provide a system that can guarantee security at the same time that it provides reliability in the delivery of the messages. The particular environment in which WSNs work, is one of the most difficult medias to provide security. Because it is a wireless channel, anyone can have access to the transmitted packets. Also due to the strong constraints that the sensor nodes have (memory storage, computing capacity, limited battery), the existing security protocols are too heavy to be used this context.

We need a protocol that is lightweight enough to be used in WSNs using very few resources, but that at the same time provides security measures strong enough to prevent the propagation of the sensor readings to undesired nodes.

A reliable communication is a key requirement before any security protocol is implemented. It is necessary to guarantee that every message from the security protocol will reach its destination if we do not want to waste a lot of energy retransmitting messages at higher levels, where the packet size is bigger and the cost of retransmitting a packet is higher. If the messages are important enough to be transmitted using a security protocol, it has no sense not to guarantee that they can reach reliably the other end of the communication.

Some of the protocols that address security on the node to node communication use several keys for every sensor node, which introduces more storage and computation capacity needs. They also assume that the nodes neighbors will not change, and that their position is static. However, in a lot of applications the sensor nodes can be moving around the sensing field, and there is a need to support this kind of mobility in which the nodes neighbors vary through time as they move.

In the context of WSNs, self-organization is a very important requirement. By making the protocol distributed instead of depending on a central base station, we can achieve better scalability, and less control messages from the central station to the nodes are needed.

When transmitting secure information, there always exist the risk of an attacker breaking into the system. To prevent this, the concept of semantic security was introduced, guaranteeing that all of the messages will be encrypted in a different way, even if they are the same plain text. The problem with the semantic security is that normally it needs some sinchronization mechanisms between the nodes, and storing several different secret keys.

Due to the constraints of WSN's, we consider it very important to reuse certain variables or functional blocks in order to reduce the memory usage and the message overhead, and to optimize the protocols as much as possible to make them lightweight. Using few memory is important if we want that the protocol can be implemented in several models of real sensor nodes. The proposed protocol should be realistic, meaning that it must cope with the limited resources present in the current sensor nodes in the market.

1.2 Project scope

In the context of wireless sensor networks, self-organization is a necessary requirement for every aspect of the protocols design. Thus, we will focus on studying always distributed mechanisms in which the nodes can configure themselves depending on the information received from their neighbors.

The project will focus on designing a protocol that provides security issues (see section 3.1), and that is built over a reliable connection.

We will focus on security inside the wireless sensor network, instead of on the communication from the network to other different networks, and more specifically focusing on the node to node communication.

In the context of WSN's, there is a difference between messages going from the nodes to the sink and messages going from the sink to the nodes. The main kind of messages are those going “upstream” (nodes to sink direction), and their importance is crucial because in these packets is carried all the information about the sensor readings. Every node broadcasts the data through other nodes to reach the sink node, and it is necessary to guarantee that the messages are being delivered only to the correct nodes and not to others that, maliciously or not, could be overhearing the network messages. For this reason, we will focus on the security and reliability of upstream messages, because there is stronger necessity for this aspects in this direction of the communication.

In particular, we will propose a protocol for secure communications between nodes with an authorization and authentication scheme that will check that the sensor nodes that are trying to send data are in fact who they are pretending to be, and that they have permission to send data. At the same time, we will build a mechanism that provides reliability and makes that the receiver nodes can ask for retransmissions of the missing packets.

For this kind of communication, we will focus on hop-by-hop mechanisms, as they perform better both in security and reliability protocols, as explained in [4], [5], and [6]. Due to this, and to the inherent multicast properties of the wireless medium, we will address the problem of communications between different nodes from the network.

This protocol should also solve the problem of interfering sensor networks, where they exist several different overlapping networks, and every node will need to identify which sensors should it listen to, and which others belong to a different network. In our protocol we will try to identify every sensor that wants to send or receive data, to make sure that it belongs to the network. Also, data authentication will allow a receiver to verify that the data was sent by the claimed sender.

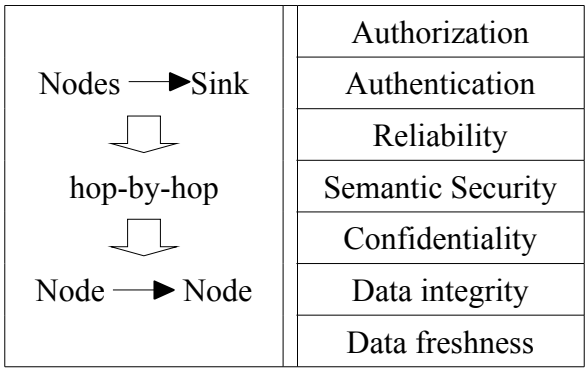


Table 1.1: Summary of the Project Scope

2 State of the art and related work

2.1 Role of Security in WSNs

Wireless communications are difficult to protect, as they work by nature over a broadcast medium. In this kind of wireless environment there is the possibility that malicious users try to interfere the data transmitted through the network.

Network security is a very wide concept involving a lot of issues. In the case of WSNs, the sensors have very limited resources (e.g. memory, power), and due to these constraints the current security protocols do not consider all the security properties required by WSNs.

In WSNs the main security requirements are:

- AAA concept: authorization, authentication and accounting.
    - Authentication: it involves proving that the node is who it is pretending to be. Wikipedia defines it as: "Authentication refers to the confirmation that a user who is requesting services is a valid user of the network services requested. Authentication is accomplished via the presentation of an identity and credentials".

      There are several authentication methods:

        - MAC (Message Authentication Code). It is a code generated with an algorithm, used to authenticate the sender of a message. The procedure to authenticate using MAC is the following:
            - The two parties have to share a private key, called "k".
            - The sender generates a MAC code using both the secret key and the content (or a part of it) of the message that needs authentication, called "m":

              $$MAC=f(k,m)$$

            - The MAC and message are sent together to the receiver. The receiver can use the received data and the private key that it already knows to calculate its own MAC and compare it with the MAC received from the sender.
            - If the two MACs do not match the message is not authenticated by the receiver, as it could consider that it was sent by a malicious user.
        - Hash Function. It is used to generate a summary of a big amount of data. Its input parameter is a message which can be of any length. This summary must have the property of guaranteeing that it is impossible to rebuild the plain text from it.
        - Symmetric key or Asymmetric key algorithms. An encrypted message with symmetric o asymmetric mechanisms can also be used to authenticate messages. The authentication is implicit in

the encryption, because the receiver node knows that only the sender can encrypt the message using that secret key.

- Authorization: it involves proving that the node has enough privileges to access to some network services. Wikipedia defines it as: “Authorization refers to the granting of specific types of service (including "no service") to a user, based on their authentication, what services they are requesting, and the current system state. [...] Authorization determines the nature of the service which is granted to a user”. In the context of sensor networks, authentication and authorization are usually managed together because the authorizacion is the following step after the authentication of the node in the network.
- Accounting: it involves keeping track of what users do while they are using the network. Wikipedia defines it as: “Accounting refers to the tracking of the consumption of network resources by users. This information may be used for management, planning, billing, or other purposes”.

- Availability: we should ensure that introducing additional security mechanisms will not limit network performance. Security mechanisms will need additional computing and communication resources, and could also introduce critical one-point failure points.
- Semantic security: One of the possible attacks in wireless media consists in trying to know how the encryption is done by looking for similarities in the transmitted packets. It is very common to send messages that look very similar, at least in some parts, and if we cipher always with the same key, an attacker could guess which parts of the ciphered message correspond to which parts of the plain text. For this reason it is important not to encrypt all the packets the same way.

  Semantic security assures that even if the same plain text is transmitted several times, the encrypted message will be different. This can be done by changing the encryption key periodically, by adding some randomness in the message, or by changing the encryption method.
- Data freshness: It checks that old messages are not running around the network wasting resources and introducing possible mistakes. It also prevents an attack consisting in a malicious user replaying older messages that he had overheard in the past. In [3] the authors have identified two types of freshness:
  - weak freshness. “It provides partial message ordering, but carries no delay information”
  - strong freshness. “It provides total order on a resquet-response par, and allows for delay estimation. Weak freshness is required by sensor measurements, while strong freshness is useful for time synchronization within the network.”
- Self-organization: In the context of WSNs we should always use distributed mechanisms to guarantee the scalability of the network and an easier deployment. In some self-organizing mechanisms, each node acts taking as

inputs only the local information it can gather from its neighbors, and following some rules defined a priori. These rules should try to align the behaviour of the individual nodes to the global goals of the network.

- Data confidentiality: It means that the exchanged data between two parties (nodes) can not be understood by other third party. The main mechanism to achieve confidentiality is using encryption.

  Encryption is the process to transform a message into another one written in cipher. It is carried out by using an encryption key. There are two kinds of ciphers depending on which kind of algorithm is used:

  - Symmetric key algorithms. The same key is used to encrypt and to decrypt.

    They are also known as "shared secret systems" or "private keys systems". Their main advantage is that they need less control messages to be sent and for this reason it does not consume so much power as the asymmetric algorithms. Some well-known examples are: DES and triple DES.

  - Asymmetric key algorithms. These systems use a pair of keys. One is used to encrypt and the other one is used to decrypt. These systems are also called "public keys systems". An example of one of these algoritms is RSA.

    Tipically one of the two keys is shared publicly and it is called the public key. The second key, the private key, should never be shared with anyone. When a message is sent using asymmetric cryptography, the sender encrypts the message using the public key, and the receiver then decrypts the message using its private key.

    This kind of systems are slower than symmetric key algorithms but the risk of someone intercepting the private key is smaller due to the fact that it is only kept in one of the two parties. The two parties do not share the private key.

- Data integrity: It ensures that the received messages have not been altered during the communication. It can be achieved using mechanisms like the MAC code.

2.1.1 Security protocols

Protocols such as IPSEC, SSL and SSH are working correctly in wired networks. However, these protocols are too heavy for using them in wireless sensor networks.

In wireless sensor networks we can distinguish two kinds of security [4]:

- Security inside the network. It ensures the communication between sensors and between the sensors and the base station. Base stations and nodes exchange different types of messages to communicate between them depending on the following patterns of communications:

  - Node to base station communication. e.g. sensor reading, alerts…
  - Base station to node communication. e.g. specific requests...

- Base station to all nodes. e.g. queries, information about routing, reprogramming of the entire network...
- Communication between a node and all its neighbors.

- Security outside the network. It ensures the communication between the WSN and the outside users. In principle, everybody should not be able to access to the services offered by the wireless sensor network, although it depends on the kind of services provided by the network.

2.1.1.1 SNEP

SNEP (Sensor Network Encryption Protocol) is a protocol that together with μTesla forms the SPIN architecture [3]. Each one provides different characteristics.

SNEP has the following properties:

- Data confidentiality and semantic security: There are different mechanisms to provide semantic security, SNEP uses two counters shared by both parties, which are incremented after each message is sent.
- Two party data authentication and integrity: It is provided using a message autenthication code (MAC).
- Replay protection: SNEP prevents the replay of old messages by adding a counter into the MAC generation.
- Weak freshness: It is achieved if the message is authenticated correctly, thanks to the counter introduced in the message MAC. The use of counters ensures message ordering.
- Low communication overhead: This protocol includes 8 extra bytes per message.

An example of exchanged messages between two nodes using SNEP would be:

$$A \rightarrow B: \quad \{D\}_{(K_{AB}, C_A)}, MAC(K'_{AB}, C_A || \{D\}_{(K_{AB}, C_A)})$$

Where A is the sender node, B is the receiver node, K is the encryption key, D is the data sent, C is the counter, M=MAC(K',C||E) is the MAC code and $E=\{D\}_{<K,C>}$ is the encrypted data.

SNEP can also provide strong freshness, although this requirement is not achieved with plain SNEP. The method to achieve strong freshness is the following:

Node A can reach strong freshness for a response from node B using a nonce. This nonce is a random number to exchange with B.

The mechanism is explained in the following scheme:

$$A \rightarrow B: \quad N_A, R_A$$

$$B \rightarrow A: \quad \{R_B\}_{(K_{BA}, C_B)}, MAC(K'_{BA}, N_A || \{C_B\} || \{R_B\}_{(K_{BA}, C_B)})$$

Node A generates nonce (N_A) and sends it with a request message to node B. Node B sends the response message with the MAC which was generated using the received nonce (N_A). If the received MAC matches the computed MAC then node A knows that node B generated the response.

2.1.1.2 µTESLA

This protocol [4] provides authenticated broadcast in WSNs. There are other protocols like TESLA that are used to achieve authenticated broadcast too, but they are unsuitable in wireless sensors environments because they use too much resources. For example, TESLA includes an overhead of 24 bytes per message, which would mean too much energy wastage in WSN environments.

µTESLA, in order to achieve an efficient broadcasts authentication scheme, delays the disclosure of the symmetric keys, and in this way it introduces asymmetry, even using symmetric authentication

In µTesla, the secret key used to encrypt the messages is updated continuously. Every key is produced introducing in a one-way function the next key, and in this way the receiver can be sure that all the messages come from the same source. If a key is lost, it can be recovered from the upcoming keys.

This authentication mechanism has the following phases for the communication pattern from base station to all nodes [3]:

- Sender set-up.

  The sender generates a sequence of secret keys of length n, known as the one-way key chain. These keys are generated using a one-way function F. µTESLA uses a cryptographic hash function such as MD5: $K_j=F(K_{j+1})$.

  Wikipedia defines a one-way function as “a function that is easy to compute but hard to invert”.

- Broadcasting authenticated packets.

  The sender decides to broadcast an authenticated packet to all the nodes. In order to do it, the time is split up into uniform time intervals, and the sender links each key of the one-way key chain with one of these time intervals.

  The sender creates a MAC (Message Authentication Code), which is generated with the message and the secret key of the current time interval. This MAC and the corresponding message are sent to every node.

  The nodes receive the packets. They store these packets in their buffers.

  The base station broadcasts the verification key (K_i) to all receivers at the time of the key disclosure, $i+\delta$. This delay known as δ depends on round trip time between the sender and the receivers.

- Authenticating broadcast packets.

  The nodes receive the verification key (K_i) and they verify these keys. If the key is correct it is used to authenticate the message stored at the buffer and the receiver replace the previous stored key (K_v) with the new received key (K_i).

- Bootstrapping a new receiver.

  Each receiver has to know the authentic key of the one-way key chain and also they should be synchronized with the base station in order to keep track of the different time slots.

  The distribution of the starting times and the shared key between sensors and the base station can be carried out with a mechanism providing strong freshness and point-to-point authentication. This mechanism is explained in the following scheme:

$$S \leftarrow R: \quad N_R, R_R$$

$$S \rightarrow R: \quad T_S \left| K_i \right| T_i \left| T_{\text{int}} \right| \delta \left| \right.$$

$$MAC(K_{SR}, N_M, T_S, K_i, T_i, T_{\text{int}}, \delta)$$

  Where S is the sender node, R is the receiver node and N_R is the nonce sent by R.

  The sender responds with a message consisting in the current time (T_S), the key from the one-way chain (K_i), the starting time (T_i), the duration of the interval (T_{int}) and the disclosure delay (δ)

2.1.1.3 LEAP

Sensor nodes and sink exchange different kinds of messages (sensor readings, routing data..) LEAP [7] establishes different security requirements for each type of message. Therefore, it uses four types of keys for each sensor node:

- Individual key. It is used to secure the communication between a node and the sink. Every node shares its individual key with the sink.
- Group key. It is used by the sink to encrypt the broadcast messages. It is globally shared between the sink and the nodes
- Cluster key. It is used by the nodes to secure the exhanged messages between them. It is shared by a node and all neighboring nodes.
- Pairwise Shared key. It is used to secure communications between a couple of nodes that require privacy. Ever node shares this key with each one of it immediate neighboring nodes.

LEAP also includes a protocol to authenticate the exchanged messages between nodes. It is based on the use of the cluster key as the MAC key. A receiver node will authenticate a received packet with its own cluster key. This key will be received in the cluster key establishment phase.

LEAP employs µTESLA to authenticate all the messages that the sink broadcasts to the sensors because in this case it is not necessary inmediate authentication, and µTESLA introduces a delay in the athentication.

2.1.1.4 Summary chart

In the following table, the advantages and disadvantages of the existing security protocols and of our protocol are showed.

Security Protocol	Communication Pattern	Advantages	Disadvantages
SNEP	Node → Sink Sink → Node Node → Node	Low overhead (8 bytes per message) It provides: ➢ Semantic Security ➢ Data authentication ➢ Replay protection ➢ Weak fressness	It uses two counters to achieve semantic security. There could appear problems to syncronize the counters.
µTesla	Sink (Broadcast) → Node	It provides asymmetry using a mechanism with delayed disclosure of symmetric keys. It does not require a very narrow synchronization between sink and node.	It does not provide inmediate authentication. The receiver node has to store in its buffer the packets until the disclosed key is received.
LEAP	Node →Sink Sink→Node Sink (broadcast) → Nodes Node→Node	It provides: ➢ Authentication ➢ Confidentiality ➢ Robustness ➢ Survivality	Nodes need more storage capabilities, each sensor node has to store four types of keys. It needs efficient mechanisms to update the keys. It asssumes that the sensor nodes are not mobile
Proposed protocol	Node → Sink Sink → Node **Node → Node**	It provides: ➢ Data authentication ➢ Data confidentiality ➢ Data integrity ➢ Data freshness ➢ Semantic security ➢ Reliability	It uses symmetric mechanisms for atuthentication, which is less secure than the asymmetric ones. It needs more energy from the sensors in order to provide reliability.

Table 2.1: Comparison between security protocols

2.2 Role of Reliability in WSN

The definition of reliability from Wikipedia is the following:

> In general, reliability (systemic def.) is the ability of a system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected

circumstances. The IEEE defines it as "the ability of a system or component to perform its required functions under stated conditions for a specified period of time."

In computer networking, a reliable protocol is one that ensures reliability properties with respect to the delivery of data to the intended recipient(s), as opposed to an unreliable protocol, which does not guarantee that data will be delivered intact, or that it will be delivered at all. [...]

Reliable protocols typically incur more overhead than unreliable protocols, and as a result, are slower and less scalable. [...]

TCP, the main protocol used in the Internet today, is a reliable unicast protocol. UDP, often used in computer games or other situations where speed is an issue and the loss of a little data is not, is an unreliable unicast protocol. [...]

In the context of distributed protocols, reliability properties specify the guarantees that the protocol provides with respect to the delivery of messages to the intended recipient(s).

One of the most common problems dealing with reliability in sensor networks is the ACK/NACK paradigm:

The most common mechanism for retransmission request in WSNs uses generally NACK messages ("*Negative Acknowledgement")* to inform about the missing fragments that need to be retransmitted. It generally needs in-sequence transmission of segments to avoid a NACK message implosion in the network. However, the main problem with NACKs is that in short messages composed of very few packets, it is not strange that all packets from the message get lost and thus the receiver can not start any recovery procedure, since it does not know in any way that it should be receiving a message.

Mechanisms based on ACK *("Acknowledgement")* messages do not have this kind of problem because an acknowledge message is expected back from every packet sent. The problem that this solution has is that in a very big network the number of control messages will grow very fast and will not be suitable for sensor networks because of energy constraints and because the collisions between packets will be more frequent, requiring more retransmissions.

2.2.1 Reliability protocols

There are several protocols for reliable transmissions over WSNs. We could classify them by whether they address the problem of upstream or downstream communication: RMST and ESRT are only upstream (nodes-to-sink) while GARUDA and PSFQ are only downstream (sink-to-nodes). ART considers reliability in both directions.

Another classificaton could be made taking into account if the mechanisms work hop-by-hop or end-to-end. As explained in [2], end to end reliability mechanisms perform better in terms of energy usage in scenarios with low loss probability, while hop by hop mechanisms achieve better results in high-loss environments. About energy usage, end to end mechanisms make the nodes further from the sink to retransmit more messages in

average and thus to spend more energy, while in hop by hop solutions the energy is more evenly used.

	PSFQ	*RMST*	*ESRT*	*GARUDA*	*ART*	*Proposed Protocol*
Reliability	Downstream	Upstream	Upstream	Downstream	Both	Upstream
	Hop by Hop	Hop by Hop / End to End	End to End	Hop by Hop	End to End	Hop by Hop
	NACK	NACK	-	NACK	ACK/NACK	NACK
Energy-aware	-	-	Yes	-	Yes	Yes
Loss Recovery	Yes	Yes	-	Yes	Yes	Yes

Table 2.2: Comparison of transport protocols

*2.2.1.1 **ART***

ART (Asymmetric and Reliable Transport) [8] is the only known protocol that addresses reliability both in upstream and downstream communication in WSNs. It is an event-based protocol that does not need to offer reliability at message level.

In the upstream direction, it assumes that the information from nearby nodes in the sensor network will be highly correlated, and proposes an scheme for event reliability, where there is no need to transport reliably all of the packets generated by the nodes as long as the sink is notified about the existence of the event. It just provides reliable transmission for the first message from each event. On the sink-to-nodes direction, it guarantees query reliability, this is, the reliable reception of messages by a subgroup of sensor nodes that cover the entire area of interest, and not necessarily by all the nodes in that area.

In ART, a set of essential nodes covering the whole area of interest are chosen by the sink in a centralized manner. It assumes that the sink can reach all nodes in a single hop and uses this fact to implement some centralized mechanisms that take into account the energy remaining in the sensors. It is more energy-efficient that the sink collects all the information about the remaining energy from the sensors and computes a near-optimal solution, than spreading all this information through all the nodes so that they could compute it distributedly.

Non-essential nodes forward the packets along the path from essential nodes to the sink, but it is the essential nodes who are in charge of providing query and event reliability.

ART uses a NACK system for sink-to-nodes messages, where the sink only retransmits a message when it receives a NACK for that packet. To guarantee that the last packet is received with reasonable delay the nodes must transmit an ACK message just for the last query or when the next query will happen a long time in the future.

In the nodes-to-sink direction messages, ART uses an ACK mechanism. In the upstream direction, the essential nodes wait for ACKs for just the first message from the same event (called event notification message). If the ACK is not received after a timer expires, it retransmits the message. ART has also a congestion managing mechanism that is based on the receival of these ACKs by the essential nodes. If they do not receive an ACK for the event notification messages after a time, they will tell their neighboring non-essential nodes not to send more packets until the ACK is received.

ART assumes that all the nodes know their position and that the sink can reach all nodes in a single hop.

2.2.1.2 PSFQ

Pump Slowly Fetch Quickly (PSFQ) [9] was the first transport protocol to introduce reliability in WSNs. It is designed to be scalable and energy-efficient, trying to minimize the number of signaling messages and relying on multiple local timers. It only addresses the reliable communication from sink to nodes.

An initial version of PSFQ appeared first in [10], and it was later extended in [9]

The main idea behind PSFQ is to send packets from the sink node at low speed in order to give time to the nodes to recover any missing fragment before the next fragment arrives. The data loss is detected like a gap in the sequence number of the received fragments, and the fragment recovery takes place hop-by-hop by aggressively asking for retransmissions via NACK messages. The hop-by-hop retransmissions will scale better in larger networks, as it reduces error accumulation.

The fragments are always transmitted in-sequence to avoid NACK implosion and cached in all the nodes in the path. The NACK messages are never forwarded unless the number of NACKs received for the same packets exceeds a thershold.

PSFQ uses random delays when transmitting messages to reduce collision probability and to avoid using RTS/CTS (*"Request to Send/Clear To Send"*) mechanisms.

Other interesting ideas implemented in PSFQ are the following:

- Whenever a node is going to retransmit a lost segment to a neighbor, it waits for a random time and if it hears another node already retransmitting that missing segment, it cancels its own retransmission.
- When forwarding normal data packets, a node will cancel the forwarding if it hears the same fragment being transmitted already by 4 neighbors, as the expected extra coverage provided by transmitting the message will be quite low.
- A node will cancel the sending of a NACK if it hears the same NACK being transmitted by a neighbor. It will resend the NACK after some time, as it is not guaranteed that the message repair to its neighbor will also reach him.
- If the file transmitted is too small or the last fragment of it is lost, the loss would not be detectable with the current mechanism, so PSFQ adds a proactive fetch operation that consists in sending NACK messages even when no loss has been detected if a timer expires without receiving any more fragments from the same file. This timer is proportional to the number of missing segments from the file.

- PSFQ provides a feedback report mechanism consisting in setting a report bit in every message that the sink wants information about, and all the nodes will aggregate their feedback information into one single packet on their way back to the sink. This is also used in the case of one-fragment messages. The sink can set up this report bit and this way it will receive a report message back that will serve as an implicit acknowledgement message.
- The last extra mechanism introduced by PSFQ is a signal-strength preference table. Each node keeps a list of nodes and their average signal strength measurements. Nodes will have a preferred parental node, and they will only send NACKs if they detect a gap in the segments sent by this preferred node. At the same time, in the NACK message they will include an identifier of which is the preferred node to make the retransmission. The rest of the nodes hearing this NACK will also prepare retransmissions but with a bigger delay, to let the preferred node answer first.

2.2.1.3 ESRT

ESRT (Event to Sink Reliable Transport) [11] was the first protocol to introduce the concept of event-to-sink reliability, which takes profit of the information redundancy from packets coming from nearby nodes, and aims not to the reliable transmission of all of the packets but just of the minimum number of packets that will give the sink node all the necessary information. It does not use node IDs but event IDs.

The ESRT protocol runs mainly on the sink node, which is in charge of detecting the state of the network depending on congestion and reliability measurements, and updating the node's reporting frequency. This protocol does not use retransmissions or caching mechanisms at the nodes, it just changes the reporting frequency so that more copies of the same message are sent and it will be more probable for one of them to reach the sink node. It searches end-to-end reliability instead of using hop-by-hop retransmissions.

ESRT however, does not support more than one event at the same time, and the mechanism used by the nodes to set up the event IDs in their messages is not very well explained. ESRT manages the reporting frequency as one global parameter of the whole network.

ESRT is designed for static sensor nodes and can support only slow topology variations through time.

2.2.1.4 RMST

RMST (Reliable Multisegment Transport) is a reliable transport protocol built on top of the multicast diffusion routing protocol. In [5] they study the benefits of implementing reliability either in the link or in the transport layers. Reliability in the link layer is always hop-by-hop, whereas reliability in the transport layer can be either hop-by-hop or end-to-end.

As a result of the study performed in [5] hop-by-hop recovery appears to be necessary instead of end-to-end traditional mechanisms, and it can be done at the link or transport layer, or at both of them. About implementing reliability in both layers, reliability in

transport layer does not help very much to improve the performance when there is already reliability at the link layer. It seems preferable to do hop-by-hop retransmissions at transport layer because the overhead introduced is less and thus the energy wastage is lower. Using RTS/CTS and ACKs at link level would introduce a lot of overhead messages and would translate into energy wastage in the context of WSNs.

On the other hand, in high error rate scenarios without any kind of reliability at link layer, the routing protocol might have strong problems stablishing the routes, as the packets get lost very often, and the reliable transport mechanisms would never get to work as they would only be supported at transport layer once the routes are already stablished.

The conclusion of the RMST study is that the best solution would be to use always reliability at transport layer, and also include it in the link layer for the case of control and unicast data packets.

About the RMST protocol implementation, it is designed to work with directed diffusion, which is a multipoint routing protocol used in sensor networks. One of the most important characteristic of direct diffusion is that it is data-centric, using pairs of attribute-value and publishing interests for them to stablish routes. RMST is built over directed diffusion and adds only one control message, which is the NACK needed for requesting retransmissions.

RMST offers two modes of functioning: employing end-to-end retransmissions where only the source and sink nodes need to keep a cache, or using hop-by-hop retransmissions where every node in the path from source to destination needs caching packets.

RMST offers some similarities to PSFQ, but it uses also retransmissions at link layer.

2.2.1.5 GARUDA

GARUDA [12] is another protocol for reliable transmission from the sink to the nodes that is “scalable with respect to the network size, message characteristics, loss rate, and reliability semantics”. It is designed to work in networks with a single sink and static sensors.

It addresses the problem about using NACK or ACK messages (see section 2.2), and solves it by transmitting a high-energy pulse before transmitting the first packet from each message to be sent. This pulse is almost immune to channel loss and it can be heard by either idle sensors or sensors already receiving a data packet. This pulse characteristics are very different from data packets, and this makes it possible to receive the pulse and the data packet at the same time without interference problems. The pulses are used to tell the rest of the network that they should expect a first packet soon, and that in case of not receiving it they should send a NACK message. In this way the first packet is always transmitted reliably, and the rest of them use the normal NACK mechanism.

GARUDA creates a set of loss recovery servers called “the core” that tries to approximate the minimum dominating set (MDS). It is constructed easily by flooding the network with just the first fragment of each message. The core nodes are updated each time a first-packet is sent. An extra mechanism that decides whether or not a node

should participate in the construction of the core is used for implementing different reliability semantics.

The recovery process in GARUDA is carried out in two steps: at first all of the core-nodes ask for retransmissions for their missing segments, and once they have completed the whole message, the non-core-nodes ask for retransmissions of the segments that they have not overheard during the first step. This method minimizes the number of NACKs and retransmissions during the first step, since only the core nodes are trying to recover fragments, and they are a just small fraction of all the nodes in the network, and in the second step the recovery is performed locally to each core node neighbors.

GARUDA uses out-of-sequence forwarding to achieve a better use of the limited bandwidth, but this introduces a problem with possible NACK messages implosion when higher sequence fragments are received first. To solve it, this protocol uses an availability map, transmitted in each packet, that tells the downstream nodes which segments are available upstream so that they will never ask for retransmission of a fragment that has not arrived yet to the upstream node. The use of the availability map introduces some extra overhead but its benefits are higher than its drawbacks, since it avoids the sending of useless NACK messages.

2.2.1.6 Multicast protocols

RMTP [13] and PGM [14] are multicast protocols with reliability mechanisms, the two of them have some ideas that could be further explored in order to apply them in sensor networks.

RMTP has a mechanism to avoid the ACK implosion problem using a hierarchy of special nodes that aggregate multiple ACKs from downstream into a single one and that do caching of data for retransmission of lost packets. This mechanism fits well in self-organized wireless sensor networks in the sense that every node acts as a virtual sink or source of information and the next node in the path can not distinguish between a virtual sink/source and the real one.

PGM uses NACK messages, and has the ability to aggregate NACKs from downstream to avoid transmitting multiple NACKs for the same lost fragment. It also waits for a random time before sending the NACK, while listening to possible retrasmissions for that same fragment, as it is very common that neighboring nodes have the same missing fragments.

However these multicast protocols can not be used directly in sensor networks because they need too many resources and they are sending control packets all the time, which would consume a lot of useful energy in a sensor network. They also assume that every node in the network has a unique address used for routing, whereas in WSNs the routing tends to be data-centric, without any global identification.

2.2.1.7 TCP-based protocols

There has been some attempts to improve the performance of TCP over wireless links [15], however, this kind of protocols require normally a lot of memory and computation resources and are more suited for wireless networks than for WSNs. TCP assumes that

the main cause of packet loss is congestion, while in wireless networks the main cause is usually bad link quality.

2.3 Wireless channel model

In WSNs, the signal is transmitted on a RF channel. It will be influenced by enviromental factors such as interferences, reflections, scattering and shadowing. Due to the these factors influence the signal may not reach the receiver node with enough power level and the packet could be dropped by the MAC layer. The power level necessary to detect a signal in the receiver node is called the sensitivity threshold.

The received power depends on the amount of power transmitted by the sender node and on the path loss. There are different models to predict the received signal strength and its degradation with the distance. [16], [17], [18].

2.3.1 Free space propagation model

This model asummes that there are no obstacles between the transmitter and the receiver node (this feature is known as line-of-sight, or LOS). The equation to calculate the received power is:

$$P_r \equiv \frac{(P_t \cdot G_t \cdot G_r \lambda^2)}{((4 \cdot \pi)^2 \cdot d^2 \cdot L)} \tag{2.1}$$

where:

P_t = transmitted power

G_t = transmitter antenna gain

G_r = receiver antenna gain

λ = wavelength in meters

d = distance between transmitter node and receiver node in meters

L = the system loss not related to propagation

The previous equation can be simplified using a constant K_1 which represents the characteristics of the transceivers :

$$P_r = \frac{(K_1 \cdot P_t)}{d^2} \tag{2.2}$$

We know that the minimum received power needed by the receiver node is the sensitivity threshold, so a packet will reach sucesfully the receiver node if the distance between both nodes (tranmitter and receiver) complies with the following equation:

$$d \leqslant \sqrt{K_1 \cdot P_t} \tag{2.3}$$

The equation 2.2 indicates that the received power is inversely proportional to the square of the distance d that separes the transmitter node and receiver node.

The free-space loss are represented in the following equation:

$$L = \left(\frac{4 \cdot \pi \cdot d}{\lambda}\right)^2 \tag{2.4}$$

2.3.2 The two-ray ground model

It considers two propagation paths between transmitter and receiver, the direct path and the reflected path. The equation to calculate the received power is:

$$P_r \equiv \frac{(P_t \cdot G_t \cdot G_r \cdot h_r^2 \cdot h_t^2)}{(d^4)} \tag{2.5}$$

where:

h_t = transmitter antenna height

h_r = receiver antenna height

We can simplify the above equation like we did in the previous model, using K_2 as a constant that includes all of the characteristics of the transceivers:

$$P_r = \frac{(K_2 \cdot P_t)}{d^4} \tag{2.6}$$

In this case, the maximum distance between nodes would be:

$$d \leqslant \sqrt[4]{K_2 \cdot P_t} \tag{2.7}$$

2.3.3 The log-distance path model

In this model the path loss is proportional to the distance *d* between nodes raised to an exponent α.

$$P_r \alpha \frac{P_t}{d^\alpha} \tag{2.8}$$

The value of α depends on the environment on which the nodes are deployed. In the following table they are represented some of these values for different environments:

Environment	α
Free space	2
Urban area	2.7-3.5
Indoor LOS[i]	1.6-1.8
Indoor no LOS	4-6

Table 2.3: Some typical values of path loss exponent α.

2.3.4 The log-normal shadowing model

It is also called slow-fading model. The path loss is modeled as a random variable with log-normal distribution.

The received power is represented as follows:

$$P_r(d) \equiv P_t + G_t - (\overline{PL(d)}) + G_r \tag{2.9}$$

where:

P_t = transmitted power in dB

P_r = received power in dB

G_t = transmitter antenna gain in dB

G_r = receiver antenna gain in dB

d_o = reference distance

PL(d) = path loss at a distance d from the transmitter

$$PL(d) \equiv PL_o(d_o) + 10 * n * \log(\frac{d}{d_o}) + X_\sigma \tag{2.10}$$

n = path loss exponent which depends on the propagation environment. The table 2.3 shows some typical values of n.

Environment		n
Outdoor	**Free space**	2
	Shadowed urban area	2.7 to 5
Indoor	**Line of sight**	1.6 to 1.8
	Obstructed	4 to 6

Table 2.4: Typical values of path loss exponent n

$Pl_o(d_o)$ = free space path loss at the reference distance

$$PL_o(d_o) \equiv 20 \cdot \log(4\pi \frac{\dot{d}_o}{\lambda}) \tag{2.11}$$

i Line of Sight

X_σ = shadowing term. It consists in a zero-mean gaussian random variable with standard deviation σ. The table 2.4 collects some typical values of σ

Environment	σ (dB)
Outdoor	4 to 12
Office, hard partition	7
Office, soft partition	9.6
Factory, line of sight	3 to 6
Factory, obstructed	6.8

Table 2.5: Some typical values of shadowing deviation ***σ***

2.4 Radio Reception model

The probability of receiving succesfully the packet in the receiver node depends on different factors: noise in the channel, the received signal strength, the modulation and encoding scheme used, the size of the packet, and the physical layer particulars of the radio. To calculate this probability we have used the model described in [16] which corresponds to a Mica2 Mote that uses a non-coherent FSK radio

The PRR ("*Packet recepcion rate*") is defined in [16] and [19] as:

$$PRR = (1 - \frac{1}{2} \cdot \exp^{\frac{Eb}{(2 \cdot No)}})^{8 \cdot L} \tag{2.12}$$

where

E_b = energy per bit (Joules)

R_b = data rate (bps, bits per second)

N_o = noise power spectral density (W/Hz)

B = noise bandwidth (Hz)

L = length of the packet in bytes

We can express the relation between E_b and N_0 using the Signal-to-noise ratio (SNR). The relation between these variables is given by:

$$SNR = \frac{E_b \cdot R_b}{(N_o \cdot B_N)} \quad \text{, and then,} \quad \frac{E_b}{N_o} = \frac{SNR \cdot B_N}{R_b} \tag{2.13}$$

The signal to noise ratio (SNR) can be calculated as follows:

$$SNR = \frac{P_r}{P_n} \tag{2.14}$$

To determine the SNR it is neccesary to know the received signal power and the received noise power. The first of these powers is calculated using the wireless channel model described in 2.3 and the received noise power is given by the following equation: ([20])

$$P_n = (F+1) \cdot K \cdot T_o \cdot B \quad (2.15)$$

where:

F = noise figure

K = Boltzmann's constant

To = ambient temperature

B = equivalent bandwidth

2.5 Study of current sensor nodes characteristics

Nowadays there are different models of sensor nodes (called also motes) present in the market. Each one of them have different characteristics, which we need to study if we want to make sure that our protocol can be implemented in the real world.

One of the most important aspect that we need to study is the storage capacity that the nodes have, to check if it is possible to store in it the variables that the protocol needs. Here we present a comparison table between some of them, showing their main features (data taken from [21]):

	MICA2	**Imote 2**	**MicaZ**	**TelosB**
RAM (KB)	4KB SRAM	256kB SRAM 32MB SDRAM	4KB SRAM	10Kb
Program flash memory	128Kb	32MB	128Kb	48Kb
Measurement flash memory	512Kb	-	512Kb	1024Kb
Configuration EEPROM	4Kb	-	4Kb	16kb
Processor	ATMega128L 7.37MHz	Intel PXA271 13MHz - 416MHz	ATMega128L 7.37MHz	MSP430 8MHz
Frequency band	433 or 868/916 Mhz	2.4 -2.4835 Ghz	2.4 -2.4835 Ghz	2.4 -2.4835 Ghz
Size	58x32x7	36x48x9	58x32x7	65x31x6
Weight (grams) excluding batteries	18	12	18	23
Battery	2xAA	3xAAA	2xAA	2xAA

Table 2.6: Comparison between current sensor nodes characteristics

Comparing the information in this table 2.6 with the size of the data that our protocol needs to keep (see section 3.5.1, “Memory requirements”) we can see that our protocol could run without problems in any of these sensor node models, which represents a great advantage of our proposed protocol over some other proposed solutions.

2.6 Study of power consumption

Nodes consume energy when they are transmiting, receiving or idle. We need to know the amount of energy consumed by our nodes in order to simulate later the effect that this energy loss can have over the sensors. More specifically, this will affect our protocol in the moment that a sensor node runs out of energy and it can no longer transmit packets.

The next table 2.7 shows the power consumption for different radio devices. This information can be found in [16].

Radio	Data Rate	Transmit	Receive	Energy/bit Transmission	Energy/bit Reception
CC 2420	250kbps	52mW	59mW	$2.08 \cdot 10^{-7}$ J/bit	$2.35 \cdot 10^{-7}$ J/bit
CC 1000	19.2kbps	50mW	29mW	$26 \cdot 10^{-7}$ J/bit	$15.1 \cdot 10^{-7}$ J/bit
MIT uAMPS-1	1Mbps	330mW	279mW	$3.3 \cdot 10^{-7}$ J/bit	$2.79 \cdot 10^{-7}$ J/bit

Table 2.7: Power consumption for different radios

In the table 2.7 we highlight the CC1000 radio device, because it is the one used by MICA2 motes and it is the one that will be used during the evaluation of our protocol.

3 Proposed distributed reliable and secure protocol

Our protocol will try to establish a secure and reliable connection between the nodes and the main sink node. In [4] and [6], the importance of using hop-by-hop mechanisms in security protocols is addressed, and in [5] they explain about the importance of using also hop-by-hop procedures in reliability protocols, and for this reason we have decided to take our protocol to the link layer so that every node can communicate with its neighbors via our protocol.

The immediate consequence that this brings out is that we are no longer addressing the problem of communicating the nodes with the sink, but communicating the nodes between them in their immediate neighborhood. What we have here in fact is a communication node-multinode, because of the inherent multicast properties of the wireless medium.

Another consequence is that instead of having a different secret key for each node-to-sink communication, known only by the node and the sink, now we need either one different key for each node, known by all of its neighbors, or one global key shared by all the nodes in the network.

The first option has the disadvantage that it requires all the nodes to keep a table with the keys from all of its neighbors, which in dynamic environments can result unpractical, as it will grow very fast, taking a lot of memory, and the nodes from WSN have very limited storage capacity.

In fact, if we used some of the existing security protocols, we would need to keep 3 different variables for every neighbor in every node:

- one secret key used to construct the MAC code
- another key for encrypting the message
- and a counter value that is updated each time a message is sent/received to achieve semantic security and reliable transmissions.

In a protocol used to communicate node-to-node, we would want to reduce the number of these variables, and a good option would be to establish some artificial relationship between them so that we could extract one of these parameters from the others using some kind of function.

The only way of not keeping tables with the neighbors' keys would be to use a global key for all the network. This option is much simpler but it has the drawback that if a single node is compromised by an attacker, it can then run free in all the rest of the network. To prevent key disclosure by an attacker, the key could be changed periodically, even though this would need that the nodes are time synchronized or that there exist one central node that informs all of the network about the new key to be used.

Instead of changing the key, another option is to change the way how we encrypt. It could be interesting to use a cipher block with 3 inputs:

- the message to be ciphered
- the secret key

- and a another number that changes with time

In this way we can guarantee semantic security (this concept was previously explained in section 2.1) and make it more difficult for an attacker to break the secret key. If instead of using a cipher block with 3 inputs, we want to use a standard cipher block with 2 inputs, we could yet combine th ciphering key K2 and this changing number making the logical XOR operation and in this way obtain a new key for every message. In our protocol we will use this option, and this changing number will be the sequence number of the message.

This protocol also solves the probem of interfering sensor networks, where they exist several different overlapping networks. The nodes will be able to detect the messages coming from other networks, and using the MAC code thay will realize that the sending node does not belong to the network, and the packets will be discarded.

One of the main characteristics of our protocol is that it will be completely distributed. There will not be any central station controlling the behaviour of the nodes. In this way, the sensors can self-organize themselves and use only local information, which makes the protocol very scalable with respect to the network size and allows us to forget about the problem of centralized control messages that have to reach all the nodes of the network.

<table>
<tr><td colspan="5">Node - node communication protocol</td></tr>
<tr><td rowspan="2">Keys</td><td>K1</td><td colspan="2">Used for computing the MAC code</td><td rowspan="2">Global constant keys.</td></tr>
<tr><td>K2</td><td colspan="2">Used together with the Seq no. to encrypt the messages.</td></tr>
<tr><td colspan="2" rowspan="2">Sequence number</td><td colspan="2">Used to provide reliability.</td><td rowspan="2">Updated as a counter with every message sent.</td></tr>
<tr><td colspan="2">Used together with K2 to encrypt the messages.</td></tr>
<tr><td colspan="2" rowspan="2">Database in every node:</td><td colspan="3">For every neighbor we keep the following information:</td></tr>
<tr><td>Neighbor identifier</td><td>Last Sequence number received</td><td>Timestamp of the entry</td></tr>
<tr><td colspan="2">Overhead:</td><td colspan="3">MAC code (8 Bytes) + Sequence number (2 Bytes) = 10 extra Bytes per message</td></tr>
</table>

Table 3.1: Summary chart with important characteristics from the proposed protocol

3.1 Issues addressed by our protocol:

- Data confidentiality: it will be achieved with encryption using a secret key. We will use only symmetric mechanisms, because although asymmetric ones are more secure generally, they consume too many resources and are not suitable for WSNs.
- Authentication: we will use a Message Authentication Code (MAC) to make sure that the message comes from the node that is supposed to be sending it, as

only the nodes from inside the network know the secret key and could compute it correctly.

- Authorization: we can identify the sender node using the MAC, and then, once we know if the node belongs to our network or not, we will also know wether it is authorized to send and receive messages.
- Data integrity: it will be provided at the same time that data authentication, because if a message has been modified in any way, the MAC code will not correspond to the modified message and it will be discarded.
- Reliability: we will add a sequence number in the messages so that the receiving node knows in every moment if some packets have been lost, and in this way it can ask for retransmissions.
- Data freshness: we will achieve this goal using the sequence number counter present in every message. Receivers will discard messages coming with an old sequence number. This counter prevents against an adversary replaying older messages pretending that they are new.
- Semantic security: One of the newest ideas from this protocol consists in including the sequence number also in the ciphering block as another input, so that we can assure that the ciphering is different for each message. In this way we can guarantee that even if the message plain text is the same, the ciphered message will be different.

In the following table we show a summary with the security requirements covered by our protocol and by the main security protocols.

	SNEP	μTesla	LEAP	Proposed protocol
Data confidentiality	✔		✔	✔
Data authentication	✔	✔	✔	✔
Data integrity	✔	✔	✔	✔
Data freshness	✔	✔	✔	✔
Semantic security	✔			✔
Replay protection	✔			✔
Reliability				✔

Table 3.2: Security requirements covered by the different protocols

3.2 Important assumptions

In order to focus on the exact scenario in which we will work, we need to make some assumptions:

We will assume that the sensors are placed randomly in the area of interest, and that the node density will be enough to guarantee that there is connectivity between all the nodes and that every point in the area is covered by the sensing field from at least one sensor.

The secret key used to encrypt the messages will be known a priori by all the nodes. This protocol will not consider the distribution of secret keys during the network bootstrap. This secret key will never change during the protocol run time, instead, we will use the sequence number together with this global key to produce a new key for every message.

In SPINS [3], the authors argue that with symmetric encoding mechanisms, a sensor node from inside the network could make its neighboring nodes believe that he is the main sink node, and thus make all the messages be redirected towards him. However, in our work we will try to build strong authorization measures that will make possible to suppose that if a node has been correctly authorized into the network, it will behave correctly and will not try to attack the network functionality in any way. In this way we will be able to keep our protocol more portable, lightweight, and simpler. We will not consider insider attacks.

3.3 Project Scenario

In this project we will work with Wireless Sensor Networks composed mainly of two kinds of nodes: the sensor nodes and the sink node.

3.3.1 Sensor nodes

The sensors will be very tiny devices placed randomly in the area of interest, covering it completely with their sensing field. They will not have any kind of external energy support. Due to the fact that they will have to depend on batteries there is a special need to control carefully every action that the nodes make, in order to maximize the node's lifetime and in general the network funcionality's lifetime.

The small size of this devices will also influence in the technologies used to build them and in this sense it will limit the amount of memory they can manage, and their computation capacity.

In our simulation, we will give the nodes a specified energy charge and we will decrease this charge for every action they make, specially transmitting or listening to the radio channel. When the energy charge from a node falls below a threshold, the node will stop working, and the rest of the network will have to adapt to this situation. When a lot of the nodes fail the network will have problems mantaining the connectivity and eventually it will stop providing the funcionality it was designed for. This is the network lifetime that we will try to maximize.

For a more detailed study of the current sensor nodes present today in the market, and a comparison between its characteristics, see section 2.5.

3.3.2 Sink node

In our sensor network we will assume that there is only one sink node that collects all the information sent by the nodes. This sink node does not have all the limitations from the sensor nodes, because as it is only one for the whole network, the maintenance will be easier and the size and economic cost will be of less importance than in the sensor nodes. It can have an external power supply and its transmission power will not be so limited.

In our protocol, we consider just node to node communication, in order to make it more scalable and de-centralized. For this reason we will consider the sink node just like any other one, even if its battery will not run out in the same way as the sensing nodes.

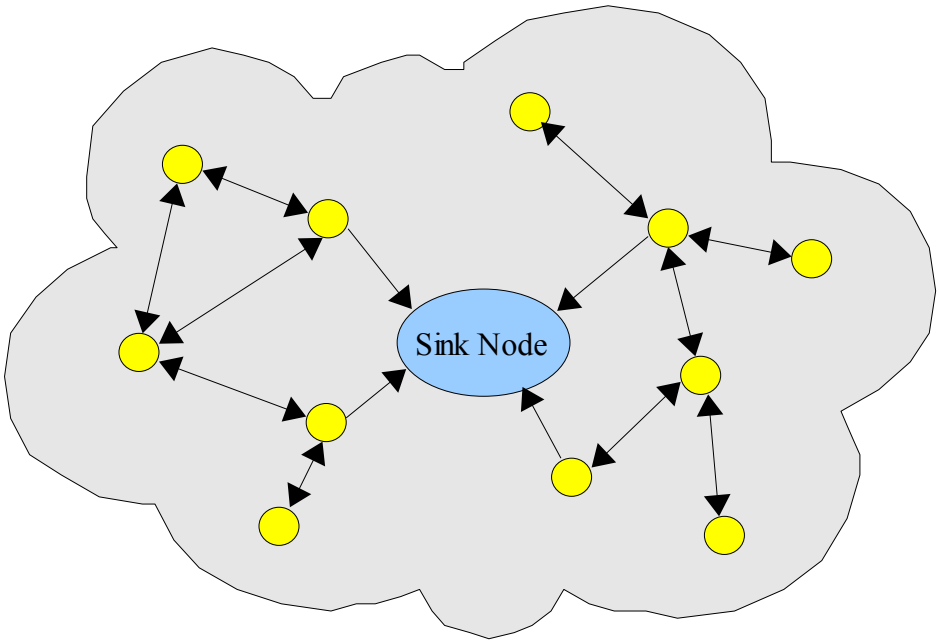


figure 3.1: Typical structure of a WSN

3.3.3 Applications

Nowadays there can be identified several applications for WSNs, like medical, home, environmental, or military appications. However, this project will focus mainly on medicine and home scenarios because we have considered them interesting to develop new applications.

In the case of home environments, wireless sensor nodes could be located in different parts of houses to measure differents parameters like the temperature, smoke or to detect movements, and in this way create an “smart house”. The sensor networks will not only be used to make people lifes more comfortable, but also to provide important emergency services, like guiding them through the safest route to save their lifes in case of emergency.

In the health care field, these devices could be placed on different body organs to monitor vital functions and report them to the doctor in real time. They could be used inside hospitals, to track material, patients and doctors, or outside hospitals, monitoring elderly people health as if they were in the hospital, for example.

3.4 Protocol details

3.4.1 Sending a packet

When a node wants to transmit a message, it will first compute the MAC code of this message using a secret key K1. This key K1 is a global parameter of the network, and it is known by all the nodes. Then it will encrypt the message using a cipher block with

two inputs: the message, and a secret key built from the K2 key and the sequence number of the message. The key K2 is also a global parameter of the network, shared by all the nodes.

The reason of including the sequence number in the ciphering block is to achieve semantic security, in order to make it more difficult for an attacker to get the K2 key. We will combine K2 and the sequence number by performing the XOR logical operation with the two of them, and this way easily obtaining a new key for each new packet.

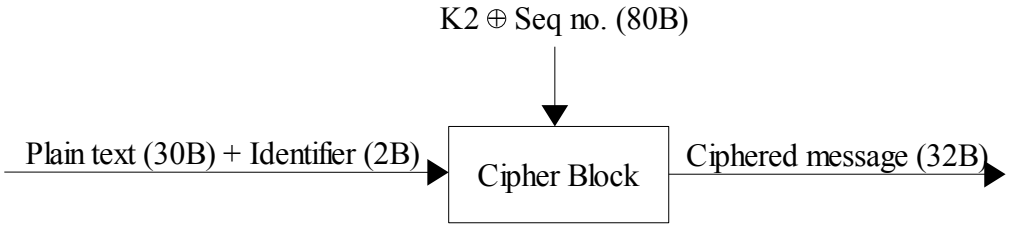


figure 3.2: Ciphering using the secret key and the sequence number

Notice that even when we are ciphering the packets in a different way each time, using K2 and the sequence number, in the case of the MAC computation we always use K1. It is not necessary to update the K1 key, for two reasons:

- First, the MAC code generated is much smaller than the original message, and in contrast with the encryption block, the output does not carry all the information present in the message.
- In second place, the MAC is generated with the uncyphered message, so an attacker would need to break first the encryption block to know which MAC corresponds to each message. It is important to compute the MAC with the plain text instead of with the encrypted message.

For this two reasons it is very unlikely that an attacker can guess K1, because the only thing he can have access to is the small MAC code, and from it the attacker cannot guess anything else.

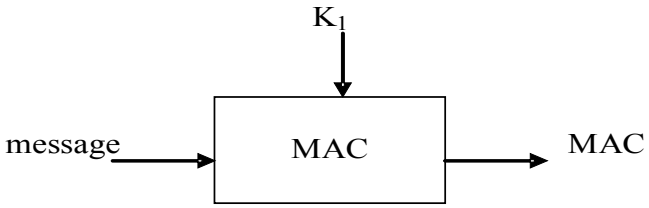


figure 3.3: Creation of the MAC code

Finally, the sent packet will consist in the sequence number, transmited without ciphering, the MAC, and the ciphered message. The objective of transmitting the sequence number without ciphering is to achieve a reliable communication, and make it possible to ask for retransmissions of lost packets. If the sequence number was sent ciphered, and a packet is lost, the receiver node would not know how to decipher the next correctly received packet, as it will be encrypted with the K2 together with the sequence number.

Seq no.	MAC	Ciphered message
2 Bytes	8 Bytes	32 Bytes

figure 3.4: Packet structure

3.4.2 Receiving a packet

Every receiving node must keep a neighbor table in which each input will have three fields:

- The neighbor node identifier.
- The last sequence number received from the neighbor.
- A timestamp used to determine which is the older entry in the table.

This table will be dynamic because in our wireless scenario the neighboring nodes are not always the same, either because they are moving or because the wireless link quality varies through time.

When a node receives a packet, it will use the sequence number received in the packet and K2 to decipher the message. It will then compute the MAC with the deciphered message and the K1, and check that it matches the received MAC. If the two MAC´s do not match, the node will discard the packet.

The use of the sequence number makes that our protocol ensures packet ordering. The receiver node will check that the received sequence number is consecutive to the last one received from the same node, and in case that there is a gap in the sequence, it will ask for retransmission. There are two different cases when a node receives a packet:

a) If the identifier received is not present in the neighbor table, the receiver node will create a new input in the table.

b) If there is already an entry in the table with this node identifier, it will check if the stored sequence number is consecutive to the received one. If the sequence number is not consecutive there are three possibilities:

- There has been some interference in the wireless channel and some packets have been lost. The protocol will detect a small gap in the sequence number and then it will ask for retransmision of all the missing packets.
- Due to the node movements it can happen that a node enters the transmision range of another node after being far from it for a long time. In this case the detected gap in the sequence number will be much bigger and asking for retransmissions of all the missing packets would be very inefficient. For this reason if the difference between the two sequence numbers exceeds a threshold *Th*, the receiver node will not ask for retransmissions. We have to suppose that the missing packets were correctly transmitted when the nodes were far from the transmission range of each other, using different neighbors to route the packets.
- If a packet with an older than the current sequence number is received, the packet will be discarded. This can happen either because a malicious user is

injecting older messages into the network, or simply because a packet suffers a strong delay due to the channel characteristics. See figure 3.5.

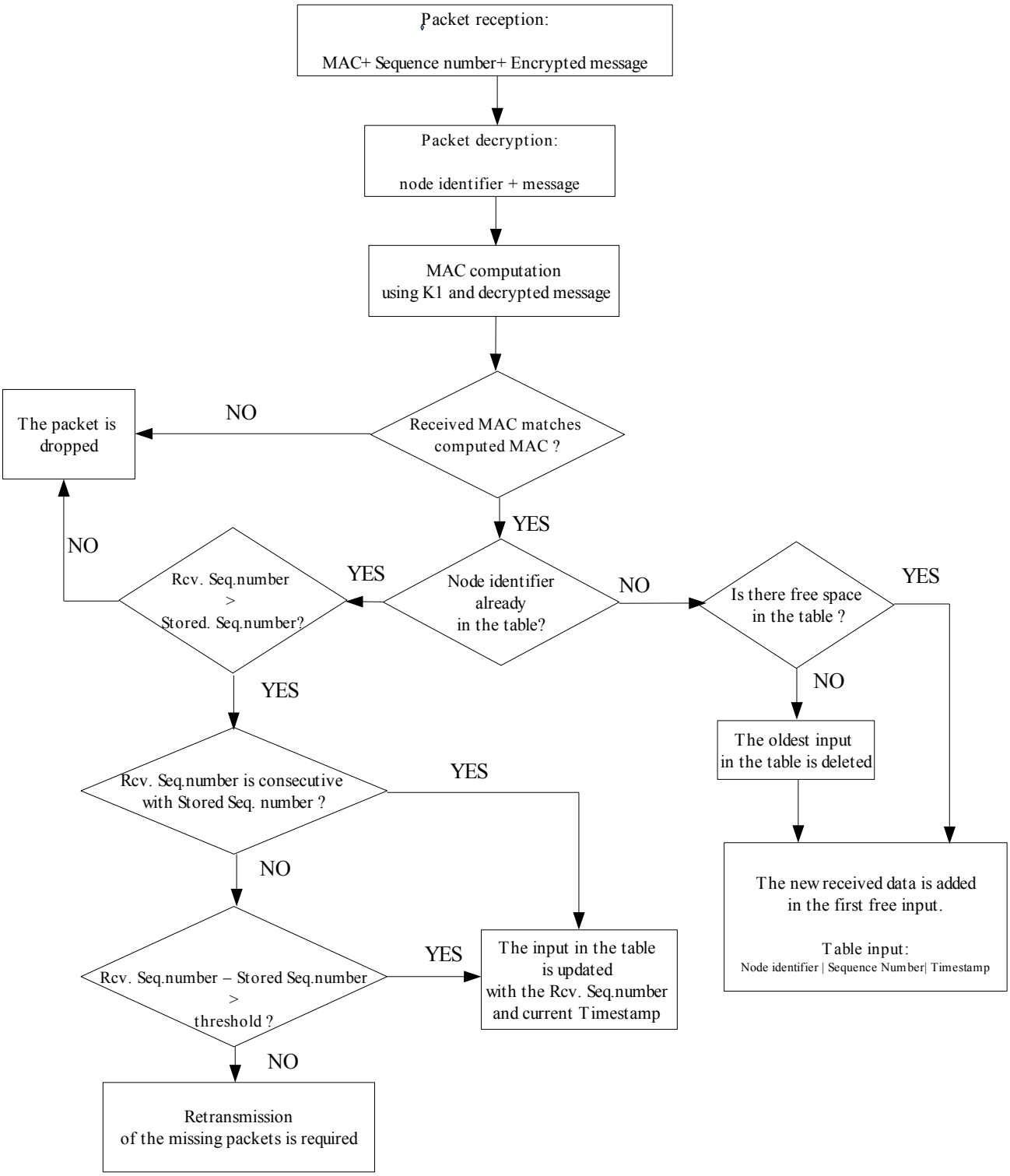


figure 3.5: Flow diagram showing the reception of a new packet

When a node wants to ask for retransmission of one or more packets, it will send a NACK message with the sequence number of the last packet received correctly in-sequence, and in this way it does not need to send a new message for every missing packet. In this NACK message the node will also include the identifier of the original sender, because it is the node that should listen to the NACK and retransmit the missing packets.

Original sender Id	Last seq. no. received
2 Bytes	2 Bytes

figure 3.6: NACK Packet structure

When a node receives a NACK message, it will check if the identifier is itself, and in that case it will start retransmitting from the packet with the next sequence number. (see figure 3.7)

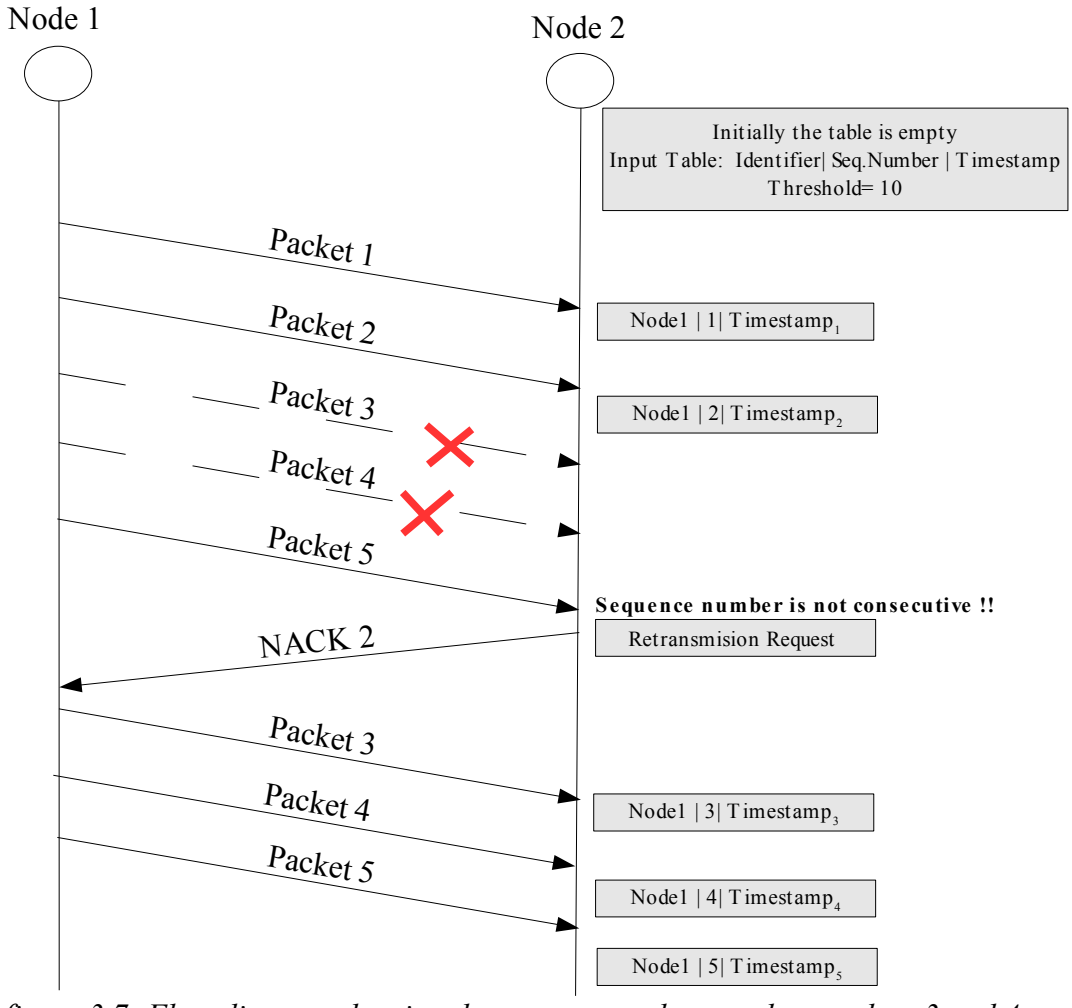


figure 3.7: Flow diagram showing the message exchange when packets 3 and 4 are lost

The node that is sending the NACK message will wait for a time T_N, and after that it will understand that either the NACK message has been lost or the retransmitted packet

has been lost again, and it will send the same NACK again. There is also a maximum number of retries for the same NACK, because maybe the original sender moved away, deleted the message from its buffer, or had any other major poblem, and it would be useless to continue trying to get the message. (see figure 3.8)

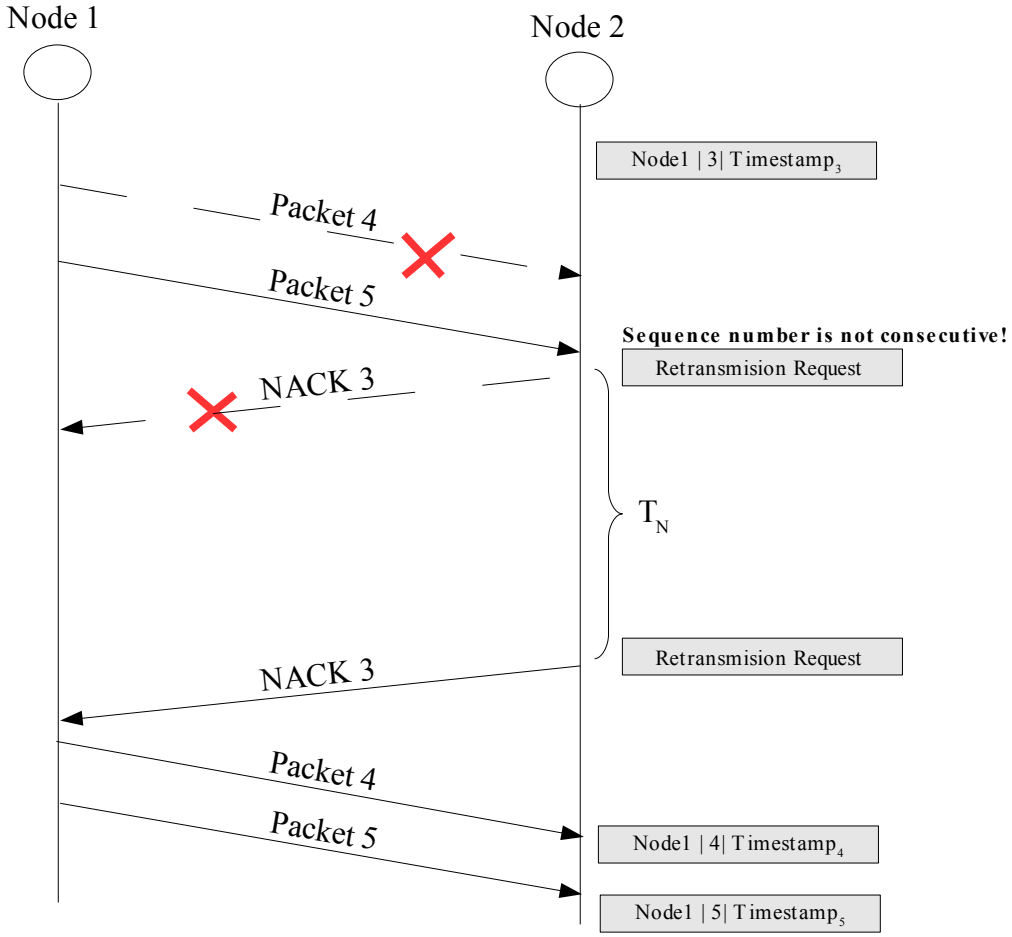


figure 3.8: Flow diagram showing the message exchange when a NACK message is lost

3.4.3 MAC generation and message ciphering

The MAC code is a serie of bits generated from the message plain text using a secret key K1 and an special encryption algorithm. We will be using OMAC (One-key CBC MAC) [22] because it is free to use, allows securing messages of any bit length, and it is simpler than CBC MAC while conserving much of its efficiency, which makes it a good choice for WSN applications.

We need to choose a cipher algorithm to use it during the OMAC generation and in the text ciphering. We will use the Skipjack encryption algorithm for both the message ciphering/deciphering block and the MAC code generation, in order to re-use the code and make the protocol less heavy. In [6] the authors study which can be the encryption algorithm most suitable for WSNs: *"We surveyed other block ciphers to find one that is well suited for sensor networks. We found RC5 and Skipjack to be most appropriate for software implementation on embedded microcontrollers". "Although RC5 is slightly*

faster, it is patented. Also, for good performance, RC5 requires the key schedule to be precomputed, which uses 104 extra bytes of RAM per key."

Comparing it with other ciphers, according to [23], when the implementation of Skipjack is speed-optimized, it is one of the best algorithms in terms of code and data memory, and it is the best one in terms of encryption efficiency. When it is size-optimized, it is the best algorithm in terms of data and code memory.

Skipjack uses an 80-bit key to encrypt or decrypt 64-bit data blocks. This way, we know that K1 and K2 need to be 80-bits long.

To generate the OMAC, the algorithm splits the message in blocks of 64 bits, for more detail about how exactly the OMAC is generated, refer to [22] and [24]. The final output of the algorithm will be a MAC code of 64 bits.

The length of the MAC code is a delicate parameter: if the MAC code is too short, an attacker could guess which is the MAC by simply trying all the possible combinations of bits (that would be 1 out of $2^{\text{MAC length}}$ possibilities), but if it is too big the overhead introduced in every message where we have to send the MAC code would be too much for a sensor network. In [24] they conclude that for mantaining the minimum level of security, the MAC code should be at least 64 bits long, and for this reason 64 bits is the length we will use, in order to keep the lowest overhead possible.

To encrypt the message we will also use the Skipjack algorithm, introducing as an input the desired message. The only requirement that the message has is that its length must be an integer multiple of 64 bits. The other input to the ciphering block is the secret key, which will be 80 bits long. As explained in previous sections, to encrypt the message we will use the secret key K2 and the sequence number, so instead of using directly the 80-bit key K2, we will calculate the exclusive or operation (XOR) bit by bit between the 16-bit sequence number and the last 16 bits of K2, leaving the rest of the bits of K2 unchanged, and we will use this result as a new key for the encryption algorithm.

3.5 Study of the protocol requirements

3.5.1 Memory requirements

The sensor nodes used tipically in WSNs have limited storage capacity, and for this reason we must be extremely careful with the amount of memory used by our protocol, if we want to be sure that it can be implemented in the real world.

This is the data that our protocol needs to store, and the size that each of them will need:

- K1 and K2: 80 bits each.
- Neighbor database: Every entry in the database will have 3 fields:
  - Neighbor node identifier: 16 bits.
  - Last sequence number received correctly: 16 bits.
  - Timestamp: 32 bits

  (Total bits per database entry: 64)
- Transmitted packets buffer: Every packet in the buffer will consist in:

- Sequence number: 16 bits
- MAC code: 64 bits
- Ciphered message: 32 Bytes

(Total Bytes per packet in the buffer: 42)

- Memory required by the Skipjack cipher block: 56 Bytes. (from [23])

K1 and K2 are 10 Bytes each because that is how it is defined in the Skipjack encryption algorithm.

Each node in the network is assigned a unique identifier of 16 bits, which provides a space of identifiers large enough without danger of running out of them.

According to [3] the average size for the packets is 30 Bytes, but the cipher block input must have a length multiple of 8 Bytes, and this is the reason why we use a message size of 32 Bytes. In this 2 extra Bytes added we will include the sender node identifier (2 Bytes).

The number of inputs in the neighbor database will depend mainly on the node density, while the frequence the inputs are renewed will depend on the nodes movement speed. We will suppose that the average number of neighbors that a node will have will be around 5, and as we have strong storage constraints, we can create a database with just only 20 entries, in order to be sure that even the nodes with higher number of neighbors will have space enough. Of course this is a customizable parameter that could be easily updated if the node density changes.

About the number of packets kept in the retransmission buffer, it should coincide with the threshold *Th* we fixed previously in the receiver node, that will decide whether a gap in the sequence numbers is too big for asking for retransmissions. To decide the value of *Th* we need to trade off between the need of reliable transmissions, the storage space needed, and the amount of energy consumed:

- if *Th* is high, the amount of memory needed for the retransmission buffer will also be bigger, and in the case that a receiving node asks for retransmission of a huge amount of packets, a lot of energy will be wasted retransmitting packets that probably did not arrive because the nodes were moving, but that could have perfectly made this way to the sink station through other neighboring nodes.
- On the contrary, if *Th* is too low, there is higher risk that when a node asks for retransmission of missing packets, the packets have alredy been erased from the original sender buffer, and the reliability level will be lower.

More especifically, if our protocol was implemented on a MICA2 mote, with 4kB of free memory, our protocol would work this way:

10 Bytes (K1) + 10 Bytes (K2) + 20 * 8 Bytes (database) + 56 Bytes (cipher) = 236 Bytes

Remaining memory = 4kB – 236 Bytes = 3860 Bytes

3860 Bytes / 42 Bytes = 91 possible packets in the retransmission buffer.

This is the maximum number of packets that could be stored in the sensor buffer in the worst case, but normally it will be lower because we are not interested in using all of the

memory avaiable in the sensor node. For example, in our implementation we only use 20 entries in the retransmission buffer, which means a total memory capacity needed of 1076 bytes (approximately 1KB).

About the code memory, the MICA2 mote has 128kB of program flash memory, while one of the heaviest blocks of our implementation, which is the ciphering block, takes only 2610 Bytes (≈ 2.5 kB).

We conclude then that in terms of memory storage, our protocol is perfectly feasible on the current sensors present in the market nowadays. (See section 2.5)

3.5.2 Overhead study

We will evaluate the effect that implemeting our security protocol has over the extra bytes that will need to be transmitted. We have added some overhead on each packet. These additional bits will be used to carry out the most of security requirements. The main drawback that this has is that longer packets need more energy to be transmitted.

As shown in table 3.3, the proposed protocol has an overhead of 10 bytes. The MAC code adds 8 bytes to the message and the sequence number adds 2 more bytes. Both are required to provide authentication, integrity and reliability. The cipher does not add overhead because encrypted message size is the same as the size of the plaintext.

In the following table 3.3 we compare the overhead needed to transmit a packet with our protocol and with SNEP.

	Payload	**Packet Overhead**	**Security Overhead**	**Total Size**
SNEP [25]	24 Bytes	20 Bytes	8 Bytes	44 Bytes
Proposed protocol	30 Bytes	12 Bytes	10 Bytes	42 Bytes

Table 3.3: Overhead comparison between SNEP and our protocol

3.5.3 Computational cost

The computational cost in an algorithm depends on the processor used to execute the program and on the number of operations in the code.

The proposed protocol has two mechanisms to achieve the most of security requirements, message authentication codes (MACs) and the cipher block. In ur protocol, Skipjack is used for both tasks, so we will only analyze the computatinal cost that it has.

In a block cipher the most important parameters are the key length, the block size and the number of rounds. In the proposed protocol we have used Skipjack with the following parameters:

Skipjack	
Block size	64bits
Key length	80 bits
Rounds	32

Table 3.4: Skipjack Parameters

We have used as reference [23] to examine the impact that the cryptographic algorithm has on a sensor node. In this study the computacional cost of a block cipher is evaluated by means of its energy consumption, asumming that the energy per CPU cycle is fix. In it, the authors use a MSP430F149 microcontroller, where theoretically the energy per instruction (CPU cycle) in this device is 1.26 nJ.

Note that this amount of energy (1.26 nJ) is much smaller than the energy used to send or receive a Byte, as shown on section 2.6. Even if we used another microcontroller, the order of magnitud of the energy will be more or less the same compared with the energy used to transmit signals, and this demonstrates that the main issue concerning constrained energy usage will always be the radiocommunication, and not the algorithm processing.

In [23] the authors have measured the CPU cycles necessary for the CBC encryption mode with different ciphers, and in the case of Skipjack it needs 550 cycles per byte when the implementation is size-optimized, and it needs 250 cycles per byte when it is speed-optimized. Comparing it with other ciphers, when Skipjack is speed-optimized, it is one of the better algorithms in terms of code and data memory, and it is the best one in terms of encryption efficiency. When it is size-optimized, it is the best algorithm in terms of data and code memory.

4 Simulation

4.1 Simulation objectives

During our simulation experiments we will try to study the performance of our protocol, more specifically we will focus on the following parameters:

- Number of packet retransmissions: In a lossy environment like ours some packets will get lost due to interferences and noise. We will simulate how our protocol detects these events and how it makes that the receiver asks the sender to retransmit the packet. Even if this will obviously spend more energy from the nodes, it is necessary for a reliable communication. We can also study the relationship between sent packets, and received, lost, or retransmitted packets.
- Error detection: The packets in a wireless medium can be altered by attackers or simply due to noise in the channel, as it is an insecure medium. We will simulate alterations in the packets like for example errors in the MAC or in any other of the packet fields. We will compare the number of wrong packets sent with the number of wrong packets detected. The goal is to check that our protocol is secure and that it is able to detect corrupted packets.
- Energy consumption: in WSNs, the energy is probably the most constrained resource, and every action that a node needs to make should be carefully planned in order to avoid wasting it unnecessarily. We will give the nodes a fix energy at the beginning of the simulation, and we will decrease it after every action they make. When the energy from a node falls below a threshold, the node will stop working, and we will have to remove it from the simulation.

4.2 Simulator choice

Considering what simulator to use, we have studied several options:

- TOSSIM.

  It is a simulator/emulator from the operating system TinyOS. It compiles code written for motes using TinyOS, and outputs it as a file executable in a normal PC. It is only valid for applications using this operating system, and works at very high detail level, emulating the motes characteristics.

- NS-2.

  It is probably the most common tool for simulating networks. It has been very used in simulating wired networks but has also been updated with some funcionalities to support wireless networks and even some WSN characteristics. However, when used to model WSN it presents some drawbacks, as those described in [26].

  NS-2 is object-oriented and this introduces a lot of interdependency between modules that complicates the task of adding a new protocol. In [26], the authors have analyzed the possible drawbacks of NS-2, and the y concluded that: *"Such interdependency sometimes makes the addition of new protocol models*

extremely difficult, only mastered by those who have intimate familiarity with the simulator. Being difficult to extend is not a major problem for simulators targeted at traditional networks, for there the set of popular protocols is relatively small. For example, Ethernet is widely used for wired LAN, IEEE 802.11 for wireless LAN, TCP for reliable transmission over unreliable media. For sensor networks, however, the situation is quite different. There are no such dominant protocols or algorithms and there will unlikely be any, because a sensor network is often tailored for a particular application with specific features, and it is unlikely that a single algorithm can always be the optimal one under various circumstance."

Even if there exist a lot of examples and references about how to use the main features of NS, we found out that the amount of information available about how to implement a protocol for a wireless broadcast link layer with all the constraints from WSN was very scarce.

- A new simulator developed by ourselves that could be able to simulate our protocol in a WSN scenario.

We have finally chosen to implement our own simulator with specific characteristics related with what we want to measure.

One of the main reasons has been that a simulator implemented by ourselves would allow us to have more control and flexibility to add the different factors which affect the data transmission in WSN environments, and in this way we could achieve a simulation in a realistic environment. We will model the differents losses in a wireless channel and the main characteristics of the current sensors (transmission power, reception power...). We will evaluate the behaviour of our protocol in different scenarios.

By implementing the simulator ourselves, we can start analyzing the protocol behaviour inside the network starting from the basics, and incrementally increasing the amount of detail used in the simulation. We can study the performance of the protocol in a simple scenario and analyze the results obtained, and in this way if there is any problem with the results obtained we can identify more clearly which is the problematic part, and if it works good we can proceed to the next step, adding a new characteristic, and studying what are the effects and how is the performance affected.

Also, as we want that our protocol can be implemented in real life sensors, we have decided to implement our own simulator so that we can know for sure how the protocol and the simulator are working together. We need to know how the simulator works in depth to ensure that the results obtained are not affected by the limitations of the simulator, because this protocol should be applied to real sensors in the future.

4.3 Simulation settings

To start with our simulation we will need to define the following parameters:

- Size of the area: The sensor network will be deployed in a two dimensional field. For a correct performance, the size of this area should be correlated with the maximum transmission distance and the number of nodes. In other case, this could lead to a shattered network without complete connectivity.
- Number of nodes, or node density if it is expressed related to the area size.

- Simulation run time: amount of time during which the simulation will take place. Another option is to define an event that would make the simulation to end, like a maximum number of packets sent.
- Movement pattern: We consider our nodes to be mobile, so we will need to define the position for each of them in every moment.
- Energy level: Each node will have defined an energy level that will decrease during the simulation.

For every node we will also need to define a unique identifier, the initial position, and other parameters. In principle we will consider all the sensor nodes to be equal. They will have the same capacities: memory, CPU, and initial energy level.

Other important parameters that we need to define carefully in order to achieve a realistic simulation are the power consumption scheme, the channel model that will introduce signal strength loss, and the radio reception model that will translate this signal loss into packet loss rate.

4.3.1 Power consumption

As explained in the section 2.6 the nodes consume energy when they are transmiting, receiving or idle. In our simulator we will not consider the power consumption when the nodes are idle because it affects every node in the same way and it would not be a differential factor. Initially, all nodes are working with their batteries completely charged.

In our simulator we use as reference the power consumption data from the CC1000 radio device because it is the one used by MICA2 motes (see section 2.6) The total energy costs depend on the number of sent or received bytes in each packet. In our case it is related to the type of message:

Sent /received	Cost (Joules/bit)	Message Type	Length (bytes)	Total cost/packet (Joules)
Sent	$26 \cdot 10^{-7}$	Sensor readings	42	$8.73 \cdot 10^{-4}$J
		NACK	4	$8.32 \cdot 10^{-5}$ J
Received	$15.1 \cdot 10^{-7}$	Sensor readings	42	$5.07 \cdot 10^{-5}$J
		NACK	4	$4.83 \cdot 10^{-5}$J

Table 4.1: Power consumption of the implemented protocol

During the simulation the nodes will decrease their energy level every time they receive or transmit a message, according to table 4.1. If a node runs out of all its available energy, it will no longer appear like a working node in the network.

4.3.2 Channel model

The implemented simulator allows to emulate a wireless channel using the log-normal shadowing model (see section 2.3.4). In order to use this model it is necessary to establish the value of some variables.

In the simulator the default values are initialized as showed below, although they could be easily modified in the simulator configuration file to analyze the influence of these parameters in the network performance:

- P_t = 16.98 dBm (50 mW, see table 2.7)
- G_t and G_r = 0 dB
- λ = 0.34 m (868 Mhz)
- n = 3.5 (outdoor, shadowed urban area)
- σ = 8 dB
- d_o = 1 m

4.3.3 Radio reception model

As explained in section 2.4, the probability of succesfully receiving a packet in the receiver node depends on different factors that can be englobed in the equation 2.12, which defines the PRR, or Packet Reception Rate.

For MICA2 motes [19] the values of the variables present in the PRR equation are: R_b= 19.2 Kbps, B_N = 30Khz. When inserting this values in equation 2.12, the PRR is given by:

$$PRR = (1 - \frac{1}{2} \cdot \exp^{\frac{SNR \cdot 1}{(2 \cdot 0.64)}})^{8 \cdot L} \quad (4.1)$$

The value of L depends on the type of message. In our protocol there are two different kinds of messages which are showed in the following table:

Type Message	Length (bytes)
Sensor data	42
NACK	4

Table 4.2: Values of L used to calculate the PRR

The only parameter left to calculate the PRR is the signal to noise ratio (SNR), which should be calculated using the received signal power and the received noise power. For the received noise power we have used the nominal value of noise power for Mica2 motes, referenced from [20], which is Pn = -105dBm ($3.16 \cdot 10^{-11}$ mW). The received signal power is calculated using the wireless channel model described in 2.3.4, and it will depend on the transmitted signal power, the distance between both nodes and other parameters explained previously.

4.3.4 Traffic model

In WSNs the generated traffic can be very different depending on the applications. Some of them might just need to send a few packets in long periods of time, like those used to monitor slow-changing variables, while other applications need high data rates sent continously.

In our simultations we have implemented a generic traffic model that is based on event detections. Each time an event is detected by a sensor node, it will send a data packet.

The average data rate is a configurable parameter in the simulator, as well as the deviation. The traffic generator of every node always waits for a time and then sends a single packet. This time is distributed randomly between a minimum and a maximum waiting time. By changing this limits we can set the average packet sending rate and the standard deviation.

If we set both the maximum and the minimum waiting time to be the same, we would have a pure constant bit rate, and in any other case we will have defined an average bit rate.

4.4 Implementation

In this section we will explain how our implemented simulator works. It is based on a multi-process solution that simulates the communication between nodes through inter-process communication.

First, we have a main process that will simulate the channel. When any node wants to send a packet to its neighbors, it will just send it to the channel process. The channel process is also the only one aware of the exact location of the sensor nodes, and it will use this information to decide what nodes should receive a specific packet.

This process will compute for every node in the network a function that simulates the probability of packet loss depending on the distance between 2 nodes, and it will forward the received packet to every node with this calculated probability. If this function depends on the distance between both nodes, the result will be that the packet will be forwarded with higher probability to nearby nodes, and with very low probability to the nodes that are placed further away.

Every node process will communicate with the channel process using 2 “pipes”. A pipe is a inter-process communication structure consisting on a file descriptor shared by both processes, and in which one of them will write and the other will read. We use 2 pipes so that the node can talk to the channel and the channel can talk to the node.

Every sensor node will consist in a separate process that will at the same time have another sub-process that generates random data traffic (see section 4.3.4, “Traffic model”). The data traffic is sent to the main process of the node so that it can store it in the buffer, and then it is sent to the channel. For this communication between the traffic generator process and the node's main process we will use another pipe structure.

It is over this platform that we will build our protocol. It is specifically designed for wireless broadcast environments, where every byte sent is heard by all the nearby nodes. The protocol built over it will be responsible of determining whether the received packets should be processed by the node or discarded.

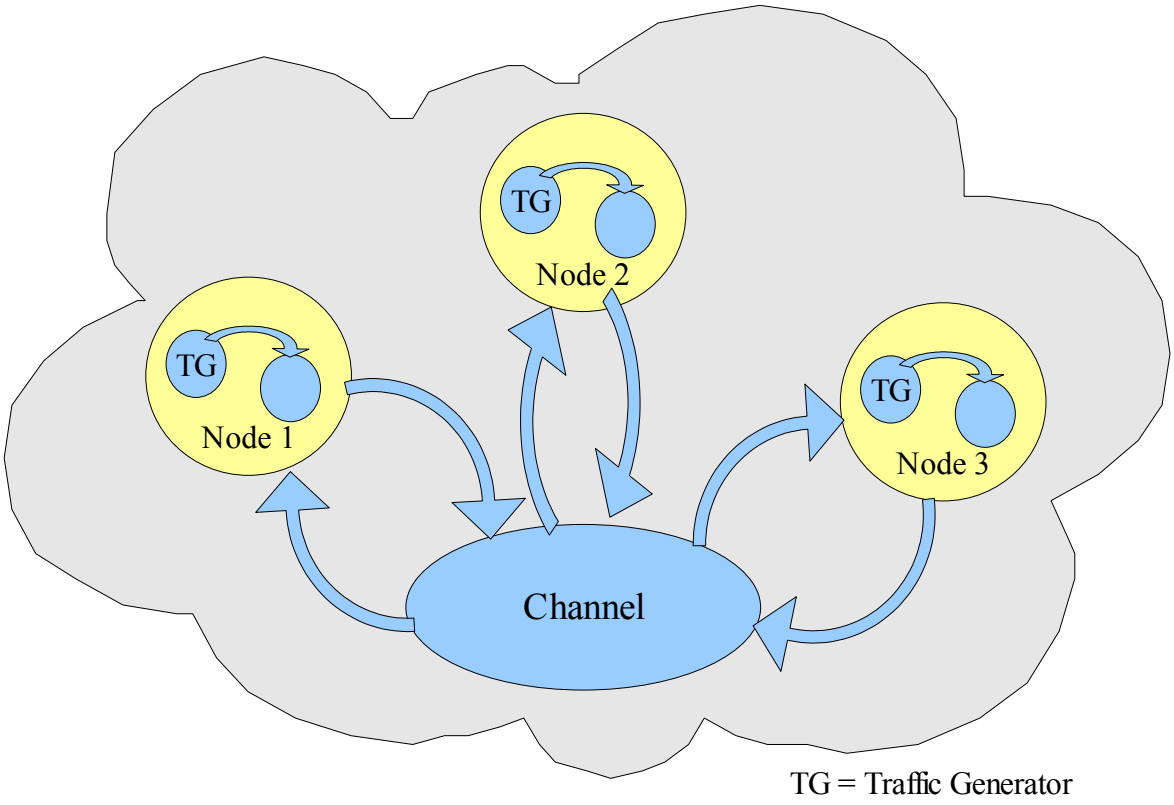


figure 4.1: Simulator structure: inter-process communication

In the figure 4.1 it can be seen the explained structure with all the processes created and the "pipes" used to communicate them.

4.5 Results

We will divide the simulation in several stages, going from the most ideal scenario to the most complex, adding new measurements in every step.

These are the steps we will follow during the simulation:

1. Perfect channel: there are no packet loss, every message from a node will reach all of the other nodes.
2. Constant loss: every packet sent by a node will reach each one of the other nodes with a fixed probability *p*.
3. Distance-dependent loss: every message sent by a node will reach each one of the other nodes with a probability that will change with the square of the distance between both nodes. The nodes will be able to move, we will define for every node dynamic x and y coordinates.
4. Packet Reception Rate calculated using log-normal shadowing model: more realistic losses modelled as explained in section 2.3.4.
5. Same as above, but introducing corrupted packets in the network to test the protocol security features.

6. Simulation of the energy drain on the sensor nodes, substracting the required amount for every packet sent or received.
7. Health care appplication: we will analyze the behaviour of our protocol working in a real scenario within a hospital.

4.5.1 Perfect channel scenario

In this scenario we will not introduce packet loss yet, and every message sent by a node will reach all the rest of the nodes in the network. There is no need to establish the position of the nodes, as the model does not depend on the distance between nodes.

As explained in section 4.3.4 each node has a traffic generator which broadcasts the sensor readings to every other node. In a real scenario, the sensors do not sent the readings with a constant rate, there exist some events which cause the packet sendings. To reproduce this behaviour in the simulated nodes we have introduced a maximum and a minimum time between different packet sendings, and the traffic generator will wait a random time between this constraints before sending a new packet. The data rate showed in these results is deduced from the average waiting time between packets.

The main purpouse of this simple scenario is to check if the protocol is working correctly, and if a receiver node is able to understand the packets that other nodes are sending. As there is no packet loss, there will be no NACK messages and no retransmissions will be needed, we can not measure this behaviour in this scenario.

After implementing this scenario, we analyzed the total number of sent and received packets, and checked that they were exactly the same, as we should expect (see table 4.3).

[i]Average data rate = 2 packets/sec Simulation run time = 5 minutes *Channel without loss*			
No. Nodes	**No. sent packets**	**No. received packets**	**No. NACKs**
5	13163	13163 (100%)	0
20	236944	236944 (100%)	0
30	528650	528650 (100%)	0

Table 4.3: Results in ideal scenario

We can see that the number of generated packets could be obtained aproximatedly by multiplying the data rate by the simulation time, the number of nodes in the network, and the number of neighbors that each node has.

i) Data rate: The number of packets sent to the channel per unit time. In a wireless medium, the messages are broadcasted to every node. To obtain the data rate of the whole network we should multiply the number showed in table 4.3 by the number of nodes in the network, and by the number of neighbors that every node has, in order to obtain the real data rate in the several environments with different number of nodes.

4.5.2 Constant loss scenario

In this situation the channel will drop messages with a fixed probability. If a receiver node detects a packet with its sequence number out of order it will send to its neighboring nodes a NACK message in order to recuperate the lost packets. The nodes do not have a fixed position in this scenario because the packet reception is independent of the distance between nodes.

The main objective of this scenario is to check if the NACK packets are sent correctly and if the receiver node can understand them and retransmit the missing packets. It is used to simulate in a very simple scenario the loss of some packets and their further retransmission.

In the table 4.4 they are showed the results of a simulation with a channel which has a constant loss of the 25%. We analyze the number of sent packets and NACKs, and also we check the number of successfully received packets, despite of the channel loss.

Average data rate = 2 packets/sec Simulation run time = 5 minutes *Channel with Ploss=25%*										
No. nodes	**No. Packets**						**No. NACKs**			
	Sent			**Channel Loss**	**Received correctly**	**Lost Packets**	**Sent**		**Channel Loss**	**No of times reached the maximum number of NACK retries**
	Generated	**Rtx**	**[i]M**				**Generated**	**Rtx**		
5	12798	9248	3,67	5449 (24.72%)	12684 (99.11%)	114 (0.88%)	4954	936	1425 (25.50%)	43
30	538116	1599238	4.91	536689 (25.11%)	2088622 (97.72%)	48732 (2.28%)	316029	1594987	473359 (24.77%)	210

Table 4.4: Results with P_{loss} 25%

In the highlighted cells on this table, we can observe that the number of correctly received packets is close to 100%. Although the channel drops the 25% of the packets (5th column), the sending of NACKs (8th column) allows the recovery of most of the lost packets by means of retransmissions (6th column).

Also, we can see how the number of retransmitted messages is greater than generated messages. It is due to the fact that the number of retransmitted messages when a NACK is received is usually more than one, see parameter M (4th column). This number M indicates the average number of retransmitted messages when a node receives a NACK.

Besides, when a node is waiting for the retransmission of a packet after having sent the apropiate NACK, it does not listen to any packet that is not the packet it is waiting for. This fact makes that even if the channel is not dropping these packets, the receiver node is ignoring them and it will eventually need that they are retransmitted too. Until the moment in which the missing packet is recovered, all the new packets sent by the original sender during that time will be discarded by the receiver node, and they will be retransmitted later, making the number of retransmitted packets to grow.

i) M indicates the average number of retransmitted messages when a node receives a NACK

Another interesting point is that the number of retransmitted NACKs is greater than the number of lost NACKs. Even if this fact could look strange at the beginning, it is a perfectly normal situation, as the protocol does not only retransmit a NACK when the original NACK message was lost, but also when the data retransmission is lost in the channel.

Finally, in the last column of table 4.4 it is showed the number of times that the maximum number of retries of sending the same NACK was reached. This limit avoids that a node is continously asking for the same retransmission during too much time. If the number of times that this situation happens is very high, the amount of successfully received packets will decrease because the NACKs will not be received by the nodes. In our simulation this limit was in principle established as 3. We can check that when the number of nodes is increased this limit is reached more often, and it will be correlated with the number of packets that are finally lost.

We have to mention that the numerical results of both this section and the section 4.5.1 can not be taken into consideration to study a realistic WSN scenario because both are ideal scenarios. These simulations have only been used to check the correct protocol behaviour and how the communication works between nodes.

4.5.3 Distance dependent loss scenario

The next step would be to introduce in the simulator the position of the nodes, and calculate the distance between each couple of nodes. Using this distance the channel will decide whether to forward the packets from one node to another or not. More specifically, we will simulate the effect of the distance by calculating the square of the distance, multiplying it by a constant and comparing it with a random number so that the probability of receiving a packet is high for nodes that are close together, and very low for nodes that are far away from each other.

Also, the position of the nodes will not be static, so the closest neighbors of a node will not always be the same.

The goal of this scenario is to study the effect produced by the fact that the neighbors in hearing distance from a node are changing through time.

We introduced in our simulator one node that would move from one of the corners of the network area to the opposite one, while the rest of the nodes stayed in fix positions all over the area. We then configured the simulator to trace the node identifiers of the neighbors of the moving node so that we could see how the neighbor table was being updated.

The trace result is too large to show it here, but we could observe how at the beginning all the positions of the table were being filled until they reached the maximum number of neighbors that a node can keep in its database (20 in this experiment). From this moment we could see how the table was updated, changing one of the entries of the table each time. The entries that changed were not necessarily the ones that were filled in the first place. Apparently we could not predict which table entry was going to be the one to be updated, but this is because the entry that is overwritten once the table is full is not the one that was created first but the one that was updated longer time ago.

We could see that sometimes nodes that had been removed from the table appeared again some time later because thay had sent a new message to the moving node, but if

we compared the entries from larger amounts of time, comparing the entries at the beginning with those at the middle of the node trajectory or at the end, we could see that almost all the nodes in the table were different in the 3 cases.

4.5.4 Realistic scenario

This is the scenario in which most of the tests should be carried out. We will simulate a realistic channel using the parameters described in section 4.3.3. For every couple of nearby nodes we will calculate the packet reception rate (PRR) that will give us an estimate of the probability of successful communication between both nodes.

If a link has a very high packet reception rate it will indicate that the communication between these nodes is reliable.

Before starting the simulations we have established the following parameters:

- Size of the network. It will be changed depending on the number of nodes in the simulation. We want to mantain a similar node density in all the tests so that the average number of neighbors of a node remains constant.
- Maximum number of retries for the same NACK. It is in principle set to 3.
- *Th*, threshold for retransmitting. As was explained in the section 3.4.2, if the received sequence number and the last sequence number received correctly differ in more than *Th* positions, the receiver node will not ask for retransmissions because it would be very inefficient. This threshold is set at 20 because the results obtained with this parameter were suitable for a generic scenario.
- Size of the buffer to store messages for retransmitting. Fixed to 20 so that it coincides with the *Th* threshold, as explained in section 3.5.1.
- Number of possible inputs in the neighboring table. It has a maximum of 20 inputs, as explained as well in section 3.5.1.
- Initial position. It is set at random, the nodes could be placed anywhere in the network area.
- Movement pattern. Every node moves its X and Y coordinate in ±1, ±2 or 0 meters randomly with respect to its previous position.
- Transmission power. It is 16.29dBm. This parameter is decided by the channel model. The details about the log-normal shadowing model that is used are in section 4.3.2.

Average data rate = 2 packets/sec Simulation run time = 5 minutes *Channel log normal shadowing model*										
No. nodes	**No. Packets**						**No. NACKs**			
	Sent			**Channel Loss**	**Received correctly**	**Lost Packets**	**Sent**		**Channel Loss**	**No of times reached the maximum number of NACK retries**
	Generated	**Rtx**	**M**				**Generated**	**Rtx**		
5	11398	16437	2,87	10728 (38.54%)	27724 (99.60%)	111 (0.39%)	5394	2279	1884 (24.55%)	38
50	1497466	433701	4,31	717428 (37.15%)	1777639 (92.05%)	153528 (7.94%)	974315	419798	406523 (29.16%)	629
100	59625380	24613447852	5,02	8677519855 (35.17%)	18679983746 (75.71%)	5993089486 (24.28%)	4192519479	1893946192	1583698368 (26.02%)	1590

Table 4.5: Results with realistic model

In the table 4.5, we can observe that the channel drops more data packets than NACKs, which corresponds to the expected behaviour because we have used a reception model (see equation 4.1) which depends on the size in bytes of the message, L. Longer packets have less probability of reception. We remember here this equation:

$$PRR = (1 - \frac{1}{2} \cdot \exp^{\frac{SNR \cdot 1}{(2 \cdot 0.64)} \; 8 \cdot L})$$

In the implemented protocol the length of the data messages is higher than NACK messages, 42 bytes and 4 bytes respectively. Due to this fact the nodes will receive with more probability the NACK messages than the data packets.

Also, the table 4.5 shows that the number of successfully received packets falls for the case with 100 nodes in the network. This is because we are reaching the limit in which the simulator starts being oversaturated with messages. The channel process has to listen to all the node processes at the same time and when there are a lot of them it stops working as it should.

4.5.5 Realistic scenario with energy consumption

In this scenario we will simulate the energy limitations in the current sensors. Depending on the type of device, the amount of consumed energy is different. We will use as reference the table 4.1. It belongs to a Mica2 mote.

Initially, we assume that each node has a battery with a capacity of 21600 Joules. Each time that a node receives or transmits a message its available energy will decrease in the amount indicated in the table 4.1. The initial energy is obtained with an approximate method: we know that the Mica2 motes use 2 AA batteries, and that these batteries have normally an electric charge of 2000mAh [27], then we can calculate:

$$2\,batteries * 2000\,mA \cdot h * 1.5\text{V} * 3600\,seg/h = 21600\,J \qquad (4.2)$$

The main goal of this simulation is to evaluate the lifetime of the sensors when they are using our secure and reliable protocol. We will study the extra energy comsumption which was generated by the sending of NACK messages and retransmitted packets.

In the section 4.5.4 they were already performed different simulations, we will repeat these simulations decreasing the amount of energy of every node each time they send or receive a message.

After a 5 minute simulation, the average consumed energy per node was about 7.5J. If we expanded the simulation for a longer time, we could see that the lifetime of a sensor with a capacity of 21600 joules was around 10 days. Notice that this result could change if we consider that we could use any other model of sensor node different from the Mica2, or if we change the data rate from the sensors.

If we look more in depth at where this energy is being spent, we can see the percentage of energy that was used for every kind of packet transmitted or received. In the following table we show the results like percentages:

Type of message		**No. messages**	**Cost (Joules)**	**Consumed Energy percentage**
Generated	**Packets**	2280	1,988	26,53%
	NACKs	1079	0,090	1,19%
Retransmitted	**Packets**	3287	2,867	38,26%
	NACKs	456	0,038	0,50%
Received	**Packets**	4839	2,454	32,75%
	NACKs	1158	0,056	0,74%
Total for each sensor			7,492	100%

Table 4.6: Consumed Energy

In the above table we can check the costs of energy associated with our protocol. About 39% of the consumed energy in the sensors is due to retransmitted messages and a 2% due to the NACKs. However both are necessary to provide the reliability required.

Average data rate = 2 packets/sec Simulation run time = 5 minutes Channel log normal shadowing model									
No Nodes	data packets				NACKs				Total Energy Consumption (Joules)
	Sent		Received		Sent		Received		
	Total No. of sent packets	Energy (Joules)	Total No. of received packets	Energy (Joules)	Total No. of sent NACKs	Energy (Joules)	Total No. of received NACKs	Energy (Joules)	
5	27835 (16437 rtx)	24.3	24194	12.27	7673 (2279 rtx)	0.63	5789	0.27	37.47

Table 4.7: Results with realistic model and energy consumption

We have described in the previous table 4.7 the energy consumption that have the five sensors during a 5 minute simulation. From the 37.47J spent, 97.59% of that energy is spent in data messages, while the 2.40% is spent on NACK messages.

In the received packet count we are also including data packets that are discarded (because they have old sequence number for example), as well as the NACK messages that are received by nodes that are not the intended destination of that NACK message.

We observe that the highest energy consumption is mainly due to the sending of data packets, both retransmitted and generated. It is a expected behaviour because in the table 4.4 we can check that the cost of sending a data packet is higher than sending a NACK message. In this way, we show that the adding of NACK messages in our protocol does not affect greatly the nodes energy consumption.

4.5.6 Comparison with SNEP

In this section we will compare the proposed protocol with an existing protocol like SNEP. For more information about SNEP see section 2.1.1.1.

We will analyze the reliability of each protocol and its energy consumption. We have chosen to study these two parameters because we have considered that they are some of the key factors in wireless sensor networks.

We simulate the following scenario with the two protocols:

- Number de mobile nodes. We have deployed 5 nodes in a network with a size of 400x400m.
- Average data rate. 2 packets/sec.
- Simulation run time. We have run the simulation during 5 minutes.
- Channel model. It is used the log-normal shadowing model.

4.5.6.1 Reliability

The simulation results related to reliability are showed in the two following tables:

SNEP Average data rate = 2 packets/sec Simulation run time = 5 minutes *Channel log normal shadowing model*							
No. nodes	No. Packets						No. NACKs
	Sent			Channel Loss	Received correctly	Lost Packets	
	Generated	Retransmitted	M				
5	12783	0	-	5226 (40.6%)	7590 (59.38%)	5226 (40.6%)	0

Table 4.8: Results with SNEP

PROPOSED PROTOCOL **Average data rate = 2 packets/sec** **Simulation run time = 5 minutes** ***Channel log normal shadowing model***										
nodes	**No. Packets**						**No. NACKs**			
	Sent			**Channel Loss**	**Received correctly**	**Lost Packets**	**Sent**		**Channel Loss**	**No of times reached the maximum number of NACK retries**
	Generated	**Rtx**	**M**				**Generated**	**Rtx**		
5	13250	49816	4,35	22767 (36.10%)	62744 (99.49%)	322 (0.50%)	11912	52718	1472 (22.75%)	41

Table 4.9: Results with realistic model – 5 nodes

In the following graph we have represented together the results from both protocols:

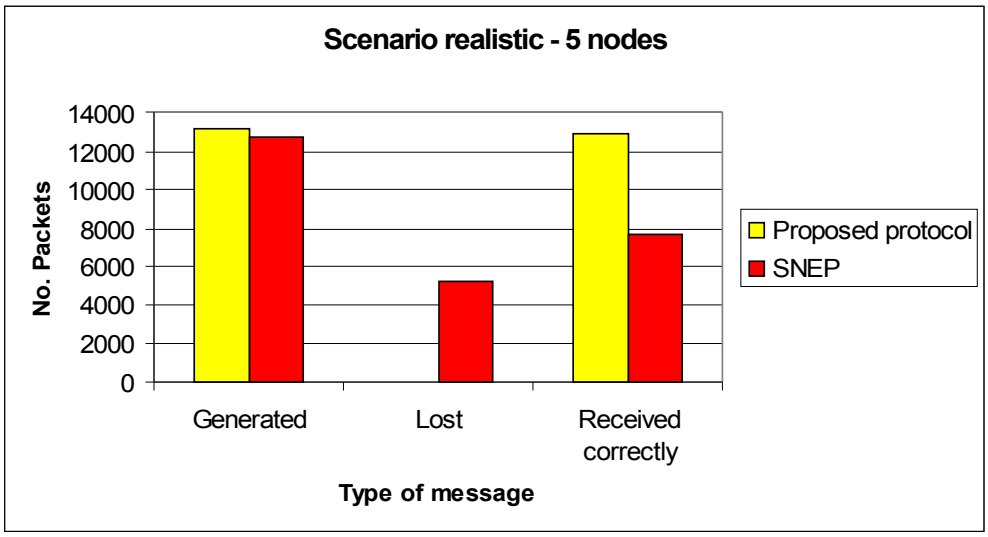


figure 4.2: SNEP/Proposed protocol comparison

At first glance we observe that the generated packets in the test with SNEP and with the proposed protocol do not coincide exactly. This is due to the fact that the traffic generator has a random component that produces that the data rate is not always the same for every node and that the number of packets sent in a time interval is never the same for different simulations.

We observe that in the case of SNEP the total number of lost packets coincides with the number of packets lost in the channel (see table 4.8). These lost packets can not be recuperated because in SNEP there does not exist any mechanism to do it. In the previous graph we can see that the lost packets in the proposed protocol are negligible compared to lost packets in SNEP, 322 and 5226 respectively.

In the table 4.8 and in the figure 4.2 we can see that the number of correctly received packets is much shorter compared to the values obtained in the simulation with our protocol (see table 4.9) where they were reached percentages of 99% successfully received messages.

After seeing these results we can conclude that a protocol without the reliability that the retransmissions provide consumes less energy, but it loses a lot of packets and could not be used in many applications where the loss of information could produce human damage, like in healthcare environments.

4.5.6.2 Energy consumption

As it was explained in section 4.5.5, it is established an initial energy in every node which will be decreased when the nodes transmit or receive packets.

After a 5 minute simulation with SNEP, and knowing that the packet size is 44 bytes [25] (a bit bigger than in our protocol), the average consumed energy per node was 7.49J. In the following table we show the results showing in detail where is the energy spent:

Type of message	No. messages	Cost (Joules)	Consumed Energy percentage
Generated Packets	2556	2.339	74.32%
Retransmitted Packets	0	-	0 %
NACK messages	0	-	0 %
Received Packets	1518	0.807	25.68%
Total for each sensor		3.146	100%

Table 4.10: Consumed energy per node - SNEP

The consumed energy using the proposed protocol is showed in the section 4.5.5 table 4.6. In this case the average cosumed energy per node was about 7.49 Joules. The proposed protocol produces an increase in the energy consumption of 238% with respect to SNEP. In the figure 4.3 we compare the consumed energy of the two protocols:

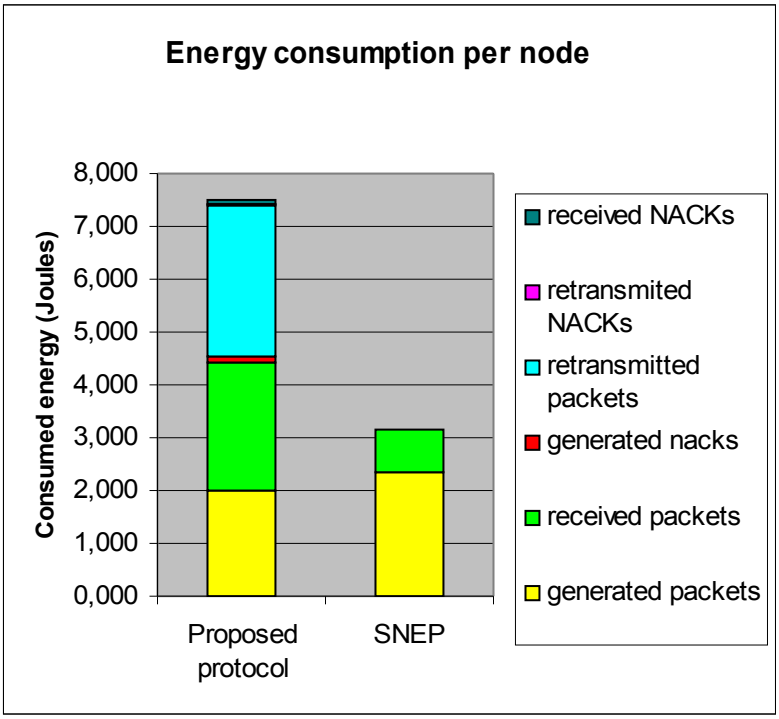


figure 4.3: Energy consumption comparison

In the bar chart in figure 4.3, we can observe that SNEP does not have any associated additional consumption for sending or receiving extra packets like NACKs or retransmissions. This protocol does not ensure reliability in the communication. In contrast, the proposed protocol has some additional energy consumption due to the retransmissions mechanism that has been implemented.

An interesting point is that the energy consumed by sending the NACK messages is very small (see red stripe in the first bar). It is due to the fact that the size of NACK packets was minimized to avoid wasting energy (4 bytes). However, the retransmitted messages produce a high energy consumption, represented in the blue stripe. This is an inevitable effect because the messages have to be retransmitted if we want to ensure the reliability in our protocol.

We have to mention that the proposed protocol has a threshold of retransmissions to avoid asking for retransmissions of old packets. This threshold could be modified depending on the reliability requeriments in the application. In the previous tests this threshold was established to 20. See section 3.4.2 for more information about this threshold.

4.5.7 Scenario with corrupted messages

In a realistic scenario the messages could reach the destination node being altered in some way. This problem can be due to different factors like attackers or noise in the channel.

We will simulate the sending of packets with incorrect packet fields i.e., errors in the MAC, sending of old sequence numbers and so on. We will analyze the behaviour of

the nodes when they receive this kind of messsages to check that the implemented protocol is achieving the desired levels of security.

We consider that these modifications in the packets can be due either to errors in the transmission, or to intentional attacks made by a malicious attacker. In any case the result is a message that is different from what it should be.

4.5.7.1 Corrupted MAC:

We have tested the protocol behaviour when we introduce at random times some packets with a wrong MAC code. The wrong MAC was created by just replacing the original MAC with some random bytes. The result was always that the packets were dropped in the receptor node, because it always checks that the received MAC matches the one that it computes from the text of the message.

Our simulator can count the number of packets dropped as result of having a wrong MAC code, and in all the simulations performed we checked that all of the packets that we introduced with wrong MAC were detected and dropped.

Data rate = 2 packets/sec Simulation run time = 5 minutes *Corrupted MAC*	
Number of wrong packets introduced	**Number of discarded packets because MAC is not matching**
79	316

Table 4.11: Results with corrupted MAC

To perform this simulation, we made that only one of the nodes sent at random times one packet with the erroneous MAC, and disabled the packet loss in the channel so that the corrupted packets were not lost and in order to be sure that all of them were detected in the receivers. The simulation was run with only 5 nodes. We can see that the number of discarded packets equals the number of wrong packets introduced multiplied by the number of receiver nodes (if there are 5 nodes, when one of them sends a message, it is received by the other 4 nodes and all of them discard the message).

4.5.7.2 Corrupted sender node identifier:

In our protocol, the sender node identifier is attached to the plain text and encrypted together with it, so in the packet that is finally sent it is impossible to differentiate which bits of the encrypted message belong to the text and which belong to the node identifier. If some bits are modified at random in the message body, the chances are that when it is decrypted in the receptor both the data and the node identifier will be corrupted at the same time. Thus, this case is contemplated in the next section, “corrupted data”.

Another possibility is that someone modifies the sender identifier before encryption. In this case the receptor node will consider this packet as coming from a new node, but we consider this case a very rare one, as the node creating the corrupted message would need to have access to the encryption key K2.

In our simulations, we have introduced both packets corrupted after and before ciphering. The messages that are corrupted after ciphering are part of the "corrupted data" case (next section). However, we tried also to make one of the nodes in the network to change from time to time its identifier in order to simulate the pre-ciphering corruption. In this last case we did not observe significant differences in the results obtained by the simulator, and this is because this "corrupted" node is in fact considered as two different nodes and it does not imply a great difference in the overall behaviour.

By tracing the complete set of messages exchanged in this case we saw that the most direct consequence of this change is that, as the sequence number is updated continously whatever the node identifier is, the receiver node detects gaps in the sequence numbers and asks for retransmissions of the packets that were sent with a wrong identifier, as they are considered as coming from another node.

4.5.7.3 Corrupted data:

In our simulation we introduced randomly some messages modifying part of the data payload just before sending the packet to the channel. The result was that these packets were dropped by the receiver node as result of non-matching MAC code. When a receiver node gets a new message it will decipher the data payload with the ciphering key K_2 and it will get both the sender node identifier and the plain text data. Then it computes the MAC code with this received data and checks if it is the same than the MAC it has received. If the ciphered message was modified, both MACs will not match and the packet will be discarded.

In our simulations we could check that all the messages sent with a modified data field were dropped by the receiver nodes, indicating that the MAC code was wrong

Data rate = 2 packets/sec Simulation run time = 5 minutes *Corrupted data*	
Number of wrong packets introduced	**Number of discarded packets because MAC is not matching**
86	344

Table 4.12: Results with corrupted data

We can see that, in the same way as previous sections, the number of discarded packets equals the number of corrupted packets multiplied by the number of receiver nodes (4 in this case, as there are 5 nodes in the simulation).

4.5.7.4 Corrupted sequence number

There are 2 possibilities in this case: that the sequence number is modified after the message has been created, or that the sender node introduces a wrong sequence number during the process of creating the packet.

In the first case, when we introduced in the simulation messages with wrong sequence number, we could see that all of them were dropped by the receiver nodes, indicating that tha MAC code did not match.

If we remember, the sequence number was also used in this protocol to encrypt the message. For this reason, if the sequence number is modified after creating the packet, when it reaches the receiver node it will try to decrypt the data using a different sequence number than the one that was used to encrypt it, and the deciphered data will be wrong. When the receiver node checks if the MAC code matches the received data it will conclude that the message has been altered and it will be discarded.

In the second case, we artificially made in our simulation that one node sent packets with random sequence number. This fact made the number of retransmissions to grow a bit in this node neighborhood and lead to a slightly worse network behaviour. Notice that in order to do this, the wrong packets sender needs to know the secret key, and this is a very rare case in which the security of our protocol has been compromised. However, the semantic security mechanisms present in our protocol make it very hard for an attacker to get the secret keys.

As the sequence number used for ciphering is the same that is sent with the message, the receiver node does not detect that the packet has been altered -because it has not, it has been wrongly created-. In our simulation we could see that if the wrong sequence number was lower than the previous one received, the packet was discarded and the trace of the simulator said that it was a replay of an old message. Notice that this is almost the same case as an adversary replaying old messages. However, if the sequence number was greater than the last one received, the receiver nodes sometimes asked for retransmissions of all the missing packets, which could lead to a slightly worse network performance.

We noticed that the receiver nodes did not ask for retransmissions in the case that the received sequence number was much greater than the last one received (greater than the previously defined threshold *Th*), so the cases in which extra retransmissions were required were fewer.

4.5.7.5 Old messages replay:

The replay of previous messages without altering them is also considered by the protocol. We introduced this effect in our simulation by making the process that simulates the channel to store some packets and replay them at random times.

The result was that almost all of these packets were dropped as consequence of having a sequence number smaller than expected. The only case in which these packets were not discarded was in special cases in which the receiver node had not heard the original message and was still waiting for the retransmission of that particular packet.

Data rate = 2 packets/sec Simulation run time = 5 minutes *Replay of old packets*	
Number of old packets replayed	**Number of discarded packets because of old sequence number**
73	2125

Table 4.13: Results with old messages replay

As we can see now both numbers does not match. The reason of this is that when a node sends a NACK message and the original sender retransmits the missing packets, those

retransmitted packets are also heard by nodes that had already received those messages before. For these nodes these packets are considered as replays of old messages and they will discard them. As we can not differentiate this 2 cases by just counting the number of discarded packets, we traced all the packets sent in the network, looking for this 73 replayed messages, and in this way we could check that almost all of them were being discarded by the receiver nodes, as we explained before.

4.5.8 Health care scenario

This scenario simulates the use of WSN for monitoring the status of patients within hospitals. In this case the sensor nodes are attached to the patient's body. This scenario is considered one of the worst cases in which WSN can be applied due to the high reliability level demanded in these environments.

WSNs in healthcare environments have some different characteristics than in other environments. To start with, the reliability and security requirements are much more strict and we will need to guarantee the correct reception of nearly 100% of the packets. Then, the physical deployment of the nodes is very different from other applications too, the nodes are much closer together, and there are fewer of them. The transmitting power needs to be reduced and this makes that the transmission range of a sensor node is also decreased.

Based on the information present on [28], [29] and [30], we will use the following parameters to simulate a health care environment:

- Number of nodes: 30
- Size of the area in which sensors are deployed: 1.5m x 1.5m
- Transmission power: -30dBm (1μW)
- Transmission range: around 10 m (obtained experimentally from the transmission power)
- Data rate: 10 packets per second

We simulated our protocol with these parameters and obtained the following results:

Average data rate = 10 packets/sec Simulation run time = 10 minutes *Health-care environment*			
No. nodes	**No. generated packets**	**No. packets received correctly**	**No. NACKs**
30	1377297	1377297	0

Table 4.14: Results in a typical health care environment

We can see in these results that due to the proximity of the sensors in this environment, in our simulation all of the sent packets are received in the first transmission. There are not any lost packet in the channel and this causes that the retransmission system is not needed, there is not any NACK message sent because no retransmission is needed. The reliability of this system covers the 100% of the packets sent.

About the security aspects of the protocol, we consider that all the results from the tests that the protocol passed in section 4.5.7 are still valid in this scenario, and conclude that the desired levels of security and reliability are successfully achieved by our protocol.

5 Conclusion

WSNs present many problems when trying to provide security and reliability due to its limited resources. However, the use of WSNs continue to grow and to become more common in all types of environments.

When we analyzed the current security protocols for WSNs we detected that reliability in the packet reception was not taken into consideration, and we considered it a necessary feature that should be added.

In this project we have proposed a secure and reliable protocol for WSNs. This protocol takes into consideration aspects like message integrity, authentication, authorization, self-organization, semantic security, data freshness, and reliability, all of it taking into account the known limitations of power and memory that the current sensor nodes have. The protocol will work in networks with mobile nodes using always self-organized mechanisms.

Our aim is to provide a security protocol that can be used in different applications which will require reliability, such as healthcare applications, where packet loss can be critical.

We have evaluated the performance of the proposed protocol using a simulator implemented by ourselves and set it in different scenarios. The simulation results have shown that the designed mechanism to achieve reliability is working correctly. It successfully recovers most of the packets that were lost by means of asking for retransmissions. We have proved that it could provide the demanded reliability level in many of the current applications, like in healthcare systems where the security and reliability is very important.

The simulation has also showed that the protocol can detect messages that were altered either intentionally or by errors in the reception. We have analyzed the effect that would have an alteration in each of the message fields, and checked that the packets were discarded in the receiver nodes.

After analyzing the energy consumption of the nodes using our protocol, we conclude that the adding of NACK messages in our protocol does not affect greatly the nodes' energy consumption. However, the retransmission of missing packets consumes some extra energy that is necessary to spend if we want to mantain this reliability level.

Furthermore, this protocol satisfies another very important property in WSNs like it is having a low overhead. The percentage of extra bits that we need to send per packet due to our protocol is less than in other security protocols like SNEP.

Despite the limited resources in sensor nodes, such as memory and energy, our analysis has demonstrated that the proposed protocol could be implemented in real sensor nodes. It has been designed to avoid unnecesary energy wastage, and it also fulfills the memory constraints that the current sensors have. In our implementation the protocol uses approximately 1Kb of memory to store the necessary databases, and it could be decreased if it was considered necessary. Current sensor nodes like Mica2 have a storage capacity of 4Kb.

By comparing our protocol with SNEP we conclude that our solution consumes more energy in the nodes mainly due to the retransmission of data packets, but in exchange it provides a reliability system that makes that the lost packets are recovered. In many

application scenarios the packet loss can be critical, so in them it will be worth to spend some extra energy on retransmissions, than to lose important data.

6 Future work

Several issues may be worth further investigation. Firstly, after we have showed that this protocol could be implemented in real life sensors, it would be necessary to check and test the performance of the protocol in real life, and to compare the results obtained with those from the simulation.

Second, we have assumed that the secrey keys used to encrypt the messages are known a priori by all the nodes. In practice, however, the solution would be more flexible if these secret keys were securely distributed to all the nodes in a configuration stage. The distribution of the keys should be investigated in depth because some of the configuration messages might get lost and the system should guarantee that the keys are not distributed to anyone from outside the network.

Our protocol could be further developed to include more reliability mechanisms like those used in PSFQ (see section 2.2.1.2) to ensure the reliable transmission of small messages and to avoid sending repeated or unnecessary messages.

In our project we have considered the log-normal shadowing model to simulate the channel behaviour. Other models could be used to complement this model, like for instance including some of the temporal properties of wireless links.

7 Appendix

7.1 Appendix A: Simulator guide

This is a quick guide to the use of the simulator implemented for this project.

This tool simulates the behaviour of a wireless sensor network using the proposed protocol. It was implemented in C programming language and it was not considered necessary to implement a graphical interface due to its easy use.

In the CD attached with this project, we can find 4 files under the folder called "Simulator":

- "srp.c". It is the source code of the simulator. It is written in C language.
- "srp.h". It is the configuration file where they are established the different variables used by "srp.c". This file should be modified if we want to change the parameters of the simulated WSN. See section 7.1.3.
- "srp.m". It is a file written for working in Matlab. It is used for processing the trace files generated by the simulator. See section 7.1.4.
- "srp.exe". It is the file that should be executed to run the simulator. See section 7.1.2

7.1.1 Requirements

The simulator was implemented to work correctly in Linux operating system. Alternatively it is possible to use cygwin, a Linux-like environment that works under Windows.

It will be neccesary to install in the operating system a package with the compiler for C programming language.

7.1.2 Execution

In order to run the simulator the user has to execute the command "srp" from the folder "Simulator". When the simulation finishes two trace files are created for each node of the WSN, and one file with information about the channel is created too.

The files noderX.txt store information related to the packets sent.

The files nodetX.txt store information related to the packets reception.

The file canal.txt stores information related to lost packets in the wireless channel.

7.1.3 Settings

The user can change some parameters of the WSN from the file srp.h. Some of the parameters that can be modified are:

- Number of nodes / Node density.
- Size of the sent packets buffer

- Initial energy in the nodes
- Size of the neighbor database
- Data sending rate
- and many other protocol parameters such as the maximum number of retries for the same NACK message, or the *Th* threshold that decides whether a node ask for retransmissions of old missing packets or not.

After changing any parameter, the code should be recompiled again. Assuming that gcc is correctly installed, it should be executed as follows in order to compile:

"gcc srp.c -o srp"

If everything works properly, we should not see any warnings or errors after compilation. In other case, the error messages should be read carefully to try to solve them.

If the compilation process finished successfully the executable file would be created, called "srp.exe".

7.1.4 Processing trace files

We have used Matlab to process the trace files obtained during the simulations. We have created a script called "srp.m", that reads and analyzes the trace files. This script reads through all the traces that were written during the simulation and gets the important statistics out of them.

To execute the script "srp.m" the simulator should have been executed at least once because the script will try to read trace files. If these files did not exist, an error would be shown in the console.

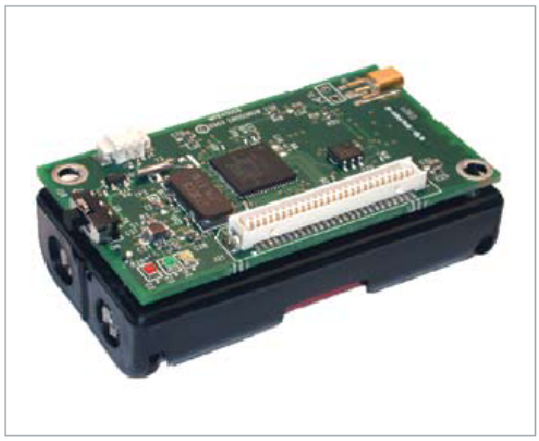
From the Matlab console the user has to execute the command "srp". The script will ask for the number of nodes in the simulation, which the user has to introduce in order to continue with the execution. Finally, it will show on the screen the simulation results such as number of packets sent, received, dropped in the nodes, lost in the channel, etc, as well as the information related with the energy consumption in the nodes and other important parameters.

7.2 Appendix B: MICA2 Datasheet

Crossbow

MICA2

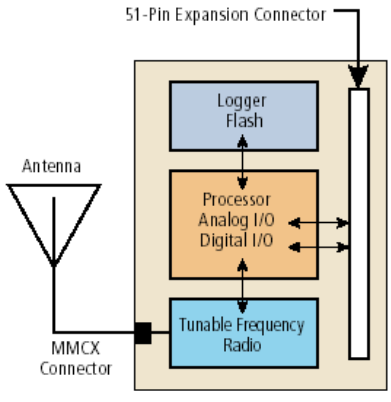
WIRELESS MEASUREMENT SYSTEM



- 3rd Generation, Tiny, Wireless Platform for Smart Sensors
- Designed Specifically for Deeply Embedded Sensor Networks
- > 1 Year Battery Life on AA Batteries (Using Sleep Modes)
- Wireless Communications with Every Node as Router Capability
- 868/916 MHz Multi-Channel Radio Transceiver
- Expansion Connector for Light, Temperature, RH, Barometric Pressure, Acceleration/Seismic, Acoustic, Magnetic and other Crossbow Sensor Boards

Applications

- Wireless Sensor Networks
- Security, Surveillance and Force Protection
- Environmental Monitoring
- Large Scale Wireless Networks (1000+ points)
- Distributed Computing Platform



MPR400 Block Diagram

MICA2

The MICA2 Mote is a third generation mote module used for enabling low-power, wireless, sensor networks. The MICA2 Mote features several new improvements over the original MICA Mote. The following features make the MICA2 better suited to commercial deployment:

- 868/916 MHz multi-channel transceiver with extended range
- Supported by MoteWorks™ wireless sensor network platform for reliable, ad-hoc mesh networking
- Support for wireless remote reprogramming
- Wide range of sensor boards and data acquisition add-on boards

MoteWorks enables the development of custom sensor applications and is specifically optimized for low-power, battery-operated networks. MoteWorks is based on the open-source TinyOS operating system and provides reliable, ad-hoc mesh networking, over-the-air-programming capabilities, cross development tools, server middleware for enterprise network integration and client user interface for analysis and configuration.

Processor and Radio Platform (MPR400)

The MPR400 is based on the Atmel ATmega128L. The ATmega128L is a low-power microcontroller which runs MoteWorks from its internal flash memory. A single processor board (MPR400) can be configured to run your sensor application/processing and the network/radio communications stack simultaneously. The MICA2 51-pin expansion connector supports Analog Inputs, Digital I/O, I2C, SPI and UART interfaces. These interfaces make it easy to connect to a wide variety of external peripherals.

Sensor Boards

Crossbow offers a variety of sensor and data acquisition boards for the MICA2 Mote. All of these boards connect to the MICA2 via the standard 51-pin expansion connector. Custom sensor and data acquisition boards are also available. Please contact Crossbow for additional information.

Document Part Number: 6020-0042-08 Rev A

Processor/Radio Board	MPR400CB	Remarks
Processor Performance		
Program Flash Memory	128K bytes	
Measurement (Serial) Flash	512K bytes	>100,000 Measurements
Configuration EEPROM	4K bytes	
Serial Communications	UART	0-3V transmission levels
Analog to Digital Converter	10 bit ADC	8 channel, 0-3V input
Other Interfaces	DIO,I2C,SPI	
Current Draw	8 mA	Active mode
	< 15 µA	Sleep mode
Multi-Channel Radio		
Center Frequency	868/916 MHz	ISM bands
Number of Channels	4/ 50	Programmable, country specific
Data Rate	38.4 Kbaud	Manchester encoded
RF Power	-20 to +5 dBm	Programmable, typical
Receive Sensitivty	-98 dBm	Typical, analog RSSI at AD Ch. 0
Outdoor Range	500 ft	1/4 Wave dipole, line of sight
Current Draw	27 mA	Transmit with maximum power
	10 mA	Receive
	< 1 µA	Sleep
Electromechanical		
Battery	2X AA batteries	Attached pack
External Power	2.7 - 3.3 V	Connector provided
User Interface	3 LEDs	User programmable
Size (in)	2.25 x 1.25 x 0.25	Excluding battery pack
(mm)	58 x 32 x 7	Excluding battery pack
Weight (oz)	0.7	Excluding batteries
(grams)	18	Excluding batteries
Expansion Connector	51-pin	All major I/O signals

Notes: Specifications subject to change without notice

Base Stations

A base station allows the aggregation of sensor network data onto a PC or other computer platform. Any MICA2 Mote can function as a base station when it is connected to a standard PC interface or gateway board. The MIB510/MIB520 provides a serial/USB interface for both programming and data communications. Crossbow also offers a stand-alone gateway solution, the MIB600 for TCP/IP-based Ethernet networks.

MIB520 Mote Interface Board

Ordering Information

Model	Description
WSN-START900CA	MICA2 Starter Kit 868/916 MHz
WSN-PRO900CA	MICA2 Professional Kit 868/916 MHz
MPR400CB	868/916 MHz Processor/Radio Board

Document Part Number: 6020-0042-08 Rev A

8 References

[1] : "Security Models for Wireless Sensor Networks", Sophia Kaplantzis, March 20, 2006.

[2] : "Data Transport Reliability in Wireless Sensor Networks -- A Survey of Issues and Solutions", A. Willig and H. Karl, .

[3] : "SPINS: Security Protocols for Sensor Networks", Adrian Perrig, Robert Szewczyk, J.d. Tygar, Victor Wen and David E. Culler, 2002.

[4] : "Security in Wireless Sensor Networks", Mayank Saraogi, .

[5] : "RMST: Reliable Data Transport in Sensor Networks", F. Stann and J. Heidemann, May 2003.

[6] : "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks", Chris Karlof, Naveen Sastry, David Wagner, November 3-5, 2004.

[7] : "ART: Asymmetric Reliable Transport Mechanism for Wireless Sensor Networks", Nurcan Tezcan and Wenye Wang, December 2004.

[8] : "Pump-Slowly, Fetch-Quickly (PSFQ): A Reliable Transport Protocol for Sensor Networks", Chieh-Yih Wan, Andrew T. Campbell and Lakshman Krishnamurthy, April 2005.

[9] : "PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks", C.-Y. Wan and A. T. Campbell and L. Krishnamurthy, 2002.

[10] : "ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks", Y. Sankarasubramaniam and O.B. Akan and I.F. Akyildiz, June 2003.

[11] : "A Scalable Approach for Reliable Downstream Data Delivery in Wireless Sensor Networks.", S. Park, R. Vedantham, R. Sivakumar, and I. Akyildiz, May 2004.

[12] : "RMTP: A Reliable Multicast Transport Protocol.", John C. Lin, and Sanjoy Paul, March 1996.

[13] : "PGM Reliable Transport Protocol", Tony Speakman, Dino Farinacci, Steven Lin, Alex Tweedly, Nidhi Bhaskar, Richard Edmonstone, Rajitha Sumanasekera, and Lorenzo Vicisano, Feb 2001.

[14] : "Improving TCP Performance over Mobile Ad-Hoc Networks with Out-of-Order Detection and Response", F. Wang and Y. Zhang, June 2002.

[15] : "Networking Wireless Sensors", Bhaskar Krishnamachari, 2005.

[16] : "Topology Control in Wireless Ad Hoc and Sensor Networks", Paolo Santi, 2005.

[17] : "The ns Manual", Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC, 2007.

[18] : "Analizyng the Transitional Region in Low Power Wireless Links", Marco Zuniga and Bhaskar Krishnamachari, .

[19] : "Link Layer Models for Wireless Sensor Networks", Marco Zuniga and Bhaskar Krishnamachari, .

[20] : http://www.xbow.com/Products/productdetails.aspx?sid=156

[21] : "OMAC: One-Key CBC MAC. Fast Software Encryption", T. Iwata and K. Kurosawa, February 24, 2003.

[22] : "Survey and Benchmark of Block Ciphers for Wireless Sensor Networks", Yee Wei Law, Jeroen Doumen, and Pieter Harte, February 2006.

[23] : "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication", Morris Dworkin, May 2005.

[24] : "“Energy Analysis of Public-Key Cryptography for Wireless Sensor Networks", S. Wander, Nils Gura, Hans Eberle, Vipul Gupta, Sheueling Chang Shantz, .

[25] : "MiniSec: A Secure Sensor Network Communication", Mark Luk, Ghita Mezzour, Adrian Perrig, Virgil Gligo, 2007.

[26] : "A Survey of Simulation in Sensor Network", David Curren, .

[27] : http://www.btnode.ethz.ch/Projects/Mica2

[28] : "Sensor Networks for Medical Care", Victor Shnayder, BorrongChen, Konrad Lorincz, Thaddeus R. F. Fulford-Jones, Matt Welsh, 2005.

[29] : "Self-Powered Wireless Sensor Networks for Remote Patient Monitoring in Hospitals", Abhiman Hande, Todd Polk, William Walker and Dinesh Bhatia, 22 September 2006.

[30] : "Medical Informatics System with Wireless Sensor Network-enabled for Hospitals", Loh P. K. K., Lee Allan, 2005.