MSc in Computer Science

<u>Term Project:</u> IPv6 in Wireless Sensor Network /6LowPAN routing

Supervisor: Dr. Nikos Samaras
Student: Maria Palaska

September 2011

Contents

PART A

Abstract

In the digital world there is a requirement, each thing is connected to the Internet. Wireless Sensor Networks (WSN) is the most appropriate way to achieve this assumption. This type of networks is very important and useful in order to achieve a digital appearance of the real world. The sensors nodes hold a radio module with which they have the ability to transfer messages to a base station or other nodes. In recent years, the majority of sensors networks use proprietary protocols. This process raises limitations in the construction of a sensor network consisting of a great number of different sensors nodes. Moreover, from the side of the Internet of Things these sensor networks need auto configuration and a huge address space. Furthermore, the sensor nodes have many differences among the usual hosts of web, according to their highly limitations in power consumption and great constraints in computation power. Recent surveys and researches have proved that the combination of IPv6 and wireless sensor networks is feasible despite all the limitations that exist. The aim of this thesis is to present the current work that has been done through the combination of IPv6 and wireless sensor networks. The thesis also focuses on the implementation of IPv6 in WPAN (Wireless Personal Area Networks) which imports an adaptation layer that enables efficient IPv6 communication over IEEE 802.15.4 enabled sensor networks. The main point of this work is the 6LowPAN routing. The useful tool used in this thesis in order to extract our results over 6LowPAN routing protocols was the OMNET++. This dissertation extends the implementation of already existing protocols like AODV, LOAD and DYMO. Finally, it focuses on the determination of the constraints of afore mentioned combination between IPv6 and WSNs. The arising results are quite interesting and become a starting point for further research through this issue.

Keywords

IPv6, Wireless Sensor Networks, Wireless Personal Area Networks, low-power, 6LowPAN, 6LowPAN routing protocols

1. Introduction

In general, Wireless Sensor Networks are necessary and very important in many applications to the Internet. Since the wireless sensor networks connect to the Internet via IPv6, they inherit the advantages of the IPv6 like the great address space (132 bit). The preparation of sensor networks in order to achieve an IP communication and the use of them in the Internet has as a consequence the necessity of specific features. A characteristic example of this assumption is the adaptation of the respective link technology, specification of ad hoc networking, handling the security issues, and auto configuration to support ad hoc deployment. Moreover, the mobility is also necessary while the sensor networks acts according to the IP side. As we said before, the IPv6 needs a great address space in order to address the huge sensor networks in global condition by presenting support for mobility and auto configuration by IPv6 neighbouring discovery [1, 2].

From wireless sensor networks point of view it is urgent the design, the creation and the implementation of an IPv6 enabled sensor network. The implementation of this new type of network and its incorporation in an IPv6 WAN configuration raises a variety of assumptions on the architecture and its functional processes.

2. Background

2.1 Wireless Sensor Networks

The new evolutionary step in home, industrial, building, utilities and transportation systems automation comes from smart environments. The smart environments mainly based on sensory data which comes from the real world. Especially, sensory data accrues from multiple sensors of different modalities in distributed locations. Thus, the smart environment needs information about its surroundings as well as about its internal workings.

The sensors which integrated into structures, machinery and environment and combined with the integral delivery of sensed information could offer great benefits to the society. The main benefits which this kind of sensors could afford are the followings, fewer catastrophic failures, maintenance and protection of the natural resources, amelioration of emergency response and manufacturing productivity and reinforce the homeland security. The main field of wireless sensor networks conflates sensing, computation, and communication into a device which may be tiny (Figure 1). Thus, in wireless sensor network via mesh networking protocols, these certain devices may shape a sea of connectivity that extends the reach of cyberspace out into the physical world. The water flows to fill every room of a submerged ship and the mesh networking connectivity will seek out and exploit any possible communication path by transferring data from node to node in order to find its destination. While the capabilities of any single device are minimal, the composition of hundreds of devices offers radical new technological possibilities. Nevertheless, there are many constraints in the use of sensors in structures and machines. Remarkable are the lead wires and the fiber optic tails which may be reason for breakage time consuming installation and connector failures. However, the wireless sensor networks can provide the reduction of these costs, easing installation and eliminating connectors [3, 4, 5, 6].

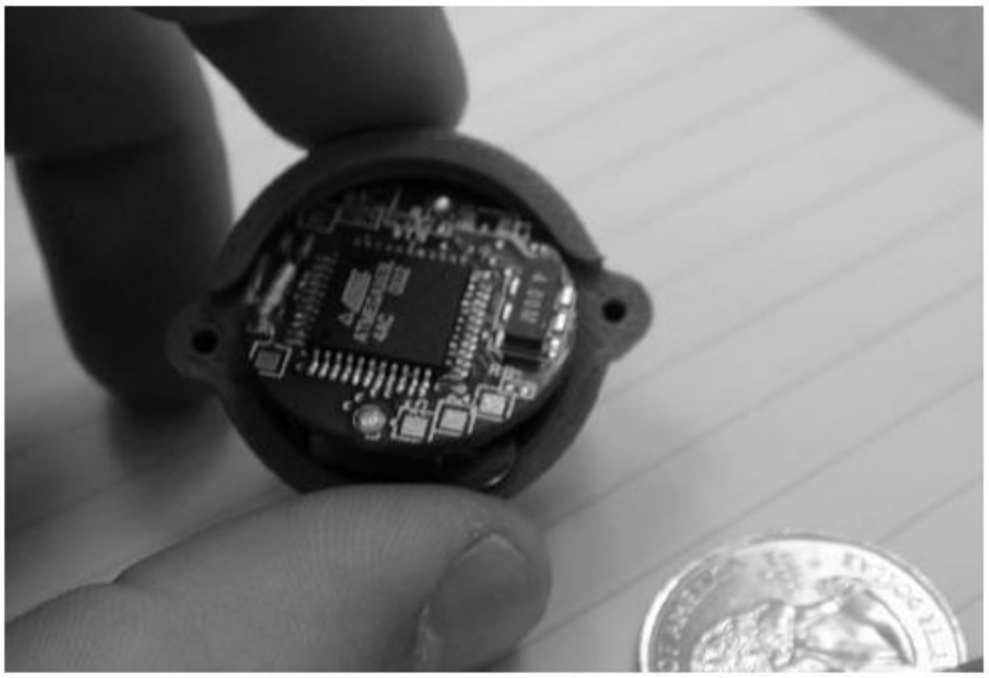


Figure 1: *Wireless sensor network device in a tiny size*

The unique wireless sensor should be networked and scalable, expends very little power, should be fast in data capture, easy to installation, costs a little, requires no real maintenance, should be reliable, software programmable and in generally it should be smart.

The ability to select the ideal sensors and wireless communications link demands knowledge of the application and problem definition. Some characteristic measurements which should be taken into consideration are size, battery life, and sensor update rates. Moreover, a lot of examples about low data rate sensors include humidity, temperature and peak strain capture passively and examples of high data rate comprises strain, vibration and acceleration.

The ability of wireless sensor networks to deploy large numbers of tiny nodes that assemble and configure themselves affects the power of the networks. The scenarios that have been used for this kind of devices range from real – time tracking, to monitoring the conditions of environment, ubiquitous computing environments, to monitoring the health of structures and equipments. While often referred to as wireless sensor networks, they can also control actuators that extend control from cyberspace into the physical world [7].

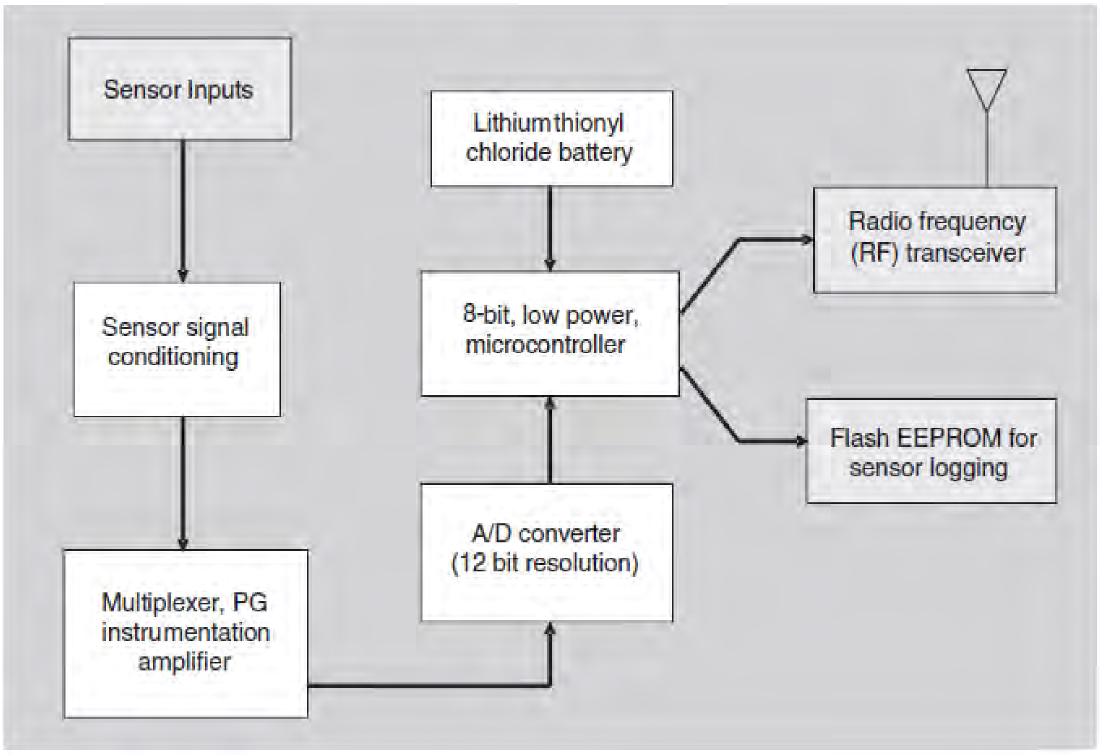
The most distinct application for the wireless sensor network technology is the monitoring of remote environments for low frequency data trends. For example, a chemical plant which could be checked for leaks by many sensors simultaneously make a wireless interconnection network and at the same time announce the detection of any chemical leak. In the same condition with a tradition wired system the deployment cost would be minimal.

The actual situation in the wireless systems only scratches the surface of possibilities emerging from the integration of sensing, energy storage, low-power communication and computation. Generally, the current thought over the wireless devices is items like laptops with 802.11, cell phones, personal digital assistants. These types of devices are expensive; they cost a lot, refer to certain applications and rely on the pre-deployment of extensive infrastructure support. In contrary, wireless sensor networks utilize small devices which are low- embedded for a variety of applications and do not based on a specific pre – existing infrastructure [4, 8, 9]

The base of wireless sensor networks relies on a simple equation: *Sensing + CPU + Radio = Thousands of potential applications*. The combination of sensors, CPUs and radios into an appropriate wireless sensor network needs a detailed understanding of the both capabilities and limitations of each of the underlying hardware components, and also a detailed understanding of modern networking technologies and distributed systems theory. Every separate node should be designed in order to provide the necessary and initial components in order to compose the interconnected web that will present as they are deployed, while finding definite limitations of size, cost and power consumption. A main challenge is to correlate the overall system requirements to device capabilities, requirements and actions. For being the wireless sensor network vision a reality, architecture must be developed that synthesizes the envisioned applications out of the underlying hardware capabilities [4].

2.1.1 Individual Wireless Sensor Node Architecture.

A modular design approach (over a wireless sensor node) presents a flexible and versatile platform in order to address the requirements of a great selection of applications (Figure 2). Specifically, the main function of every wireless sensing node is to eliminate the power consumed by the system. Generally, the subsystem of radio demands the largest amount of power. Consequently, it is positive to transfer data over the radio network only when needed. This sensor event-driven data collection model requires an algorithm to be loaded into the node to define when to transfer data based on the event. Moreover, it is meaningful to eliminate the power consumed by the sensor itself. Thus, the hardware which is used should allow the microprocessor to control power to the radio, sensor, and sensor signal conditioner [10].



Sensor Inputs
Lithium thionyl chloride battery
Radio frequency (RF) transceiver
Sensor signal conditioning
8-bit, low power, microcontroller
Flash EEPROM for sensor logging
A/D converter (12 bit resolution)
Multiplexer, PG instrumentation amplifier

Figure 2: *Wireless sensor node functional block diagram*

2.1.2 Physical Layer and Wireless Sensor Network

The physical layer determines the modulation scheme, the hardware interface of the radio to the system and the operating frequency. There are a lot of low power radio integrated circuits that are right selections for the radio layer in wireless sensor networks, including those from companies such as MicroChip, Atmel, ChipCon, Melexis, and Micrel. Radio standards that exist today and may or may not implement to wireless sensor network are presented below [10, 11, 12]

IEEE802.11x: is a standard that is appropriate for local area networking and for data transmission between computers or other devices with high bandwidth. In this standard the data rate fluctuates from 1 Mbps to over 50 Mbps. Moreover, the transmission range is 300 feet with a standard antenna. If a directional high gain antenna is used then the range can be improved. Also, both direct sequence spread spectrum modulation and frequency hoping schemes are available. Finally, in this standard, when the data rates are too high for wireless sensor applications then the power requirements block its use in this certain kind of applications.

IEE802.15.1 & .2 (Bluetooth): is a standard which is low power than 802.11 and it was initiated by the by the IEEE as Wireless Personal Area Network (WPAN) in 1998. It is appropriate for applications which are responsible for data transfer from personal computers to peripheral devices like mobile phones, PDAs, or personal digital assistants. This standard relies on star network topology and supports one base station node and up to seven remote nodes for communication. Many companies have created wireless sensors which rely on Bluetooth, but they have not been accepted by the public. This happens because of Bluetooth protocol limitation, some of them are presented in the following, The limitations are the low number of nodes per network (up to seven), the long time for synchronization nodes to network, from sleep to active mode, the Medium Access Control (MAC) layer is totally composite when compared to that required for wireless sensor applications and the quite high power for a short transmission range.

IEEE802.15.4: is a standard that it was created in order to cover the requirements of wireless sensing applications. The IEEE802.15.4 standard is very flexible; it prescribes multiple transmission frequencies and multiple data rates. Although the power is quite low, the hardware which is used eliminates the power to minimal level. Generally, the characteristics which support this certain standard are presented below:

- Uses star and mesh network topologies.
- Data rates could be 20 Kbps, 40 Kbps and 250 Kbps.

- Link quality indication, which refers to multi-hop mesh networking algorithms.
- Transmission frequencies fluctuate at 868 MHz/902–928 MHz/2.48–2.5 GHz.
- Supports direct sequence spread spectrum (DSSS) for robust data communications
- Uses AES-128 security for encryption of transmitted data.

ZigBee: standard consists an association of companies which working together in order to enable reliable, low power, cost effective, wirelessly networked monitoring and control products based on an open global standard. This type of standard prescribes the IEEE 802.15.4 as the physical and MAC layer and is appropriate for higher level applications like HVAC monitoring and lighting control (Figure 6). Moreover, ZigBee supports star and hybrid star – mesh network topology. Finally, this standard includes the IEEE802.15.4 specification and elaborates on the network and the application interface.

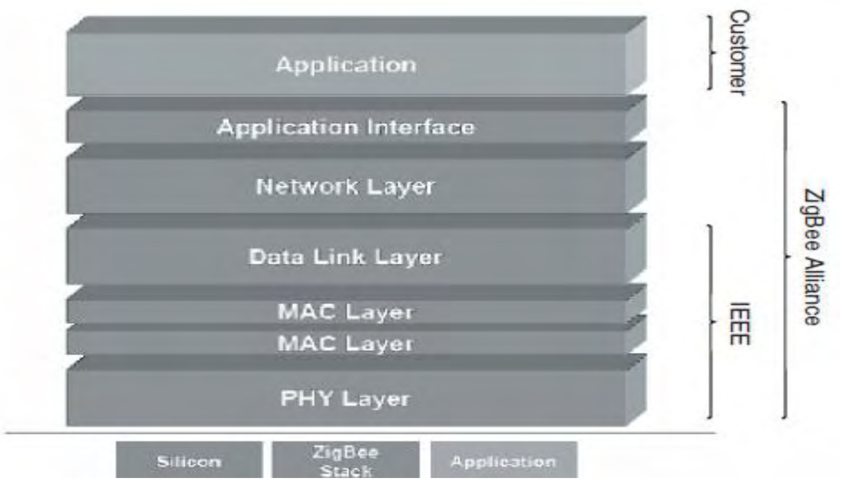


Figure 6: *ZigBee*

IEEE1451.5: the working group of this standard wants to cover the vulnerable points of previous standards in order to standardize the interface of sensors to a wireless network. Today, this standard has been chosen as the wireless networking communications interface and the working group is trying to reach a sensor interface.

2.1.3 Metrics

The metrics are very important as they will be used in order to evaluate a wireless sensor network. The main evaluation metrics for wireless sensor networks are cost and ease of deployment, response time, lifetime, coverage, security, effective sample rate and temporal accuracy. A lot of time some metrics are concerned. For example, sometimes it may be necessary to eliminate one metric like the sample rate in order to increase another like the lifetime. The metrics are described more specifically below [5].

Cost and ease of deployment. A great advantage of wireless sensor network is its ease of deployment. It should be necessary and possible for a person without experience to put the nodes throughout the environment. Furthermore, the system would shape itself automatically when a node is replaced. Nevertheless, the real systems should put limitations on nodes placement. For example is not possible for a node not to have a certain range. Moreover, the wireless sensor network should have the capability to estimate the quality of the network deployment and remark any problem which will exist. Besides that the system should be able to alter the environment conditions. Specifically, during the lifetime of a deployment, nodes may change their position or large physical objects may be placed something which may affects the communication between two nodes.

The initial deployment is the first part of network lifecycle. The total cost of ownership for a system mainly affects the maintenance cost than the initial deployment cost. Generally, in real application, a piece of the total energy sum should be dedicated to system maintenance and verification. The procedure of finding and reconfiguration of traffic eliminates the network lifetime and the effective sample rate.

Response time. The response time is a very important metric for the performance of an application and when environmental monitoring is used in order to check factory machines and equipments. These systems are practical only when the response time guarantees could be met.

The necessity to have low response time clashes the techniques which are used in order to increase network lifetime. The lifetime of the network can be increased only with nodes which operate their radios for certain and small periods of time. When a node turns on its radio every minute in order to forward and receives data messages, it is very difficult to meet the application requirements for response time of a security system.

Response time can be upgrade when the system contains nodes that are powered all the time. This type of nodes listens and understands the alarm messages and can transmit them when it is necessary. Nevertheless, this function may eliminate the ease of deployment of the system.

Lifetime. Lifetime is a very critical metric for any wireless sensor network deployment. The aim of environmental monitoring and security application scenarios is to have nodes which are not checked for months or years. A great limitation of lifetime is the energy supply. Every node should control its local energy supply in order to maximize the total lifetime of the network. In the most cases the average lifetime of a node is not very critical but the minimum lifetime of the node is. For the wireless sensor networks the nodes should be endure for many years, because a failure of s node would make the system vulnerable.

Coverage. With the lifetime, coverage is the most important metric of wireless sensor networks. It is very positive for a network to deploy over a larger physical area. This ability can augment a system’s value to the end user. It is remarkable the fact that the coverage of the network is not equal to the range of the wireless communication links which are used. A variety of multi - hop communications have the ability to extend for a no certain time period the network well beyond the range of the radio technology alone. Nevertheless, for a certain transmission range the multi – hop protocols can increase the power consumptions of the nodes which eliminate the lifetime and with a certain node density can increase the deployment cost.

Security. The procedure of maintaining the important information of a network in a safe and secure mode is very necessary. Some certain information can be easily extracted from a trace and when this information go to wrong people may cause many problems like an attack.

In a network not only should the system provide privacy but also may be able to provide authentication on data communication. It should not be possible to introduce a false alarm message or to replay an old alarm message as a current one. The usage of encryption and authentication code costs power and also network bandwidth.

Effective sample rate. In data collection network the effective sample rate is very important metric for the performance of the application. Specifically, the effective sample rate is the sample rate that sensor data can be taken at every individual sensor and communicated to a collection point. Usually, environmental applications demand one or two sample rates in a minute.

Temporal accuracy. In tracking and environmental applications the sample of the nodes should correlated with the time in order to define the origin of the phenomenon which is measured.

In order to achieve a temporal accuracy the network should be able to construct and keep a global time base which can use the sample rates and the events in a chronological order [5].

2.1.4 Software for Wireless Sensor Networks

A very important and critical point for the completion of a wireless sensor network is the design of a software architecture that can eliminate the gap between the raw hardware capabilities and the complete system. The characteristics for an appropriate hardware are a lot. Specifically, a right hardware should be efficient in memory, processor and power in order to be compatible in the strict requirements of a variety of applications. Moreover, it should be too tolerant in order to support simultaneously many applications and many resources like memory, communication and computation. A very appropriate example of hardware for wireless sensor networks is the TinyOS [4, 5, 10].

2.2 IPv6

In general, IPv6 is the heir of the Internet Protocol IPv4. An IPv6 network contains a number of nodes and routers which communicate with each other with a certain way (Figure 7, 8). The choice of a certain topology is justified for reasons of reliability. Moreover, in case of fault, the topology has alternative paths.

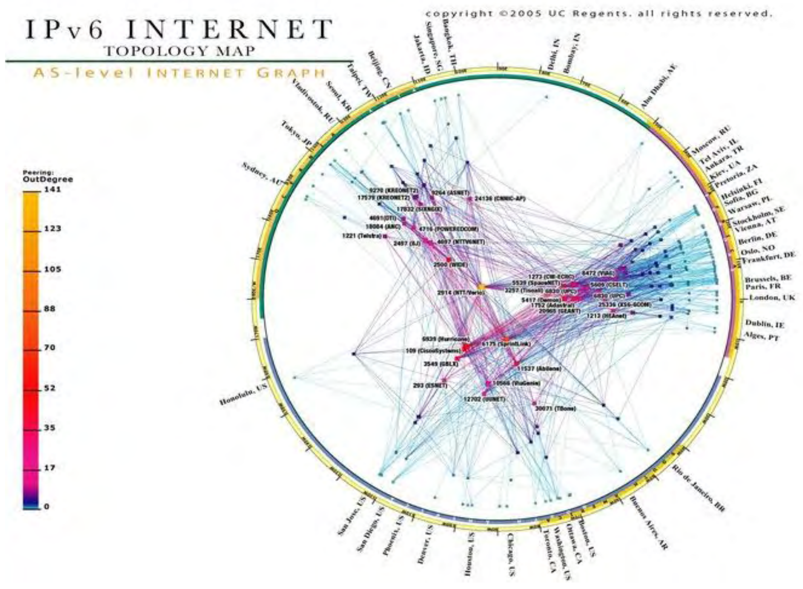


Figure 7: *Ipv6 Internet topology*

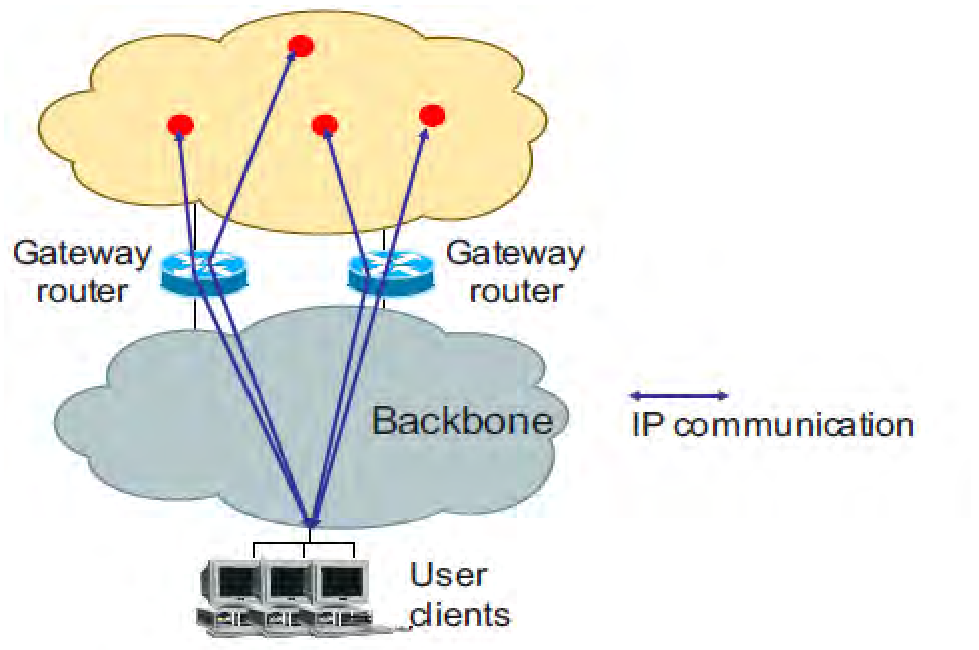


Figure 8: *IP enabled sensor network in the Internet*

A very important point of an IPv6 topology is the unique identification of every node in order to reach all the nodes in the network. The IPv6 provides a 128-bit numerical address to each network interface. However, in many cases, users in order to find a node use a more convenient name than a numerical address. The name and the address have the same capability, to define the unique identification of an interface in a network. The difference between the name and the address is that the address is responsible to interact with routing mechanisms and is numerical, while the name is more accessible to the users and it is alphanumerical and easier to remember it. Moreover, in the last years the growth of network sizes creates the necessity of adopting a distributed database which called *Domain Name Service* – DNS [13].

As we said the address should be unique. This requirement was already existed in IPv4, but the IPv6 extends the addresses due to the growth of Internet and Intranets. This happens via the organizations that assign sets of addresses to end users. In IPv4 the sets are known as networks and can be divided in smaller nets, the subnetworks through a netmask. Thus, two nodes are connected only when their addresses are belonged to the same subnetwork.

In IPv6 the situation through the network and subnetworks are the same with two important differences. First of all the addresses are longer. IPv6 contains 128 bits while the IPv4 contains 32 bits. Thus, IPv6 supports more levels of addressing hierarchy, more addressable nodes and simpler autoconfiguration of addresses. Also, in IPv4 there is the concept of netmask while in IPv6 there is the concept of prefix. The prefix shows how many bits are used to identify the subnetwork.

In general, the IPv6 is the next generation of IPv4 with some differences [14]:

- Header format simplification. Header fields have the ability to keep the bandwidth cost as low as it can, in spite of the big size of the addresses.
- Quality – of – Service (QoS) capabilities. IPv6 capacitates the labelling of packets which are belonged to a certain traffic for which the sender requests special handling, such as non-default quality of service or "real-time" service.
- Authentication and privacy capabilities. IPv6 contains the definition of extensions that gives support for data integrity, confidentiality and authentication.
- Improved support for options. The changes that exist in IPv6 headers support more efficient forwarding, better flexibility in the entrance of new options and less strict limits.

2.2.1 Routers

When a user wants to use an application on a certain computer, should require it by reporting the specific name of the computer. Then, the network consults the Domain Name Service and extracts the IPv6 address of the remote computer. The address of the computer which comprises the destination is the main point which specifies the most suitable routing in order to reach the remote node. Moreover, when the destination and the sender are connected on the same physical network, the transmission can be directly. In contrary, the function of internetworking is necessary. In this case, the sender transmits the data and the router waits its delivery [13].

The main operation of the router is to transmit messages over the network. The appropriate routing technique can very depending on network architecture. Protocols like IPv4, IPv6, IPX, OSICLNP, DECnet, and so on use the routing by network address technique. A node contains its address in the layer 3 packet which should be unique over the network. Every router utilizes this certain address as an index in its routing table specifies the path on which the packet should be forwarded.

The instant when the packet reaches a router via a geographical or local network interface, the router transmits the packet to its forwarding process, which exports the source address, utilizes this certain address to check the routing tables and resolves on which interface should forward the data (Figure 9) [14].

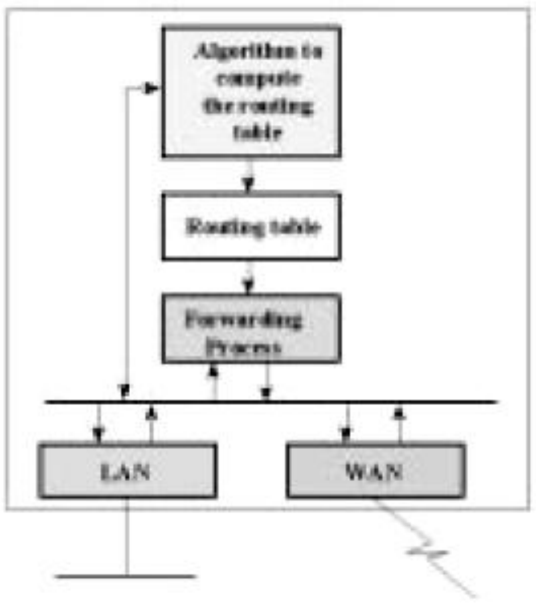
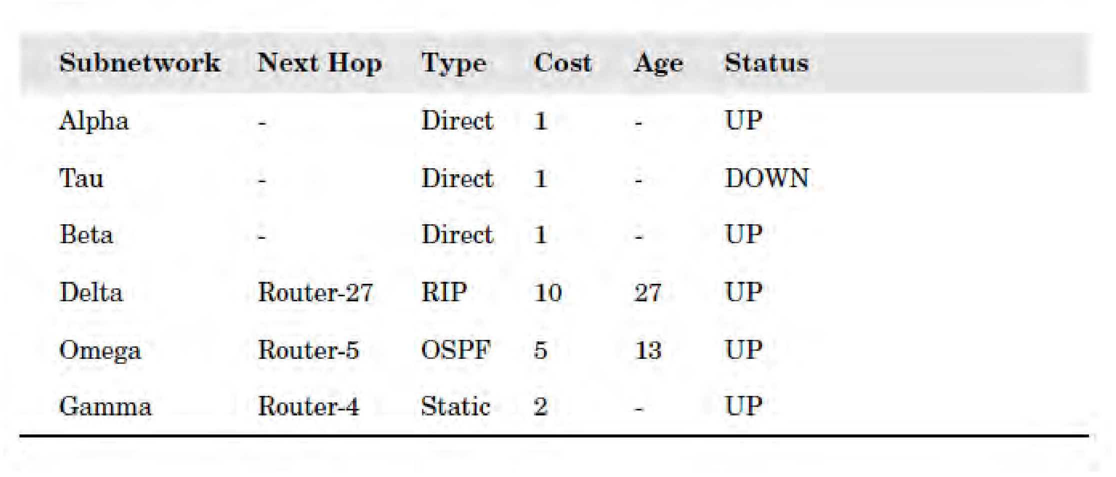


Figure 9: *Router internal architecture*

2.2.2 Routing Table

In a routing table of an IPv6 router includes an entry for every subnetwork which reaches the router itself. The routing tables can be written manually or automatically by certain protocols like OSPF and RIP [13].



Subnetwork	Next Hop	Type	Cost	Age	Status
Alpha	-	Direct	1	-	UP
Tau	-	Direct	1	-	DOWN
Beta	-	Direct	1	-	UP
Delta	Router-27	RIP	10	27	UP
Omega	Router-5	OSPF	5	13	UP
Gamma	Router-4	Static	2	-	UP

Table 1: *A general scheme for a routing table organization*

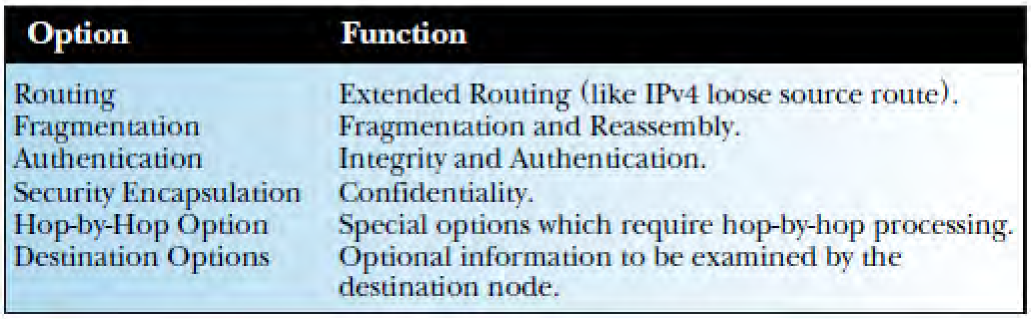
In this certain case, the address of the subnetwork which has the form of *FEDC: BB87:0:0:0:0:0:0/80* and a 80-bits prefix can be colligated with the name of Delta. Furthermore, for the Next Hop field, the Router-4 could have an address with a form of *FEDC: BB87:0:0:0:0800:2B3C:4D73*. Moreover, the Type field provides a type of approach which correlates with the subnetwork. Also, Direct type, provides a router which has an interface which is connected directly to the subnetwork. RIP and OSPF are protocols which show the reachability of the subnetwork. Furthermore, the Age field determines the seconds which are left and is appropriate only for entries which correlated with networks whose reachability is learned via protocols of the routing tables. Finally, the Status field shows the status of the each entry (Table 1–UP, DOWN).

The transmission function of the router consulted the routing table for every data by searching in the column of subnetwork for which subnetwork the destination address belongs and then by routing the packet to the associated Next Hop.

2.2.3 IPv6 Extensions

IPv6 options are contained in separate extension headers that are placed between the IPv6 header and the transport layer header in a data packet (Table 2). Usually, the IPv6 extensions headers are not checked through the process of transmission. They are checked only when they reach their final destination. This capability accommodates the router performance for packets containing options. Also, this capability improves IPv6 extension headers which can be of an uncertain length and the total amount of options carried in a packet is not limited to 40 bytes [15, 16].

IPv6 options are an integer multiple of 8 octets long, to retain this alignment for subsequent headers.



Option	Function
Routing	Extended Routing (like IPv4 loose source route).
Fragmentation	Fragmentation and Reassembly.
Authentication	Integrity and Authentication.
Security Encapsulation	Confidentiality.
Hop-by-Hop Option	Special options which require hop-by-hop processing.
Destination Options	Optional information to be examined by the destination node.

Table 2: *IPv6 extension headers*

2.2.4 IPv6 Addressing

IPv6 addresses contain 128-bits and are identifiers for individual interfaces and sets of interfaces. The form of an IPv6 like x:x:x:x:x:x:x:x where x is one to four hexadecimal digits and is known as group. An example of an IPv6 address is the following, ABCD:EF98:7654:3210:ABCD:EF98:7654:3210. Moreover, there are some rules in IPv6 addresses. The first rule is correlated with the first zeros of each group. Any zero which leads a group can be skipped. Thus, the following address AB05:0:0:0:0:0:0:505 can be written like AB05:505. It is very important to know that this option can be done only once in an address [14].

- All types of IPv6 addresses are correlated with interfaces and not with nodes. While every interface corresponds to a single node any of that node's interfaces' unicast addresses act like an identifier of the node. A single interface has the capability to define more than one IPv6 addresses of any type. There are three types of addresses, **unicast**, **anycast** and **multicast** [17, 18, 19, 20]. The unicast is an identifier for a single interface. In this case, the data packet which sent to a unicast address is transmitted to the interface which identified by that address. One differs between *Global Unicast*, *Site-Local Unicast* and *Link Local Unicast* (Figures 10, 11, 12).

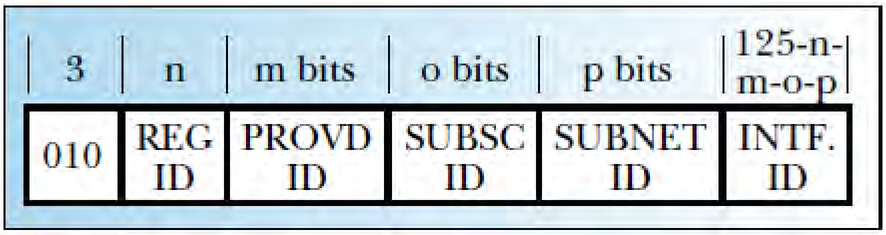


Figure 10: *A scheme for unicast addresses*

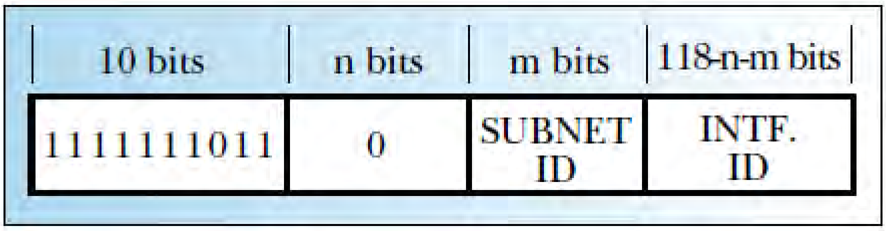


Figure 11: *Site-Local Unicast address format*

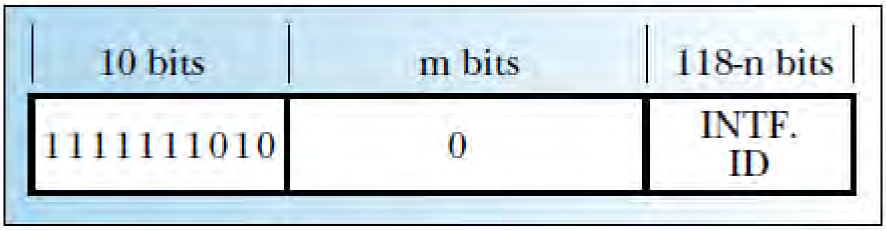
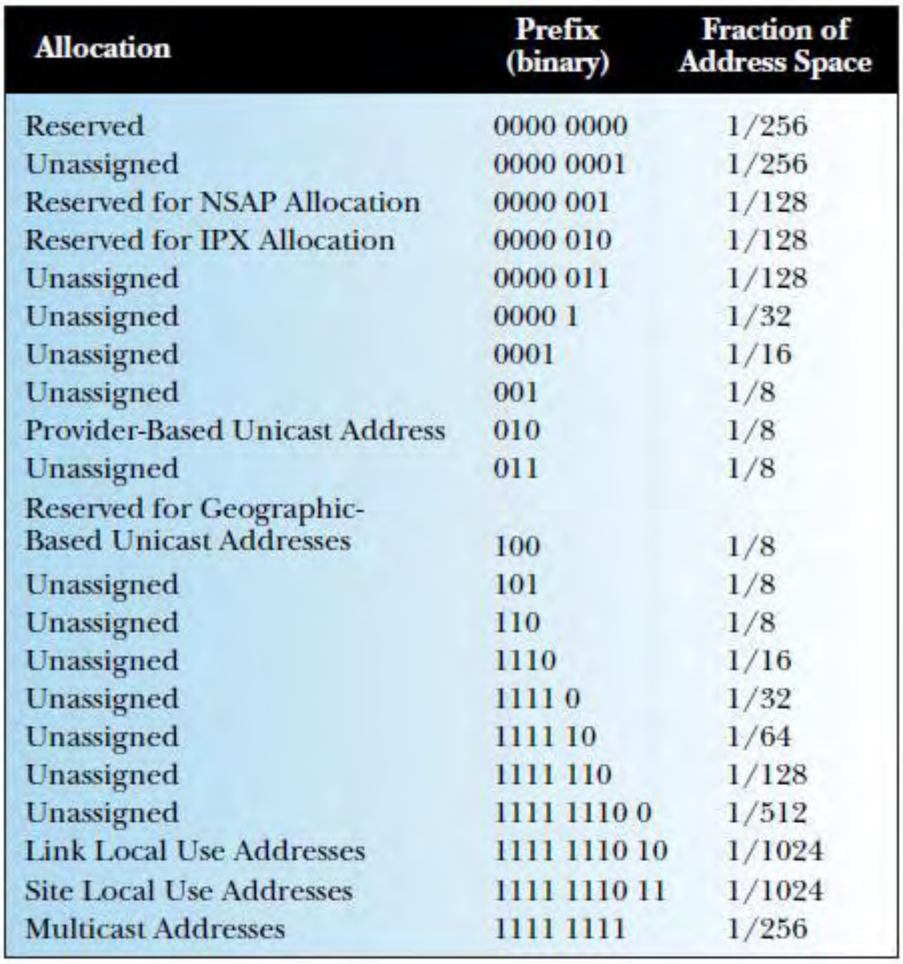


Figure 12: *Link-Local Unicast address format*

Moreover, anycast acts an identifier for a set of interfaces which contain different nodes. The data packet which sent to an anycast address is transmitted to the appropriate and nearest interface which is identified by that addresses. Finally, multicast similarly with the previous case, an identifier acts for a set of interfaces which contain different nodes. A data packet which sent to a multicast address is forwarded to all interfaces identified by that address. In IPv6 there are no broadcast addresses, their operation being superseded by multicast addresses.

The IPv6 supports addresses that contain four times the number of bits which can support the IPv4. It is the extraordinary number of 340,282,366,920,938,463,463,374,607,431,768,211,456 addresses. This is a very large address space. Practically, in the routing of addresses is necessary the development of hierarchies which eliminate the efficiency of the use of the space of addresses.

The certain type of IPv6 address is distinguished by the first bits of the address. The field which contains these leading bits has a variable length and is called Format Prefix – FP (Table 3).



Allocation	Prefix (binary)	Fraction of Address Space
Reserved	0000 0000	1/256
Unassigned	0000 0001	1/256
Reserved for NSAP Allocation	0000 001	1/128
Reserved for IPX Allocation	0000 010	1/128
Unassigned	0000 011	1/128
Unassigned	0000 1	1/32
Unassigned	0001	1/16
Unassigned	001	1/8
Provider-Based Unicast Address	010	1/8
Unassigned	011	1/8
Reserved for Geographic-Based Unicast Addresses	100	1/8
Unassigned	101	1/8
Unassigned	110	1/8
Unassigned	1110	1/16
Unassigned	1111 0	1/32
Unassigned	1111 10	1/64
Unassigned	1111 110	1/128
Unassigned	1111 1110 0	1/512
Link Local Use Addresses	1111 1110 10	1/1024
Site Local Use Addresses	1111 1110 11	1/1024
Multicast Addresses	1111 1111	1/256

Table 3: *Format Prefix*

This allocation provides the direct allocation of provider addresses, local use addresses, and multicast addresses.

IPv6 nodes can have a feeling about the internal structure, depending on the role which a node may have. For example, a node may think that unicast addresses have no internal structure. Thus, a slightly sophisticated host may additionally be aware of subnet prefix(es) for the link(s) it is attached to, where different addresses may have different values.

2.2.5 Header Format

IPv6 has a design goal which is the simplification of the header format. The new edition of header format is correlated with the new addressing format and should have a certain size (Figure 13).

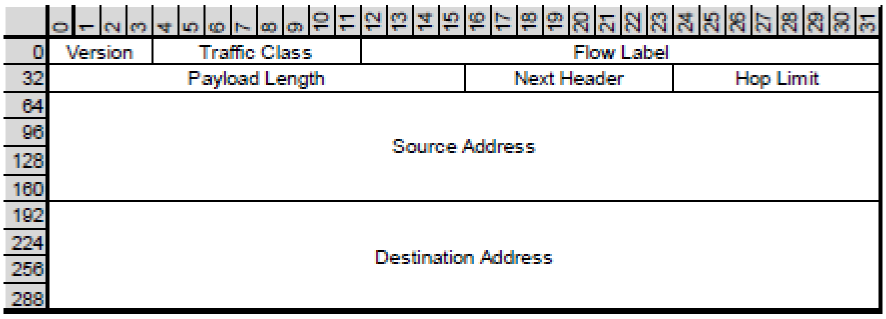


Figure 13: *The new header format*

The first field includes the IP Version number and it is set to the value six. The two following fields, the Traffic Class field and the Flow Label field are responsible to offer quality of services. Specifically, the Traffic class field correlated with the packet priority and the flow Label Field is responsible for the Quality of Service (QoS) management but now is unused. Furthermore, the Payload Length Field assigns the length of the IPv6 payload. The Next Header field identifies the type of header. Moreover, every node that transmits the packet eliminates the hop limit field by one. Finally, the Source and the Destination Address include the according 128-bit IPv6 address [15, 17, 21, 22, 23].

2.2.6 Neighbor Discovery Protocol

The aim of the Neighbor Discovery Protocol (NDP) is to find solutions in problems which related to the link layer. This contains the prefix and router discovery, the address resolution and autoconfiguration, the duplicate address detection and the neighbor unreachability.

Explicitly, the Router Solicitation message is used from a host that becomes enabled. It transmits these messages in order to request routers in order to create router advertisement messages. Thereafter, the Router Advertisement message is transmitted by a router in order to show their presence with various links and Internet parameters and network prefix. The identification of the link-layer is happened by the Neighbor Solicitation message and the response to this is the Neighbor Advertisement message [13, 18].

2.2.7 IPv6 Security

Internet has a lot of security vulnerable points and absence of effective privacy and authentication mechanisms under the application layer. For theses cases, IPv6 has two functions which provide security services. These certain functions can be used separated but also can be used together depending on the security needs of each security. The usage of these functions supports different levels of security and different kind of users.

The first security function of IPv6 is known as IPv6 Authentication Header. This certain header is an extension which is algorithm independent and supplies authentication and integrity to IPv6 datagrams. The usage of IPv6 Authentication Header is necessary when source routing is used with IPv6 because of the known risks in IP source routing. Its function at Internet layer can assist provide host origin authentication to the upper layer protocols and services which have lack of protection.

The second security function of IPv6 is known as Encapsulating Security Header. This certain header is also an extension header which supplies integrity and confidentiality to IPv6 datagrams. The Encapsulating Security Header is simpler than some other similar security protocols like ISO, SP3D and so on but it is also flexible and algorithm independent. In order to succeed interoperability through the Internet, the usage of DES CBC is being used as the standard algorithm for use with the IPv6 Encapsulating Security Header [15].

2.2.8 Encapsulation of IPv6 on LANs

The IPv6 and many other protocols and also IPv4 should coexist on LAN. For a great time period the designers of IPv6 wondered how to implement this coexistence and focused on two main options [13].

1. They consider that the IPv6 is the next generation of IPv4 and thus, to maintain, at the local network level, the Protocol Type equal to that of IPv4. Therefore, IPv6 packets being distinguished by the *Version* field which means for the four bits of the IP packet.
2. The IPv6 designers consider that the protocol is different at all from IPv4 and therefore to assign a *Protocol Type* different from that of IPv4.

The last solution is preferable because it is stronger and more reliable during the transition from IPv4 to IPv6, the moment when both protocols are active.

2.2.9 IPv6 on Upper Layers

The TCP/IP network architecture is not well layered, so the transition from IPv4 to IPv6 protocol has an impact on upper layers by including applications like Telnet SMTP and so on.

Initially, should be considered that applications let us to find the destination node by learning its name or its IP address. The addresses go through TCP and UDP transport protocols. TCP utilizes source and destination IP addresses like connection identifiers.

Generally, TCP and UDP should work either if the network is IPv6 or if it is IPv4. Thus, from the IPv4 period till the IPv6 period, many users are able to support both the IPv4 and the IPv6 simultaneously [13].

2.2.10 Sensor Nodes and Internet

The last approach of the Wireless Sensor Network (WSN) Community is the connection of their nodes straight to the Internet. The aim is allow the ideas of the Internet of Things community and to specify the different network protocols which used in WSNs.

At the beginning of 21st century, the researchers believed that WSNs do not have the same requirements like the one of the Internet. The reason was that there was the presumption that the layered architecture could not be used anymore due to the resource limitations. Furthermore, the researchers thought that the scalability and the robustness could only be succeeded by utilizing localized algorithms. Finally, they consider that a WSN device might not need an identity.

The thought was brought by the fact that the sensor nodes are connected with a serial interface to the outside. For this reason, an application level gateway should be installed at the root and the connection could be compared with USB. Adam et al. argues that “protocol gateways are inherently complex to design, manage, and deploy.”

In recent years the protocol of IEEE 802.15.4 is totally accepted as physical and MAC layer protocol for WSNs. A conditional network layer protocol has to count the limitations which correlated with the MAC layer protocol.

As we mentioned before, the IPv6 is not accords with the properties of the IEEE 802.15.4 protocol. However, the Internet Protocol for Smart Objects (IPSO) Alliance suggests using IP also for smart objects. The main reasons are the compatibility with a wide range of applications, the stability and the high scalability.

The above allusion explains the reason why IP is suitable on the sensor nodes but does not justify why one should use IPv6 on such devices. The explanation comes from Jonathan W. Hui and David E. Culler “IPv6 is better suited to the needs of WSNs than IPv4 in every dimension” [21]. IPv6 implements more efficient network architectures than the current solutions and this theory is based on the mechanism of IPv6 which correlating with sample listening, collection routing and hop-by-hop feedback.

3. State of the art

3.1 6LowPAN

This section presents the result of the attempt of the implementation of IPv6 in wireless sensor networks and more specifically in low power wireless personal area networks, the limitation that are raised, the most extensive implementation of IPv6 and low power wireless personal area networks and the 6LowPAN routing protocols which exist [1, 23].

Extending IP to LoWPAN was once considered impractical. This happens, because these networks are strictly constrained and should acts without observation for a huge time space on modest batteries. Many experts, through some proprietary protocols, considered that IP was too resource intensive to be scaled down to operate on low power links used in LoWPAN settings. However, 6LowPAN (IPv6 to LoWPAN) changes the computation by inserting an adaptation layer that forwards the IPv6 communication over LoWPAN links [25, 26, 27, 28].

6LoWPAN is a low cost communication network which gives the opportunity of wireless connectivity in applications with limited power and soft throughput requirements. The LoWPAN contains devices that work together in order to combine the physical environment to real world applications (wireless sensors) (Figure 14).

6LoWPAN is the name of the working group in the internet area of IETF. It is a model which its goal is to let the IPv6 packets be sent to personal area networks (PANs) and received from this type of networks, more specifically over IEEE802.15.4-standard based networks. IPv6 is referred to data delivery for wired networks. Moreover, devices of IEEE802.15.4 offer the opportunity of sensing communication in the wireless domain. Some characteristic applications are: wireless internet connectivity at lower data rates for devices with very limited form factor (e.g. automation and entertainment applications in home, office and factory environments)

The inequable properties of IPv6 and IEEE 802.15.4 cause many assumptions to the 6LowPAN protocol, which have to be considered in order that this kind of network effectuates the needs of a modern wireless sensor network [27].

The main specifications of IEEE 802.15.4 are the low power, the low bandwidth and the maximum link layer packet size of 127 bytes [23]. Applying IPv6 over IEEE 802.15.4, without changes would result in extremely small packets payloads for higher level protocols. Moreover, the header of the IPv6 has 40 bytes, which in TCP or UDP results in 41 bytes, the TCP header is 20 bytes, so only 28 bytes are available for each packet for application layer protocols [21, 24, 25].

The main assumption of 6LowPAN is related with the compression standard for the IPv6 header like for the upper layer headers. A fragmentation and reassembly layer should be inserted while the minimum MTU of 1280 bytes of IPv6. Moreover the routing protocol should not appose an overhead on data packets but should preserve the use of memory and the computation power.

Furthermore, some other requirements are related with the topology of the network. It is necessary a stateless address auto configuration in order to reduce configuration overhead. Also, many security requirements are raised by the 6LowPAN protocol like countermeasures towards man-in-the middle attacks and denial of services attacks.

Moreover, the 6LoWPAN standard is a special version of the IPv6 protocol to the limited resources of IEEE 8.15.4 devices. The main point of the 6LoWPAN layer is called dispatch header. This certain header characterizes the frame as 6LoWPAN packet and specifies the type of address is to be expected. The dispatch header is the first header in a queue of headers.

Mesh routing is used for IEEE 8.15.4 devices in order to augment their range by searching for normal nodes in order to perform like routers and forward the data packets (Figure 15). Furthermore, the mesh header assigns the source node and the final destination node, whereas the normal header is utilized for the current link [28].

6LowPANs comprise devices which are suitable with the IEEE 802.15.4 protocol. The nodes which are taking part in a 6LowPAN may confine its action only in the role of the host while other nodes take part in the process of routing. Specifically, these separate roles of host – router are related with the processing and storage capabilities of the device.

Although the above protocol of IEEE uses star and mesh topologies, the 6LowPAN format and the IEEE 802.15.4 protocol do not explain how mesh topologies could be function in this environment. So the multi-hop routing and the 6LowPAN can be supported by the IP layer or above the IP layer. Furthermore, the numbers of the protocols which are developed by many experts do not ground the demands of the multi-hop routing in 6LowPAN.

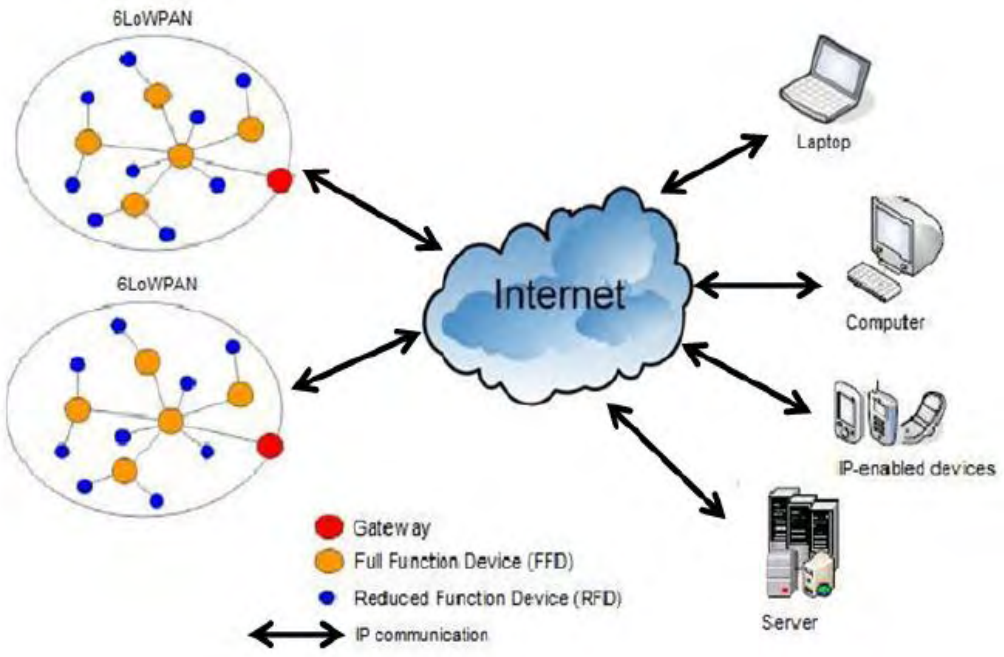


Figure 14: *An example of 6LoWPAN architecture*

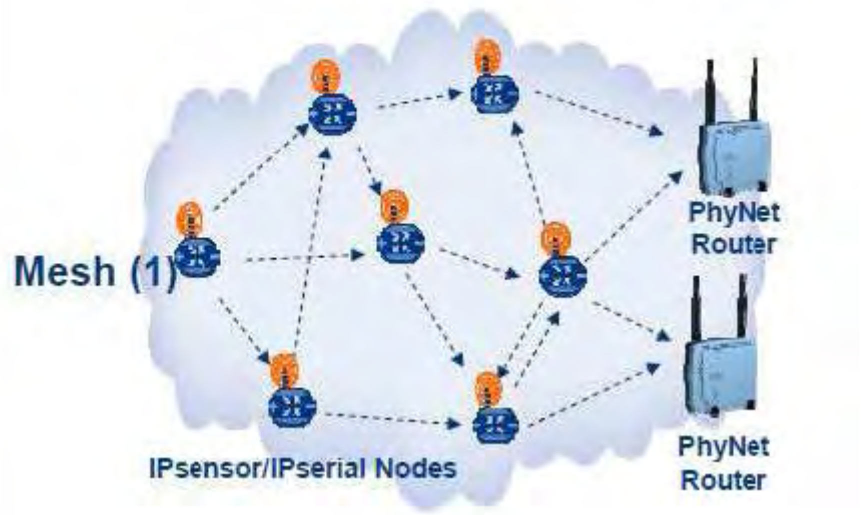


Figure 15: *An example of mesh routing network*

As we mentioned before one problem of the 802.15.4 devices apart from limited range is the frame size. A normal frame of IEEE 802.14.5 has 127 while the minimum maximum forwarding unit of an IPv6 packet should be 1280 octets. In order to solve this difference in the frame size the 6LoWPAN protocol assigns a certain process of fragmentation and reassembling IPv6 data packets to several frames.

One more process which is suitable for remaining the size of IPv6 packets in a normal level is the compression of headers. Specifically, the 6LoWPAN standard assigns a stateless compression for the link local address of the IPv6 header. This compression procedure is known as Header Compression one (HC1). For the higher level protocol another compression scheme is defined for UDP, the Header Compression two (HC2) [29].

Moreover a more important part is the multicast system. Multicast is responsible for the neighbour discovery. Thus, the 6LoWPAN protocol specifies a certain header for multicast over meshed routing for finding motes that are more than one hop far.

3.1.1 Specifications

The different properties of IPv6 and IEEE 802.15.4 provoke many requirements to the 6LoWPAN protocol, which should be consider in order the network still supplies the demands of a modern Wireless Sensor Network (WSN).

The most remarkable characteristics of the protocol of IEEE 802.15.4 are the low power, its low bandwidth and the maximum link layer packet size of 127 bytes. If IPv6 applied unaltered over IEEE 802.15.4 standard would have as a result extremely small packet payloads for higher- level protocols. According to the year 2006 the IEEE 802.15.4 standard’s maximum size of a frame would be in the worst case 88 bytes. On the other hand the IPv6 header contains 40 bytes which reach the 41 bytes for upper layer protocols as the TCP and UDP. Moreover, the length size of the TCP header is more 20 bytes. Consequently, this means that 28 bytes for each packet are available for application layer protocols (Figure 16).

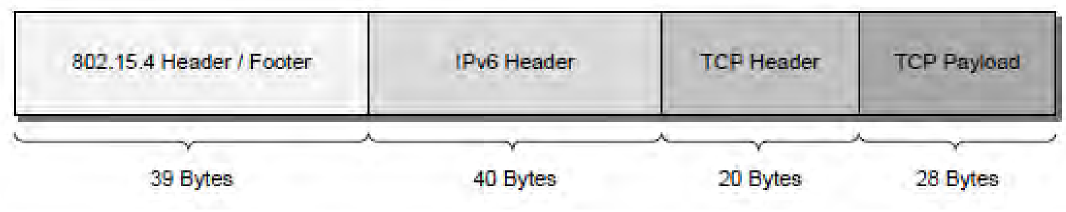


Figure 16: *802.15.4 and IPv6 header sizes*

The previous IPv6 has a minimum MTU of1280 bytes, should be introduced a fragmentation and a reassembly layer. Moreover, the routing protocol has not to demand an overhead over data packets. Because the WSN devices have a restricted performance, a possible appropriate protocol has to preserve the memory utilization and the computation power.

Except of the above requirement more requirements come from the network topology. In order to eliminate the configuration overhead, a preferable action is a stateless address autoconfiguration. Furthermore, some more requirements that correlated to the 6LoWPAN standard are the security requirements like denial of services attacks and countermeasures against man-in-the-middle attacks [30, 31].

3.1.2 Adaptation Layer and Header

In general, by utilizing compression and fragmentation of the header then the 6LoWPAN packet demands a much smaller header than an IPv6 packet would use.

The adaptation layer lies between the layer 2 and layer 3. This layer contains datagrams which should be forwarded by a header stack. The adaptation layer of 6LoWPAN standard contains mainly three parts, the header compression, the fragmentation and the layer-two forwarding. The compression is stateless and shared-context in order to eliminate the length size of the IPv6 header to fewer bytes.

The 6LoWPAN standard like the IPv6 protocol, utilizes also an encapsulated header format which contains the IPv6 header compression subheader, the fragment header and finally the mesh addressing header (Figures 17, 18, 19). At the start point of every header, a header type field assigns the header format [30, 31, 32].

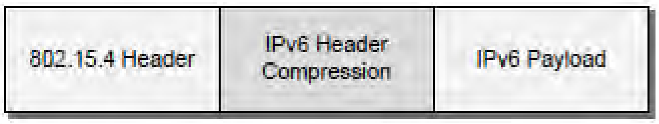


Figure 17: *Header stack → 6LowPAN utilizes only the IPv6 header compression subheader*

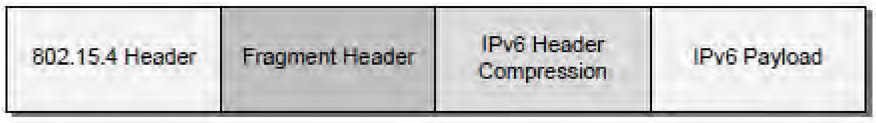


Figure 18: *Header stack → 6LowPAN utilizes only the IPv6 header compression subheader and the fragmentation header*

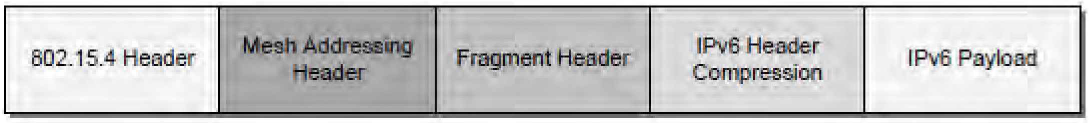
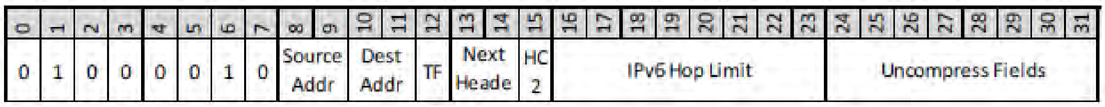


Figure 19: *Header stack → 6LowPAN utilizes only the IPv6 header compression subheader and the fragmentation header and the mesh addressing header*

3.1.3 Header Compression

The 6LowPAN header compression is characterized by a stateless compression. The stateless compression scheme is divided in two parts, the first part is the header compression one (HC1) and the second part is the header compression two (HC2). The first compression is responsible for the compression of the IPv6 header with an original size of 40 bytes into three bytes in the best case. Furthermore, the second compression, the HC2, presents a compression format in order to eliminate the length the transport protocol header. Finally, both the header compression one and also the header compression two contain one encoding byte and non-compressed fields.

The header compression one (HC1) eliminates the length of the IPv6 header. The compression expects many values for the IPv6 header fields. If the assumptions will demonstrate as wrong, the non-compressed values of the fields should follow the encoding field. The main assumption is that the version is IPv6. The destination and the source address are assumed to be link local. Also, the Flow Label and the Traffic Class fields have the value zero. Finally, the Next Header is considered to be TCP, UDP or ICMP. While there are no ways to compress the Hop Limit field, it should be carried in full (Figure 20) [30, 32].



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	1	0	0	0	0	1	0	Source Addr		Dest Addr		TF	Next Heade		HC 2	IPv6 Hop Limit								Uncompress Fields							

Figure 20: *HC1. IPv6 header compression*

The figure above presents the standard of the compression (Figure 20). The first octet acts as the dispatch byte which specifies the header format as an HC1 header. The following byte is known as HC1 encoding octet. The bits of the source and the destination address assign whenever the interface identifier and the prefix are carried in-line. The TF bit when the Traffic Class and the Flow Label are uncompressed is zero. Thereafter, the Next Header bits are zero when the fields cannot be compressed. Next, the HC2 encoding bit specifies whether a HC2 header immediately follows the HC1 encoding. The next octet, the Hop Limit field immediately follows the HC2 encoding octet or in the case where the HC2 is not presented, it is appended the HC1 octet. Finally, uncompressed fields hold the last octets.

Furthermore, the 6LowPAN protocol supports the compression of the transport protocol header, which is characterized as HC2. It is defined a compression format for UDP which eliminates the length of the UDP header to four octets from eight octets [31].

Header Field	IPv6 header length	6LoWPAN HC1 length	Explanation
Version	4 bits	------	Assuming communicating with IPv6.
Traffic class	8 bits	1 bit	0 = Not compressed. The field is in full size. 1 = Compressed. The traffic class and flow label are both zero.
Flow label	20 bits		
Payload length	16 bits	------	Can be derived from MAC frame length or adaptation layer datagram size (6LoWPAN fragmentation header).
Next header	8 bits	2 bits	Compressed whenever the packet uses UDP, TCP or Internet Control Message Protocol version 6 (ICMPv6).
Hop limit	8 bits	8 bits	The only field always not compressed.
Source address	128 bits	2 bits	If Both source and destination IPv6 addresses are in link local, their 64-bit network prefix are compressed into a single bit each with a value of one. Another single bit is set to one to indicate that 64-bit interface identifier are elided if the destination can derive them from the corresponding link-layer address in the link-layer frame or mesh addressing header when routing in a mesh.
Destination address	128 bits	2 bits	
HC2 encoding	------	1 bit	Another compression scheme follows a HC1 header.
Total	40 bytes	2 bytes	Fully compressed, the HC1 encoding reduces the IPv6 header to two bytes.

Table 4: *Comparison of IPv6 header and compressed 6LoWPAN header fields*

3.1.4 Fragmentation Header

Fragmentation header is introduced by the 6LoWPAN standard in order to support the minimum MTU of IPv6. When the payload is large enough and it does not fit into a single IEEE 802.15.4 frame, it will be fragmented into many packets.

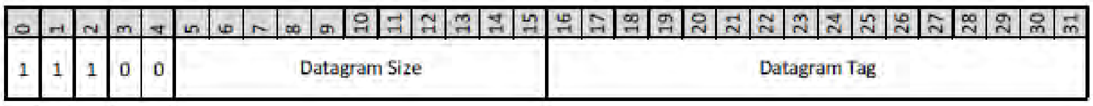


Figure 21: *Fragmentation header of 6LoWPAN, First Fragment*

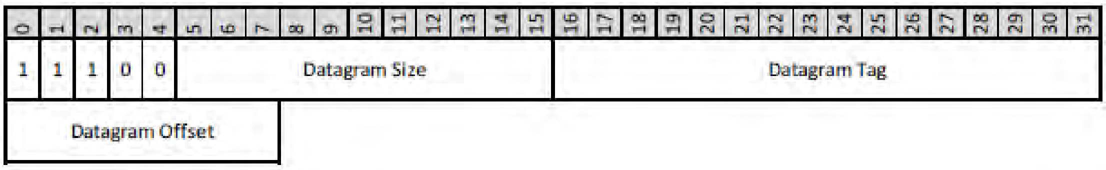


Figure 22: *Fragmentation header of 6LoWPAN, Subsequent Fragment*

The figures 21 and 22 present the fragmentation header of the 6LoWPAN standard. According to the figures, the first field is the Datagram Size, which is the size of the Entire IP packet before the procedure of the fragmentation. This field there is in every packet in order to simplify the packet handling in the case of arrivals which are out-of-order. The Datagram Tag field assigns the fragmented payload and the Datagram Offset field specifies the offset of the fragment within the original payload [30, 32].

3.1.5 Mesh Addressing Header

In the header of the IEEE802.15.4 standard is included the source and the destination address of the next hop. When a packet have to be forwarded to a node which is not a neighbour of the source, a higher-level protocol is necessary in order to implement this functionality. In IPv6 protocol the source and the destination address are included in the IPv6 header. However, with the compression header this information will be lost. A possible and appropriate solution to this problem is the introduction of another header which is called Mesh Addressing Header. This header is responsible for layer-two transmission. Moreover, the Mesh addressing header has the ability to support multi-hop forwarding of 6LowPAN payloads.

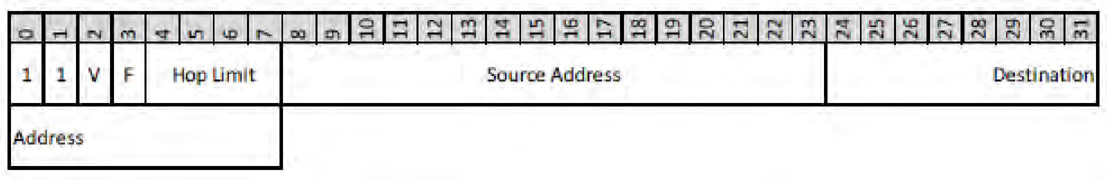


Figure 23: *Mesh addressing header of 6LoWPAN*

The V bit specifies if the Source Address is a short 16-bit address or an IEEE extended 64-bit address. Similarly happens with the F bit and the destination address. Moreover, the Hop Limit field is like to the Compression Header. Finally, in the Source Address field is included the source address of the 6LoWPAN packet and in the Destination Address field the final destination address (Figure 23) [30, 32].

3.1.6 Broadcast Header

The broadcast header is used in the case where a packet is multicast or broadcast. The structure of broadcast header is presented above in the figure 24:

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0|1|LOWPAN_BC0 |Sequence Number|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 24: *Broadcast header structure*

The sequence number of this case is used in order to differentiate between duplicate packets. When the originator transmits a new mesh broadcast or multicast packet this certain 8-bit field is increased [32].

3.1.7 Forwarding and Routing

The environment of the 6LoWPAN which we present, acts totally different from a typical IPv6 network. In LoWPAN should be consider multi-hop, power issues and other specific features. With the previous features are correlated the functions of forwarding, routing and addressing.

Thus, in a multi-hop environment the packet forwarding can be happened in the link layer or in the network layer, as 6LoWPAN can support both approaches. On the network layer the forwarding is known as route over, and its most important advantage is that can be utilized all the existing capabilities of the IP. However, with the route over forwarding the full features of the header compression cannot be used, as the IPv6 addresses may have to be forwarded with every multi-hop packet. One more existing problem is that forwarding on the network layer may not react directly to possible changes in the link state.

In order to face these problems, the 6LoWPAN protocol, as we mentioned above, supports forwarding also on the link layer which is known as mesh under. As far as the source is concerned, if the destination node is reachable by the source node directly then the data packet is transmitted to the destination. However, when the destination node is not accessible by the source node, then the source node should to contain the Mesh Addressing header. The destination and the source link-layer address are included in the Mesh Addressing header, while the header of IEEE 802.15.4 protocol will include the forwarder’s link-layer address. A node which accepts a frame with a Mesh Addressing header controls the final destination. In the case where the node itself is not the final destination, it places the address of the next hop in the destination field of the IEEE 802.15.4 header and forwards the packet.

More explicitly, as far as the routing is concerned, is the ability to transmit a packet of information from one node to another by specifying a certain path via the network while meeting some specific efficiency criteria. With a certain 6LoWPAN header format, routing protocols can be presented at two layers which are described below:

- Mesh-under Routing: The mesh-under routing is responsible for routing under the IP link and is directly based on link-layer standard 802.15.4 standard. It uses 64-bit IEEE extended address or 16-bit short address. This routing is referred only to a 6LoWPAN network (Figure 25).
- Route-over Routing: This certain type of routing, the route over routing, is implemented at the network layer of 6LoWPAN stack. It uses IP address for addressing and locating nodes. The route-over routing has the ability to access nodes outside the 6LoWPAN as well as inside. This type of routing is almost same as used in regular IP networks (Figure 25).

In the case of fragmentation or overhead of reassembly, the routing data should match into a single physical frame of the 802.15.4 protocol and the application data should not match in frame that they do not fit anymore. . The link-state protocol and also the distance vector routing protocol are not fitted well to the 6LoWPAN networks. Lately, the IEFT with the working group of Routing over Low Power and Lossy Links (ROLL) are deal with the problem and are trying to find an appropriate routing protocol [30, 33].

```
+----------------------------+  +-----------------------------+
|      Application Layer     |  |      Application Layer      |
+----------------------------+  +-----------------------------+
| Transport Layer (TCP/UDP)  |  |  Transport Layer (TCP/UDP)  |
+----------------------------+  +-----------------------------+
|     Network Layer (IPv6)   |  |  Network       +---------+  |
+----------------------------+  |  Layer         | Routing |  |
|  6LoWPAN                   |  |  (IPv6)        +---------+  |
|  Adaptation                |  +-----------------------------+
|  Layer        +----------+ |  |  6LoWPAN Adaptation Layer   |
+---------------| Routing* |-+  +-----------------------------+
| 802.15.4 MAC  +----------+ |  |        802.15.4 MAC         |
+----------------------------+  +-----------------------------+
|        802.15.4 PHY        |  |        802.15.4 PHY         |
+----------------------------+  +-----------------------------+
```

Figure 25: *the 6LowPAN routing in the whole network stack*

In the case where a route over mechanism is in use through a multi-hop communication, all routers act IP routing in the stub network. Specifically, the link-local scope canopies the number of nodes via the symmetric radio range of a node. Moreover in the case of a mesh under configuration, the only IPv6 router in the LowPAN is the 6LBR. Thus, the IPv6 link-local scope contains all nodes in the LowPAN. After all, the mesh under mechanism should be appropriate for the support of a multihop transmission (Figure 26).

```
     h   h
    /    |                       6LBR: 6LoWPAN Border Router
6LBR -- 6LR --- 6LR --- h         6LR: 6LoWPAN Router
        / \                         h: Host
       h  6LR --- h
           |
          / \
       6LR - 6LR -- h

     h   h
    /    |                    6LBR: 6LoWPAN Border Router
6LBR --- m --- m --- h           m: mesh under forwarder
        / \                      h: Host
       h   m --- h
           |
          / \
         m - m -- h
```

Figure 26: *route over and mesh under LowPAN*

It is useful to notice that in route over and mesh under network there is no a case of topologically based address assignment in the 6LowPAN. Specifically, the addresses usually based on EUI-64 addresses and other times based on 16 MAC addresses [34].

3.1.8 Addressing

The process of addressing in 6LoWPAN protocol mainly based on the stateless address configuration of IPv6. The RFC 4944 describes with details the way which the Interface Identifier is derived. Broadly, it is based on the IEEE EUI-64 address of the IEEE802.15.4 device. Furthermore, in mesh-link network the link-local address is utilized in collaboration with the LoWPAN and routable address are used in order to communicate outside.

The Neighbor Discovery Protocol in 6LoWPAN like in IPv6 contains prefix discovery and default route configuration. However, the protocol is not suited well to the 6LoWPAN requirements and maybe will generate an overhead in the amount of messages [30].

3.1.9 Stateless Address Autoconfiguration

The 6LoWPAN standard has the capability to support stateless address autoconfigurartion. This is very important because a user can create its own address by using a combination of locally information which is available and information which advertised by routers, without making a certain binding with routers. While each device of 802.15.4 standard has a unique EUI-64 identifier, on the other hand an IPv6 interface identifier can be obtained from this EUI-64 identifier using stateless autoconfiguration.

Though each device has a EUI-64 address, every device may have another 16-bit short address after the communication with a PAN coordinator. These certain addresses have to be unique within a PAN. Moreover, there is the possibility of using the 16-bit addresses for autoconfiguration (Figure 27). In this case, is formed a pseudo 48-bit address by binding this address with the 16-bit PAN ID and 16 zero-bits in sequence. More explicitly: *16_bit_PAN:16_zero_bits: 16_bit_short_address*. Furthermore, an interface identifier can be created with the 48 bits above by utilizing IP over Ethernet and also the IPv6 link-local address are shaped by adding the interface identifier to the prefix FE80: : /64 (Figure 28).

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     16-bit short Address      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 27: *16-bit short address*

```
  10 bits            54 bits                  64 bits
+----------+-----------------------+----------------------------+
|1111111010|        (zeros)        |    Interface Identifier    |
+----------+-----------------------+----------------------------+
```

Figure 28: *The formation of IPv6 link- local address*

3.1.10 6LoWPAN Neighbor Discovery

The Neighbor Discovery protocol is refereed to an Internet Protocol Suite which is used with IPv6. This certain protocol is used by the nodes in order to find other nodes on the link, to specify the link-layer address of other on-link nodes, to discover the routers that are available and final to maintain information on their reachability. In the case where a router or a path to a router fails, the node tries to discover for a functioning alternatives [35]. The Neighbor Discovery protocol for IPv6 assigns mechanisms which are related with:

- Router Discovery: nodes are able to locate routers that are on an attached link.
- Prefix Discovery: nodes are able to find the set of address prefixes that are used in order to specify their totally unique address.
- Address Resolution: this mechanism related with the mapping to link layer address. This is not appropriate in 6LoWPAN since there is direct mapping between their unique address and the link-layer address.
- Next-hop Determination: in this mechanism the nodes try to find next hop routers for a destination.
- Neighbor Unreachability Detection (NUD): in this mechanism the nodes are able to specify that a neighbour is no longer available on the link.
- Duplicate Address Detection (DAD): in this mechanism the nodes have the ability to control if an address is already in use.
- Address Autoconfiguration: in this mechanism the nodes have the ability to configure their address automatically without the use of a stateful configuration protocol like DHCPv6 (Dynamic Host Configuration Protocol for IPv6).

In LoWPAN the address resolution for neighboring is not radically possible. According to the physical mobility of nodes or the radio strength, a node may go from one node to another. Furthermore, the characteristics of the 6LoWPAN standard are different from the classic networks, thus the Neighbor Discovery protocol is not suitable and appropriate for the 6LoWPAN. Consequently, for this case a 6LoWPAN-specific ND is proposed. In this proposal are introduced new entities which are related with the mechanism of neighbor discovery. These entities are described above:

- Edge Router: this mechanism is related with an IPv6 router which connects a 6LoWPAN mesh to another IP network.
- Whiteboard: is about a data structure which is supported by Edge Routers. The Whiteboard is used as NUD and DAD through the entire 6LoWPAN mesh. This entity contains information about a node regarding its Owner Interface Identifier, IPv6 address, Transaction ID, Owner Nonce and remaining time of binding.
- Node Registration: this entity refers a method in which nodes in LoWPAN register with Edge Routers, thus forming Whiteboard bindings of their all IPv6 addresses in LoWPAN.

The Neighbor Discovery for 6LoWPAN specifies some message formats along with altering standard ND messages for IPv6. Above are described some types of messages which are used with modification from:

- Router Solicitation: in this case, the nodes send this message in order to request a Router Advertisement immediately.
- Router Advertisement: in this case, the routers send this message periodically or in response to Router Solicitation. In Routers Advertisements are included a lot of link and Internet parameters as well as prefixes information.
- Neighbor Solicitation: in this case, the nodes transmit this message in order to determine the link layer address of their neighbors or confirm their reachability. Moreover, the Neighbor Solicitation is suitable for Duplicate Address Detection. Only used is extended LoWPAN
- Neighbor Advertisement: in this certain case, the nodes transmit this message in response to Neighbor Solicitation. Furthermore, they transmit in order to announce link layer address alteration. Only used in extended 6LoWPAN. The extended 6LoWPAN is an aggregation of many 6LoWPAN which are interconnected by a backbone link through Edge Routers and farming a single subnet.
- Node Registration: in this type of messages, a node transmits this message along with its IID and requested lifetime in order to register at an Edge Router. This specific message can contain possible options as the address option which contains the address of the node which wants to register.
- Node Confirmation: in this case an Edge Router transmits this message to the node in response to its Node Registration. This is used in order to certify the binding of the node at the Edge Router.

The last two types of messages are assigned in Neighbor Discovery for 6LoWPAN.

3.1.11 Security

In 802.15.4 networks the nodes have the ability to perform either in secure mode or non-secure mode. In order to achieve different security objectives two security modes are specified in the specification. The two security modes are the following, the *Access Control List (ACL)* which supports limited security services and requires every in order to keep its own ACL. This certain mode accepts receiving frames which are characterized as trusted nodes, only from nodes which are present in the device's ACL. The frames from nods which are not registered should be filtered. Nevertheless, this mode does not support cryptographic protection. The other security mode is the *Secure mode*. This mode supports all the security service according to the appropriate security suite. This certain mode offers message integrity, confidentiality of the frame, sequential freshness and access control.

The issue of security acts like a bit tradeoff in the 6LoWPAN, as security is always a costly function. This point is mainly certified in the LoWPANs. In the case where a IPv6 putting on the top of the 6LoWPAN it is probably appropriate to use IP security protocol and turn off the security mechanism assigned by IEEE 802.15.4. On the other hand the IP security protocol is mature for the services at IP or upper layers. Thus, according to their innate properties and limitations the 6LoWPAN presents totally unique characteristics which traditional security techniques cannot be supported directly.

The attacks against the 6LoWPANs are separated in two groups in external and internal attacks. In the first group, the enemy is not authorized of the 6LoWPAN. Furthermore, external attacks can also be separated in two subcategories, the passive attacks and the active attacks. On the one hand the passive attacks try to discover important information by involving eavesdropping on network's radio frequency range. On the other hand during an active attack, a denial-of-service attack at the physical layer has the possibility to present important consequences [36].

Moreover an enemy has the ability to make a node of a 6LoWPAN inefficient or capture it and in the following of this action the enemy extracts the key and uses it for eavesdropping or cue at some instances.

The applications that are related with the 6LoWPAN usually demand confidentiality and integrity protection. This certain requirement can be provided at the application, network, transport or/and link layer. In each case, there will be some constraints that will influence the right choice of the appropriate protocol. Some of the restrictions which affect the selection of the protocol are the low power operation, the small bandwidth requirements, the small code size and the low complexity.

Due to the previous limitations a certain model for 6LoWPAN devices should be created in order to face any risks while developing meaningful assumptions and simplifications. Some cases which are related with the treats that should be considered are denial of service attacks and man-in-the-middle attacks.

According to the initial key establishment and protocols for subsequent key management as well as to secure the data traffic do fall under the control and the jurisdiction of 6LoWPAN. The alternative choices like TLS, IPsec, etc, should be considered due to the 6LoWPAN limitations.

The most devices of 802.15.4 have already support for AES link-layer security, thus is very reasonable the using of link layer security. AES is block cipher which operates on groups of standard length like 128 bits. In order to encrypt messages with longer size, many different modes of operation may be used. The versions of modes which are presented as CBC, CFB, ECB, OFB assign only confidentiality and no message integrity. Other modes like CCM mode have the ability to provide both confidentiality and message integrity. Any 6LoWPAN network has the ability to operate in any of the previous modes. However, the most suitable solution is the use of the most secure mode which is available for link layer security and build on it.

For the network layer security the models that are most operable are two and are the end-to-end security by using for example IPsec transport mode and the security that is limited to the wireless portion of the network by using a security gateway and IPsec tunnel mode. The second model has a disadvantage which is the larger header size which is significant at the 6LoWPAN frame MTUs. Generally, in order to simplify the implementations of the 6LoWPAN model, it is necessary to identify the appropriate security model and determine the appropriate set of cipher suites according to the given limitations.

Generally, some possible attacks against 6LoWPANs are intrusion replay and sink-hole attacks. Specifically, in 6LoWPANs the enemy has the ability to “hit” the routing mechanisms mainly by informing the WPAN networks with false data. This has as a result the instability of the routing mechanism. Moreover an unauthorized node has the ability to “spy” the packets and then aggress with replay attacks against the nodes of 6LoWPAN. These threats may cause many harmful problems and mainly when the attacker is a high-power device like a laptop. Finally, has the capability to drain the batteries of the 6LoWPAN devices by redirecting routes and transmitting broadcast messages.

A hopeful and possible solution for facing security vulnerable spots is the implementation of application level security. Thus, by this way the application level security protects from another spy and the link layer security protects from intrusion.

3.1.11a Requirements

The necessary requirements of security in 6LoWPANs would be the above.

Data Integrity→ this attribute certifies that the data which received is not altered by an enemy

Data Confidentiality→ creates information that is not accessible by users that are unauthorized

Data Authentication→ this attribute certifies the user that the data are originated from authorized and trusted sources

Assurance→ this attribute transmit different information at different assurance levels

Robustness→ certifies the operation of the network without pause despite failure of nodes or attacks

Resistance→ this attribute has the capability to avert the enemy from having total control of the network

Availability→ this attribute certifies the survival of the network services only to authorized users and devices in spite of DoS attacks

Energy Efficiency→ a security scheme should be energy efficient in order to augment network lifetime

Resiliency→ this attribute has the capability to specify and maintain an acceptable level of security in case some nodes are compromised

3.1.11b Attacks

The main percentage of the attacks and the threats against the data security and the user cause much destruction. Furthermore the 6LoWPAN is very sensitive to physical attacks. This means threats due to relocation, masking and node destruction. After the offensive of a physical attack a lot of 6LoWPAN nodes can be knocked out permanently and this will cause definitive losses. Moreover, the physical attack has the ability to extract cryptographic secrets, permit the enemy, the malicious node to take the total control and change programming in the nodes.

Furthermore, 6LoWPAN areas can accept many DoS attacks in different layers. There are attacks against the physical layer as we mentioned before and also there are attacks against MAC layers. The MAC layer pertains to exhaustion and collision. The devices try to be in a sleep mode in order to conserve energy, because usually they low rates of energy. Thus, this condition has as a consequence the appearance of many attacks against the battery of the devices.

Moreover, there is one more type of attack which is against network availability. In this type of offensive the enemy has the ability to eliminate the network performance and degrade the throughput.

The attacks in the network layer can be divided in the following categories:

Selective forwarding→ in this certain attack the enemy which will be a malicious device do not transmit some messages or transmit certain messages.

Wormhole attack→ in this type of attack the enemy records bits at one location and then tunnel them to another location. The wormhole attack can be performed in the phase where the nodes of 6LoWPAN try to find their neighbors.

Sinkhole attack→ in this type of attack, the enemy which is a malicious device has the ability to get all traffic from a certain area and this will result in a DoS attack.

Spoofed, altered, or replayed routing information→ in this certain type of attack the enemy which will be a malicious node utilizes spoofing, altering or replaying in order to aim routing information which transmit between nodes in an attempt to create routing loops, extend or shorten source routes, attract or push network traffic, create false messages and so on.

Neighbor discovery attack→ this attack is referred to a modified version of IPv6 Neighbor Discovery protocol. This attack comprises unsecured router advertisement and neighbor discovery DoS attacks.

Sybil attack→ in the Sybil attack, a single node presents many identities to other nodes in WPAN. This certain type of attack hit with a definite threat a geographic routing protocol and attack against the routing mechanism, distributed storage, misbehavior detection, data aggregation and so on. It is very important to mention that the Sybil attack cannot be detected easily in progress.

At the transport layer, the enemy which is a malicious device has the ability to create false messages carrying sequence numbers or control flags which will ultimately cause the endpoints to request retransmission of missed frames.

3.1.12 6LoWPAN Implementation

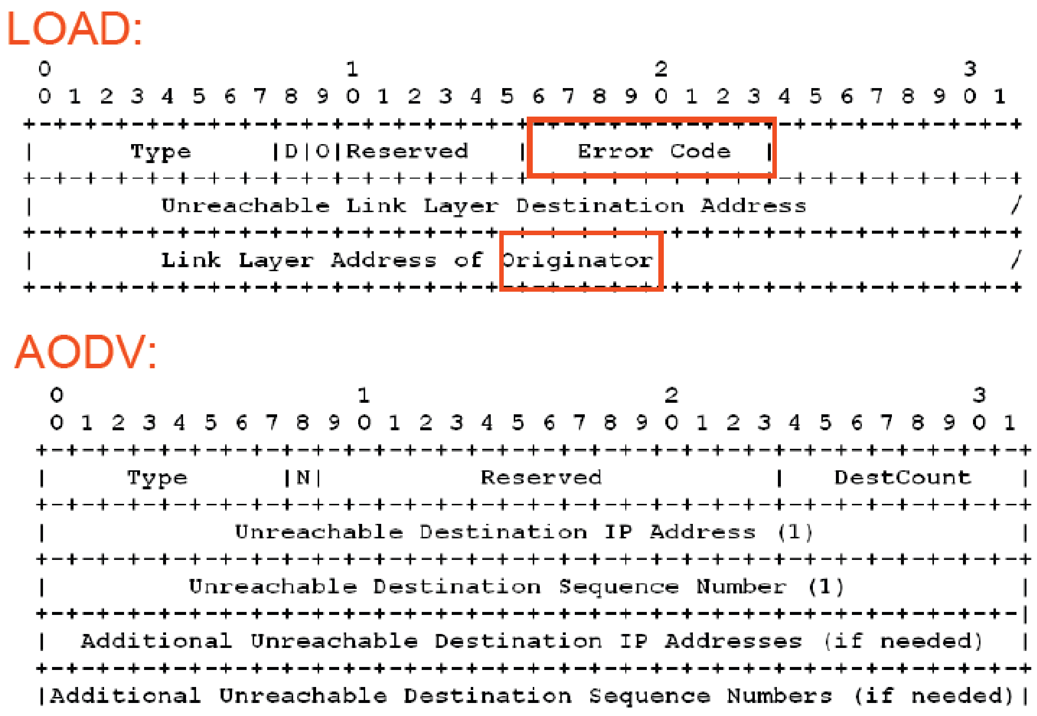
There are many versions of OS for WSNs. TinyOS is one of the most widely used and has many types of functionalities for sensor nodes. Specifically, the most known implementation of IPv6 and LowPAN for TinyOS is known as blip. This 6LowPAN implementation of TinyOS is being developed by the University of California in Berkley, as it is mentioned in the paper "*IPv6 for Wireless Sensor Networks*". Although, it implements a variety of functionalities, it only supports platforms with RF transceiver. Thus, the blip it has been decided to be port to the RF transceiver family. This process separated into two sub processes, the extension of current radio driver of RF transceiver and the adjusting of blip to a new driver. The blip is inserted by presenting its functionalities and its limitations. In the same paper, Lars Schor, extend TinyOS implementation by inserting the blip to the new platforms [37, 38, 39, 40].

3.1.13 6LoWPAN Routing Protocols

There are some routing protocols for 6LowPAN but not a great number because the 6LowPAN is a very recent issue. The 6LowPAN routing protocols which are appeared in relevant papers most often and the protocols which raise the highest interest are two and they are the LOAD and the DYMO-low routing protocol.

As we mentioned before the issue is quite recent so there are not many sources over this issue. Nevertheless according *Ki-Hyung Kim (Ajou University), S. Daniel Park (SAMSUNG Electronics), G. Montenegro (Microsoft Corporation), S. Yoo (Ajou University), Karl Mayer*, *Wolfgang Fritche and Vassil Stefanov*, 6LoWPAN Ad hoc

Routing On-Demand Distance Vector Routing Protocol (LOAD) is a simplified on demand routing protocol based on AODV for 6LoWPAN. Its basic characteristics are that it uses16 bit addresses and broadcast in the route discovery and on the other hand it does not use destination sequence number and the local repair. Moreover it reports back to the originator by RERR (Route Error) upon a link break (Figure 29). Specifically, does not maintain the precursor list and sends RERR only to the originator of the data which caused the link break. Also, it uses the route cost by utilizing the LQI of the 6LoWPAN physical layer and it uses the Acknowledged transmission option for keeping the connectivity of a route. Finally it consists two tables the Routing table and the Route Request table [38].



```
LOAD:
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |D|O|Reserved   |   Error Code  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Unreachable Link Layer Destination Address             /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Link Layer Address of Originator                       /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

AODV:
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |N|          Reserved           |   DestCount   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Unreachable Destination IP Address (1)             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Unreachable Destination Sequence Number (1)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-|
|  Additional Unreachable Destination IP Addresses (if needed)  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Additional Unreachable Destination Sequence Numbers (if needed)|
```

Figure 29: *Route Error Message*

According to *Ki-Hyung Kim (Ajou University), S. Daniel Park (SAMSUNG Electronics), G. Montenegro (Microsoft Corporation), I. Chakeres (Boeing Phantom Works), S. Yoo (Ajou University) and Vassil Stefanov*, DYMO-low (Dynamic MANET On-demand for 6LoWPAN) routing protocol obviates UERR (Unsupported Element Error). DYMOcast is mapped as broadcast. There is no path accumulation and only the final destination responds. Moreover, DYMO low allows Multiple Routing Elements (RE) and by this way there is the possibility of eliminating the number of control messages by aggregation. Finally, DYMO low protocol set a limit on the number of control message, inserts the Error Code field into RERR, uses LQI for route cost calculation and does not use HELLO message and Sequence Number.

<u>3.1.14 AODV (Ad-hoc On-demand Distance Vector routing)</u>

AODV is one of the most studied routing protocols for MANETs. Furthermore, it is a reactive routing protocol. In the case where a node needs a route; it starts a process about the discovery of a route by broadcasting Route Request (RREQ) messages. The time when the node receives the RREQ message, it creates and transmits a Route Reply (RREP) message back to the source node. Each node has route entries which contain the forward and backward next hop information. This information will expire when the path becomes inactive. For every node there is an pre-existing list which includes the nodes which utilizes this one as the next hop in the path to a certain destination. The AODV routing protocol uses the metric of hop count [37].

In the case where a link breaks through an active path, then the upstream node which is responsible for the detection of this break may locally rectifies the route if the destination is close in number of hops to the node. In the where this attempt is not successful the upstream node creates and send a Route Error (RERR) message which contains the destinations that are not approachable. Thus the source node of the active path tries to search a new route.

A node of an active route periodically transmits local HELLO messages in order to communicate with other nodes. Then, if the neighbour node does not send back a reply, this means that the link is broken [40, 41, 42].

<u>3.1.14a AODV and LoWPANs</u>

AODVjr → AODVjr is on of the earliest routing protocols and according to the scientists its action and reaction are like AODV’s. However, this routing protocol has not as a metric the hop count. The route related to the first RREP message which received by the chosen source, instead. Moreover, in this case the pre existing lists and the HELLO messages are not exist. In AODVjr, if the communications are unidirectional, the destination node forward CONNECT messages the originator. In the case where the communications are bidirectional then no messages are used. Generally, when the source does not receive messages from the destination, then confirm a link break.

AODVbis → This routing protocol is an iteration of the AODV which wants to define some aspects related to the functionality. The use of the RREP messages is not compulsory as well as the hop count is not the compulsory metric. Moreover, the AODVbis do not include the local repair. In this certain case there is the capability of path accumulation which gives the ability to the node to learn routing information from RREQ and RREP messages.

LoWPAN – AODV → As mentioned by its name the LoWPAN – AODV is the most appropriate routing protocol for the LowPANs. This certain type is a combination of the two previous routing protocols.

TinyAODV → This routing protocol is appropriate for devices which running TinyOS. In the certain protocol only one node, known as sink node, could be the final destination of any data transmission. Furthermore, the TinyAODV has the ability to develop a communication between any pair of nodes of the network. Thus, in the case where there is not exist any route entry for the aim and a data packet must be sent, route discovery is performed, but the packet requiring the route is discarded.

3.1.15 LOAD

The 6LoWPAN Ad Hoc On-Demand Distance Vector Routing (LOAD) [3] is another routing protocol which is referred to the 6LoWPANs. This certain routing protocol should be implemented only on Full Function Devices (FFDs). In this case, only the final destination node can transmit RREP messages and the local repair is utilized. If the last action is not successful the node which defines the link break has the ability to create and unicast a RERP message to the source (Figure 30). In the LOAD the accumulated link cost from the source to the destination is the count metric. This metric uses the Link Quality Indicator (LQI) of IEEE 802.15.4 physical layer as an input for its calculation [38, 40, 41, 43].

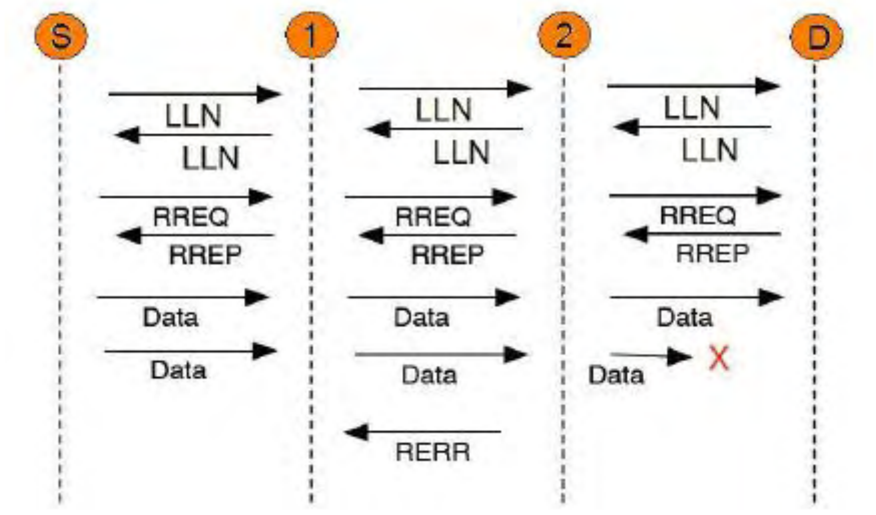


Figure 30: *Message exchange in LOAD routing protocol*

3.1.16 DYMO - Dynamic MANET On-demand for 6LoWPAN Routing (DYMO-low)

The DYMO is one more routing protocol which is based on AODV with some specificity. In the same way like AODV, the DYMO implements route discovery and maintenance through RREQ, RREP and RERR messages. During the process of route discovery the RREQ and RREP messages aggregate routing information from every intermediate node. On the other hand, although the DYMO transmits HELLO messages in order to keep the path of link connectivity, it does not use the local repair. Moreover, DYMO is placed on top of IP using User Datagram Protocol (UDP). Nevertheless, it can not be fitted to the 6LoWPAN directly because of its great power consumption and memory size. So, the DYMO routing protocol should be modified in order to be appropriate for 6LoWpan environment (DYMO-low). Thus, the DYMO-low operates on the link layer directly in order to develop a mesh network topology of 6LoWPAN devices secret to IP, such that IP sees the WPAN as a single link. In this case all the devices of 6LoWPAN operate on the same IPv6 link and share the same IPv6 prefix. Moreover, the DYMO-low routing protocol utilizes 16-bit link layer short address or IEEE 64-bit extended address (EUI-64). Finally, the DYMO-low has almost the same function with the LOAD with the difference that the DYMO-low does not use the local repair and the route cost accumulation [38, 41, 44].

3.1.17 HiLow (Hierarchical routing)

HiLow is another routing protocol for 6LoWPAN which is used in order to increase the network scalability. This routing protocol uses 16-bit unique short address as interface identifier in order to increase the memory saving and the scalability. In this case, when a device wants to join a 6LoWPAN, as first step it tries to find an existing 6LoWPAN. Then, if there is not an existing network the device becomes the initiator of a new 6LoWPAN and assigns its short address by 0. In the case where there is an existing 6LoWPAN, the new device (child) tries to communicate with the existing device (parent) at the MAC layer in order to receive a 16-bit short address (Figure 31) [45, 46]. In this case there is an appropriate equation which presented above:

$C = MC * AP + N, \quad (0 < N \leq MC).$

According the previous equation the C is the child-device, the MC is the maximum number of devices a parent-device can have, the AP is the address of the parent-device and the N is the nth child-device.

In the HiLow protocol each node knows its depth. Moreover the node which accepts a packet is known as current node. The current node initially specifies if it is either the ascendant or descendant nodes of the destination and then specifies the next hop node to forward the packet. Finally, in the HiLow routing protocol if exists a link break there is not a repair path mechanism like in LOAD and in AODV (Tables 5, 6, 7).

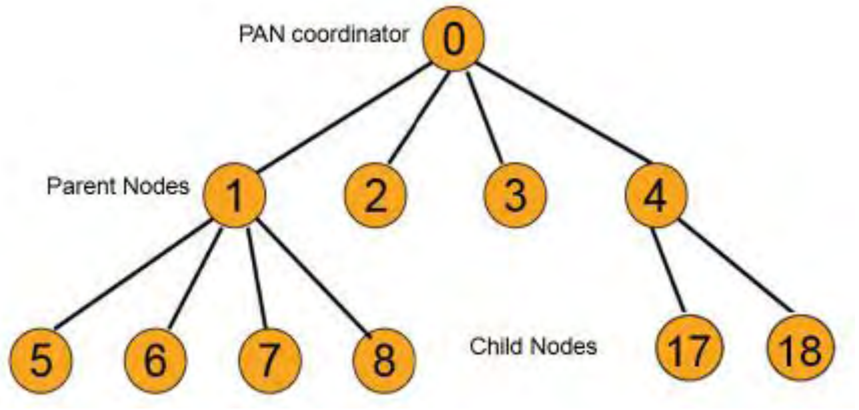
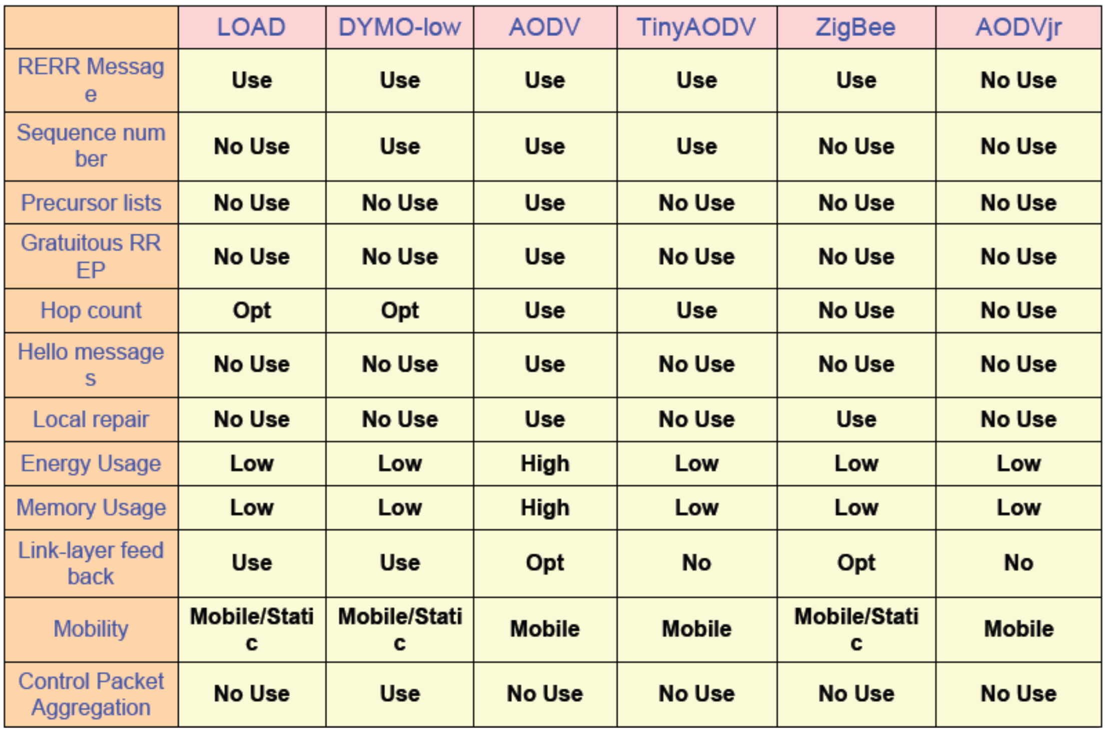
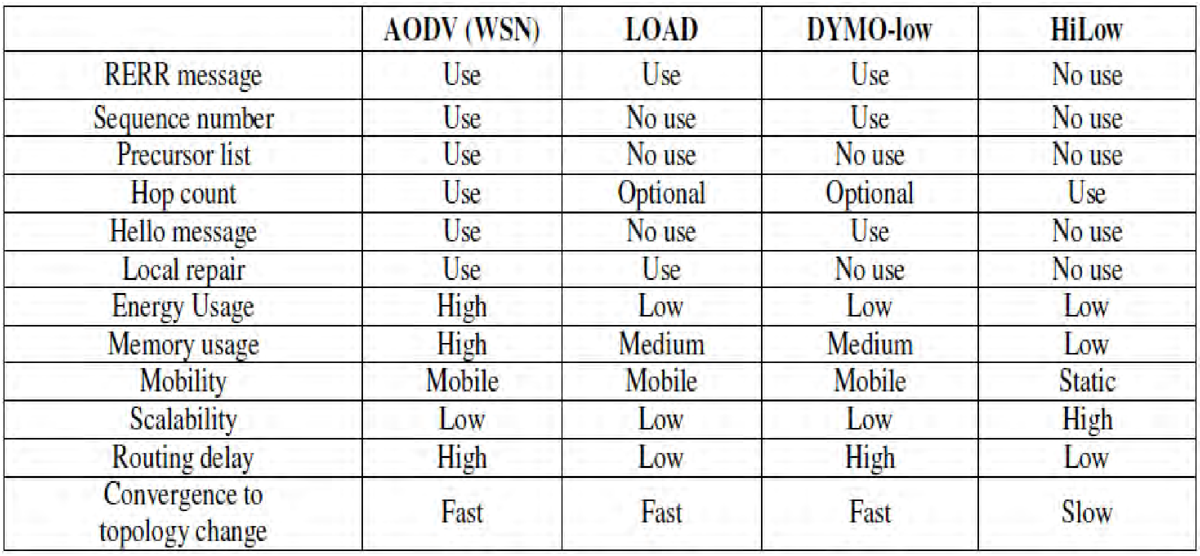


Figure 31:Routing structure of HiLow



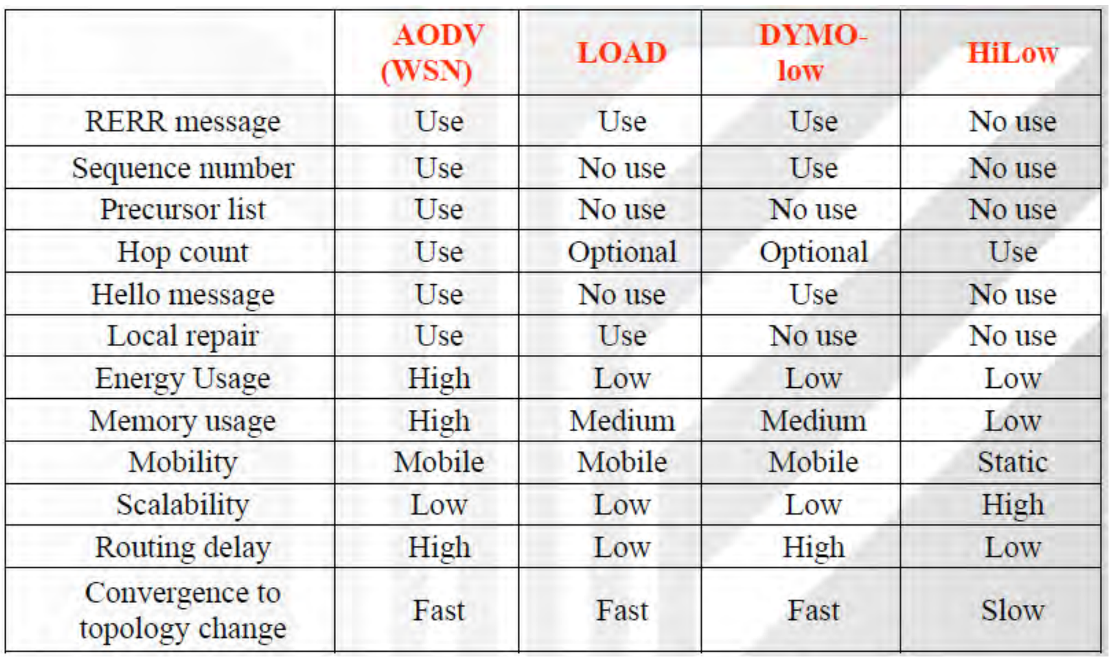
	LOAD	DYMO-low	AODV	TinyAODV	ZigBee	AODVjr
RERR Message	Use	Use	Use	Use	Use	No Use
Sequence number	No Use	Use	Use	Use	No Use	No Use
Precursor lists	No Use	No Use	Use	No Use	No Use	No Use
Gratuitous RREP	No Use	No Use	Use	No Use	No Use	No Use
Hop count	Opt	Opt	Use	Use	No Use	No Use
Hello messages	No Use	No Use	Use	No Use	No Use	No Use
Local repair	No Use	No Use	Use	No Use	Use	No Use
Energy Usage	Low	Low	High	Low	Low	Low
Memory Usage	Low	Low	High	Low	Low	Low
Link-layer feedback	Use	Use	Opt	No	Opt	No
Mobility	Mobile/Static	Mobile/Static	Mobile	Mobile	Mobile/Static	Mobile
Control Packet Aggregation	No Use	Use	No Use	No Use	No Use	No Use

Table 5: *Comparison of Routing Protocols for 6lowpan (1)*



	AODV (WSN)	LOAD	DYMO-low	HiLow
RERR message	Use	Use	Use	No use
Sequence number	Use	No use	Use	No use
Precursor list	Use	No use	No use	No use
Hop count	Use	Optional	Optional	Use
Hello message	Use	No use	Use	No use
Local repair	Use	Use	No use	No use
Energy Usage	High	Low	Low	Low
Memory usage	High	Medium	Medium	Low
Mobility	Mobile	Mobile	Mobile	Static
Scalability	Low	Low	Low	High
Routing delay	High	Low	High	Low
Convergence to topology change	Fast	Fast	Fast	Slow

Table 6: *Comparison of Routing Protocols for 6lowpan (2)*



	AODV (WSN)	LOAD	DYMO-low	HiLow
RERR message	Use	Use	Use	No use
Sequence number	Use	No use	Use	No use
Precursor list	Use	No use	No use	No use
Hop count	Use	Optional	Optional	Use
Hello message	Use	No use	Use	No use
Local repair	Use	Use	No use	No use
Energy Usage	High	Low	Low	Low
Memory usage	High	Medium	Medium	Low
Mobility	Mobile	Mobile	Mobile	Static
Scalability	Low	Low	Low	High
Routing delay	High	Low	High	Low
Convergence to topology change	Fast	Fast	Fast	Slow

Table 7: *Comparison of Routing Protocols for 6lowpan (3)*

3.1.18 Vulnerable Points

As I mentioned before few routing protocols have been developed for the 6LoWPANs, however these protocols have some vulnerable points and they are not able to satisfy the requirements of the 6LoWPANs. The reasons for this situation are presented below.

- The 6LoWPAN nodes have specific roles and types. For example there are power-affluent nodes, nodes which draw their power from primary batteries, data aggregators and so on. The 6LoWPAN routing protocols should support multiple device types and roles.
- The process of routing in LoWPAN might mean a harder problem than routing in higher-performance networks. For this certain procedure are necessary stable operation in lossy environments, power optimization, data-aware routing and so on. Unfortunately, there is not a specific routing protocol which can satisfy all the previous demands.
- The process of routing in 6LoWPAN may mean a simpler problem than routing in higher-performance networks. The LoWPANs can be transit networks or stub networks. In the case where the LoWPANs can be transit networks, routing protocols may be drastically simplified.

- The nodes which are known as handling sleeping are very important in LoWPANs more than in traditional ad-hoc networks. These nodes have the ability to be in a sleep mode for a lot of time. Taking advantage of appropriate time for transmissions is important in forwarding the efficient number of data packet.
- The strictest demands come from LoWPANs and they related to higher performance or non-battery-operated networks. The nodes of 6LoWPAN have low processing power, small memory sizes and are running on very limited power which comes from primary non-rechargeable batteries. Generally, the lifetime of a node is defined by the lifetime of its battery.

Thus, the main requirements for the routing protocols are the *low routing overhead*, the *support for sleeping nodes considering battery saving*, the *low overhead on data packets* and the *minimal memory and computation requirements*.

4. Ethics

In this area of the thesis the ethics has to deal with the security and privacy. Both security and privacy are two areas that should be studied in order people trust this type of network and certain routing protocol. It is very important for all the data packets that are transmitted over this type of network to reach their destination. Moreover, very important are the integrity, the authentication and the encryption of the data.

A remarkable example will be the files of an enterprise. These files are important as they have confidential data for the employers and for the actions of the enterprise. Thus, the enterprise will be harmed if it looses its files. Another example is the data of a hospital. A hospital would want to transmit the therapy of some patients. In this case, the problem of the loss or of the alteration of the data over the certain network will cause health problems.

Another statement which related with the topic of the ethics is the right use of the “IPv6-Wireless Sensor Networks”. For example, the deployment of such an advanced network in a country which is not so developed will raise many challenges. However, it will also raise many social problems and disparities.

Thus, for the above reasons, it is absolutely necessary to have authentication, integrity and in general security and privacy in IPv6 wireless sensor networks which also raise the acceptance of the networks from the public.

5. Resources

This dissertation includes two parts: the theoretical part and the implementation part. We do not have the opportunity to develop a hardware application.

As far as the theoretical part is concerned, it required a deep research of published literature which is related to our subject of thesis, IPv6 in wireless sensor networks and 6LowPAN routing protocols. Some other relative field that could be searched are the IPv6 protocol, low power wireless sensor area and so on. The finding of all these necessary information would be by the library of the TEI, books and the Internet.

As far as the implementation part is concerned, we will try to extend the existing routing protocols. Also, we will try to evaluate our proposal for the extension and try to have some observations by a simulation. Some candidates programming platforms are the ns-2 which is a discrete event simulator, Java which is an object oriented language, C++ simulation library and framework which offers a graphical runtime environment.

6. Literature

For the study, the composition and the completion of this thesis there was the necessity of the access in a variety of literature resources of relative fields such as IPv6, wireless sensor networks, LowPAN, interconnection between IPv6 and wireless sensor networks, 6LowPAN routing protocols, open problems, applications and so on. Thus, we had the opportunity to search for technical reports, journal papers, e-books, presentations of various conferences, proceedings and so on through the library of the TEI of Larisa. Moreover, we also had the opportunity to search for useful on line articles via the search engines in the web (e.g. Google, pdfdatabase and so on). This all inquiry has a consequence, the chance to discover and study all the past work that other researchers have done in the previous years or they continuous doing it till today. This procedure leads to enrich our knowledge and make it deeper than before and to sharpen our interest for our subject of research. Finally, with no doubts, this process helped the writing of our proposal.

In the paragraphs that are followed, are presented more specifically some of the resources (papers, web sites and so on) that were used in order to complete the composition of the dissertation.

Many of the papers that are used for this thesis are presented in conferences such as "*A Quantitative Evaluation of the Simulation Accuracy of Wireless Sensor Networks*", "*Simulation von plattformunabhδngigen TinyOS-Applikationen mit ns-2*" and "*A 6lowpan Implementation for TinyOS 2.0*". These papers have been published in a conference organized by the Department of Computer Science of RWTH Aachen the university, in 2007. Also, "*IPv6 Embedded Systems and Sensor Networks*" is presented in conference in America, in 2009 and the "*WIRELESS SENSOR NETWORKS: STATE OF THE ART AND FUTURE TRENDS*" which has been published in 2nd national conference (Funchal - Madeira), in 2007.

Moreover, many of the papers that are used in order to complete this work are from journal. In this category belong the "*IPv6*" which is most like a technical guide and is published by the school of Electronics and Computer Science in the University of Southampton. Also, the "*Extending IP to Low-Power, Wireless Personal Area Networks*" which is published by the IEEE Computer Society, the "*Interconnection between 802.15.4 Devices and IPv6: Implications and Existing Approaches*" which published in IJCSI - International Journal of Computer Science Issues, on January 2010 and the “A survey on IP-based wireless sensor network solutions” which has been presented in International Journal of Communication Systems.

Another category of papers is the proceedings of conferences. Many, papers belong to this category such as the “*IP Next Generation Overview*” which has been presented in the meeting of IETF in 1994. Also, the paper which has the title “*Securing IPv6 Neighbour and Router Discovery*” has been presented in proceedings of the 1st ACM workshop on Wireless security. Moreover, the paper “*Adapting AODV for IEEE 802.15.4 Mesh Sensor Networks: Theoretical Discussion and Performance Evaluation in a Real Environment*” has been presented in proceedings of the 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06). Finally, the “*IP-enabled Wireless Sensor Networks and their integration into the Internet*” which is the Proceedings of the First International Conference on Integrated Internet Ad hoc and Sensor Networks, May 30-May 31 2006, Nice, France

Last but not least, many resources are coming from the internet. However, the most useful link of a web site is the www.ipv6.com

As mentioned above, this paper focuses on the 6LoWPAn routing. This area is quite new and gains the interest from many experts. Thus, the variety of the pertinent papers is smaller but with very interest articles. Particularly, Gee Keng Ee*, Chee Kyun Ng, Nor Kamariah Noordin and Borhanuddin Mohd. Ali, via their article “*A Review of 6LoWPAN Routing Protocols*”, presents the main routing protocols that exist still today (AODV, LOAD, DYMO-low, HiLow) and compare them. This papers has presented In a similar way the “*Mesh Routing for Low-Power Mobile Ad-HocWireless Sensor Networks Using LOAD*” focuses in the AODV and LOAD routing protocol and moreover the author Vladimir Iliev develops an implementation of LOAD in nesC under TinyOS over RREP and RRER. Finally, a group of experts, E. Kim, D. Kaspar, C. Gomez and C. Bormann have written the latest draft over this subject (expires August 2011) with title “*Problem Statement and Requirements for 6LoWPAN Routing*”. This draft arrives to some security considerations. According to the certain draft in the case of ad-hoc 6LoWPANs where are dynamic in node membership and also in topology, the static security configuration is not adequate. Furthermore, the authors consider that secure localization shelf-organization and time synchronization should be critical to support.

PART B

7. Simulation

In this part of dissertation studied the previous work over the 6LowPAN routing and finally eventuated in the paper of Gee Keng Ee, Chee KYun Ng, Nor Kamariah Noordin and Borhanuddin Mohd. Ali, the "*Path Recovery Mechanism in 6LowPAN Routing*". In this certain paper proposed a step parent node algorithm in a tree topology, to the conventional HiLow. Specifically, according to this step parent node (spn) algorithm, each node knows its MC (MC → the maximum number of children a parent can have) value. In the case where there is a link break after a failure node, each child node of the failure parent node will send a spn request message to the neighbor nodes, as shown in the Figure 32.

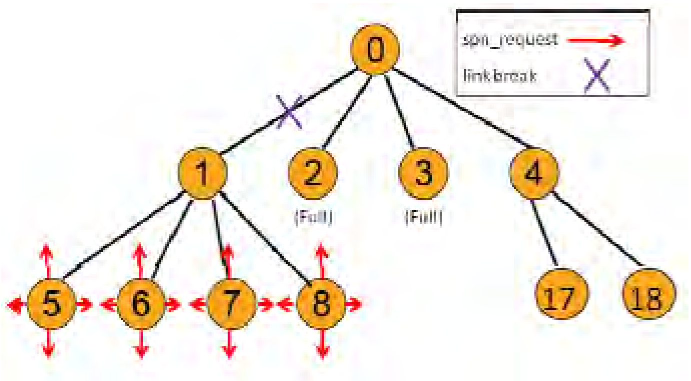
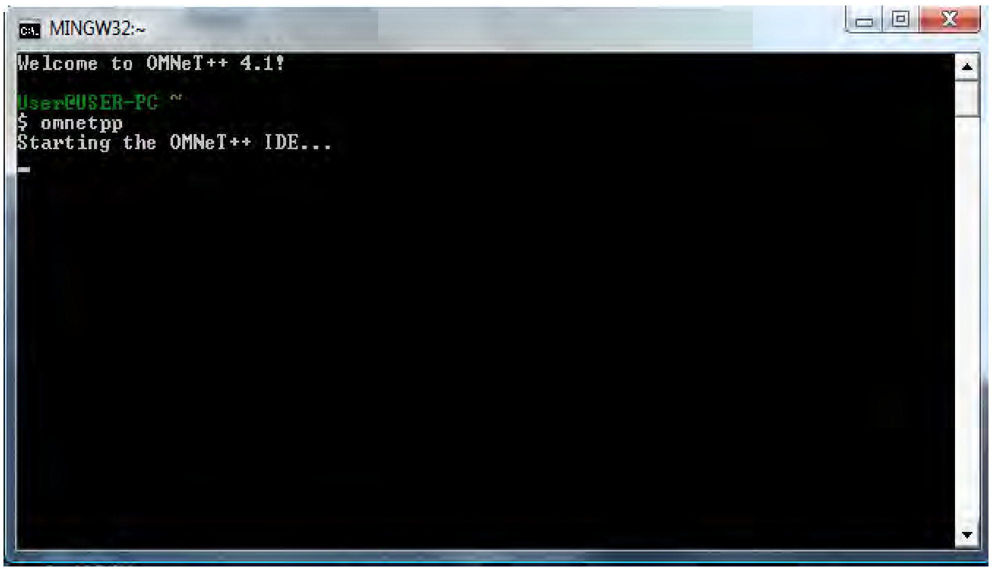


Figure 32: *spn request message*

The neighbour node which accepts the request will define directly if its existing number of child nodes has the appropriate number of nodes (MC) or not. The neighbour node which has the appropriate number of nodes and it is less than the MC number will send a step parent node reply message to the sender of spn request sender. In the following procedure the spn request sender who accepts the spn reply from different neighbors in its personal operating space (POS) will examine the address and the path quality indication (PQI) of each spn reply sender. The node which is interested in finding a step parent node, the spn request sender, exclusively makes a 'bond' with the node which is not the descending node of the spn sender and it has a high PQI. After this connection between the spn request sender node and the neighbour node, the last node will become the step parent node of the child node which has sent the spn request.

Thus in this simulation is tried to develop a different SPN algorithm. In specific, is tried to observe and interpret how the nodes react in different values of MC. After a thorough search I finally eventuate in the platform of OMNeT++ and specifically the edition 4.1, in order to achieve my aim. OMNeT++ is a very popular and useful platform with an open architecture simulation framework (Figures 33, 34, 35). More specifically, OMNeT++ is a discrete time simulator appropriate for wired, fixed, and distributed systems. It is compatible with many operating systems like Windows, Linux and DOS and support parallel execution. Moreover in OMNet++ the simulated objects are presented by simple or composed modules which communicate with each other by messages [47]. The modules consist of .ned file (interface description) and C++ class (behaviour description). Finally, in this program, in the case where there is a project about mobile nodes which collaborate in a wireless way, the user should only develop the appropriate algorithm without focusing on how these points are handled. Thus, there is the opportunity of the change of all the parameters and no new piece of code must be rewrite.



```
Welcome to OMNeT++ 4.1!

User@USER-PC ~
$ omnetpp
Starting the OMNeT++ IDE...
```

Figure 33: *the starting point of OMNeT++*

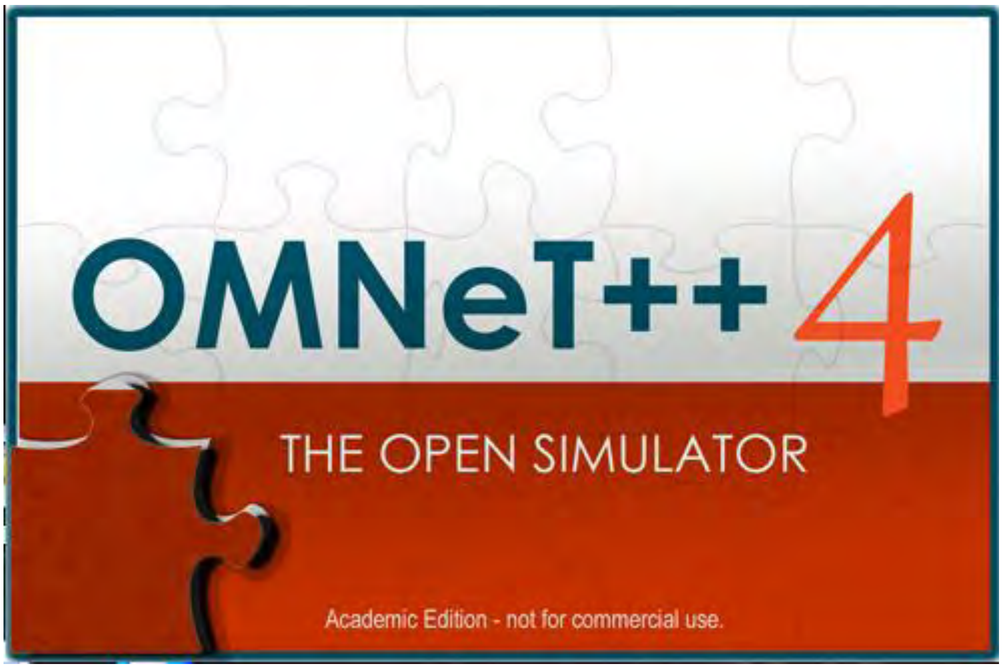
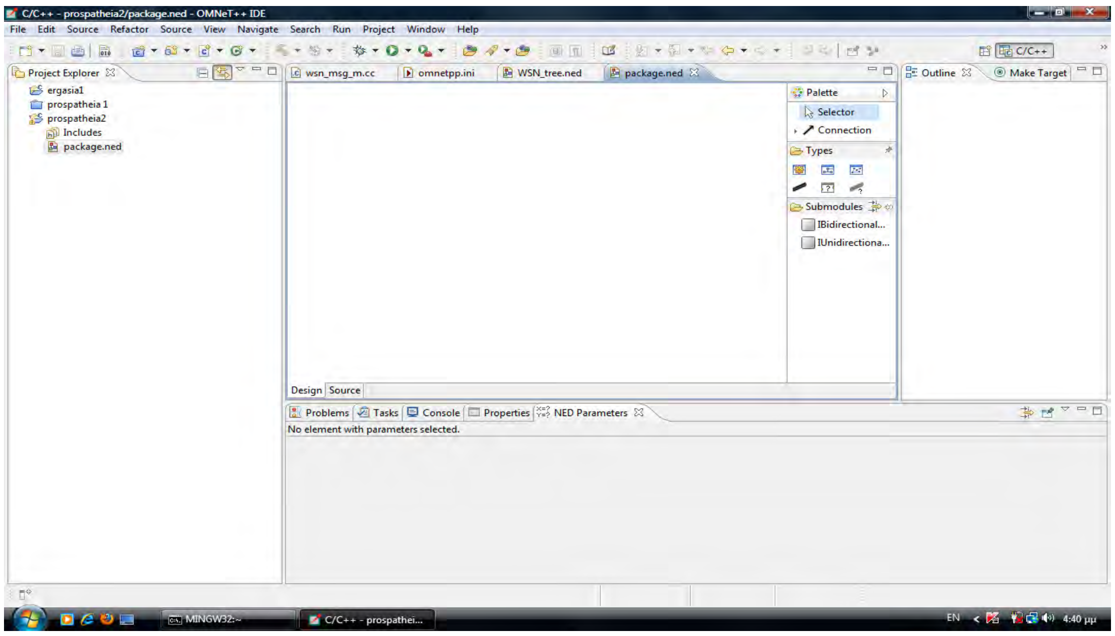


Figure 34: *the edition of OMNeT++ which is used in this simulation*



No element with parameters selected.

Figure 35: *the interface of OMNeT++*

In the specific simulation is created an OMNeT++ project folder with the name of “ergasia1” and in the project folder are created some files (Figure 36).

Initially, in the file package.ned which is already exist is designed the tree topology and then the package.ned file renamed as WSN_tree.ned (Figure 37). Generally, in the .ned file are created the modules, the gates and links in a statically way.

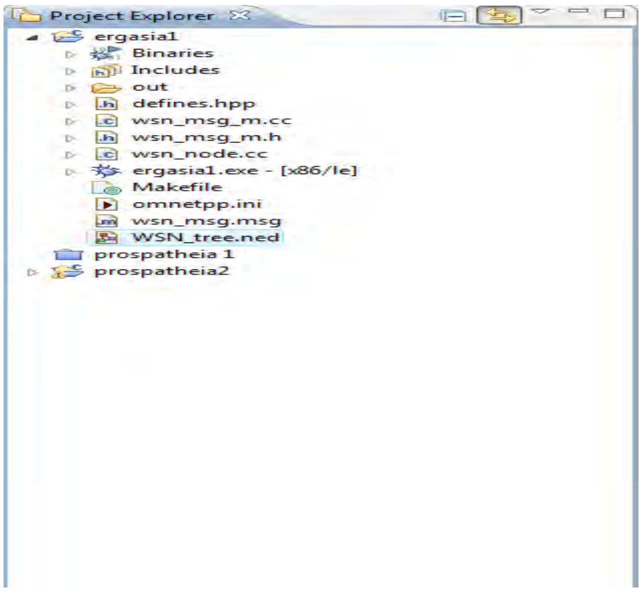


Figure 36: *the files of the folder "ergasia1"*

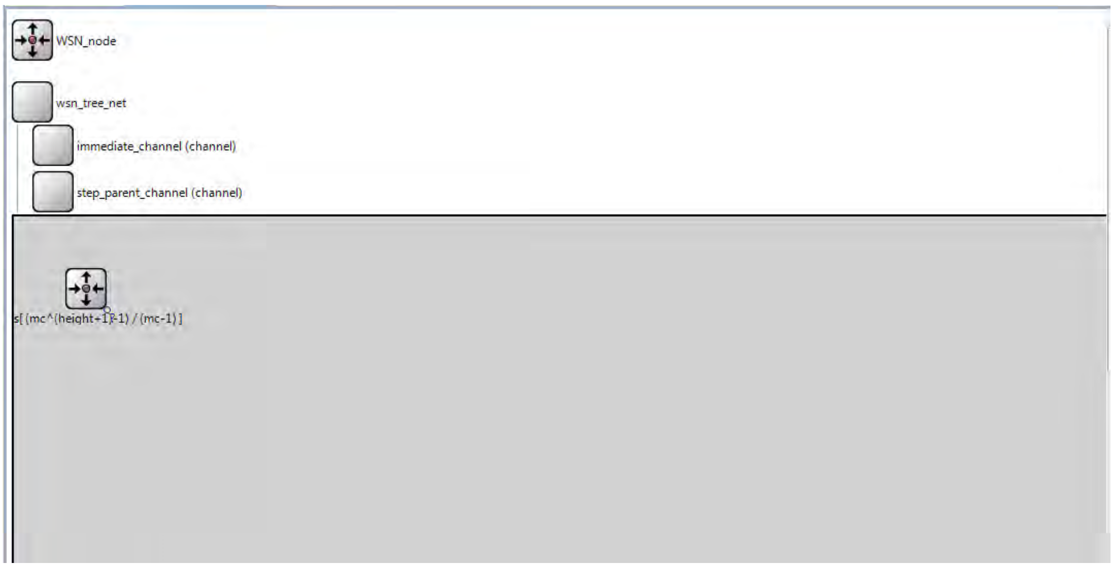

WSN_node
wsn_tree_net
immediate_channel (channel)
step_parent_channel (channel)
s[(mc^(height+1)-1) / (mc-1)]

Figure 37: *the design of tree topology of WSN_tree.ned file*

Above presented the source code of the WSN_tree.ned file:

```
simple WSN_node
{
    parameters:
        double xpos;
        double ypos;
        @display("i=block/routing");
        // property in order to provide a custom layout topology
        // by providing the x,y dimensions for each node in the tree
        @display("p=$xpos,$ypos");
    gates:
        // communication gates for immediate parent-child node'
connections.
        inout gate[];
        // communication gates for step parent-child nodes'
connections.
        inout PAN_gate[];
}

network wsn_tree_net
{
    parameters:
        int height;
        int mc;

//        @display("bgb=970,600;ls=da");
        @display("bgb=1280,800;ls=da");
    types:

        // Two channel types.

        // [A] normal parent - child channels
        channel immediate_channel extends ned.DelayChannel
        {
            delay = 100ms;
            @display("ls=yellow,2");
        }
        // [B] - step parent - child channels.
        channel step_parent_channel extends ned.DelayChannel
        {
            delay = 100ms;
            @display("ls=blue,2");
        }
    submodules:
        nodes[ (mc^(height+1)-1) / (mc-1) ]: WSN_node {
        }
    connections allowunconnected:

        // Channels' initialization

        // [A] normal parent - child channels
        for i=0..sizeof(nodes)-(mc^height)-1, for j=1..mc {
            nodes[i].gate++ <--> immediate_channel <--> nodes[ i * mc
+ j ].gate++
                    if i * mc + j < 17;
        }
            // [B] - step parent - child channels.
            for i=1..sizeof(nodes)-(mc^height)-1, for
j=5..sizeof(nodes)-1 {
```

```
            nodes[i].PAN_gate++ <--> step_parent_channel <--> nodes[
j ].PAN_gate++
                  if ( j < i * mc + 1 || j > i * mc + 4 ) && j < 17;
        }
}
```

Moreover, in this certain file two variables define the height and the max number of children nodes (MC) that can have any node which is no-leaf. These parameters are very important because are hard coded in the piece of code where the node is described.

```
types:

        // Δύο τύποι καναλιών.

        // [A] τα άμεσα κανάλια
        channel immediate_channel extends ned.DelayChannel
        {
            delay = 100ms;
            @display("ls=yellow,2");
        }
        // [B] -τα έμμεσα κανάλια των step parents
        channel step_parent_channel extends ned.DelayChannel
        {
            delay = 100ms;
            @display("ls=blue,2");
        }
```

In specific these lines are referred to two types of channels, the direct and indirect channels.

```
submodules:
        nodes[ (mc^(height+1)-1) / (mc-1) ]: WSN_node;
```

The simulation based on the submodules. The submodules are a table of object of WSN_node class. The number correlates with a full general tree where all the no-leaf nodes have as many children as it has been defined before. Some of these nodes will be not used in order not to be the tree full.

```
connections allowunconnected:
      for i=0..sizeof(nodes)-(mc^height)-1, for j=1..mc {
            nodes[i].gate++ <--> Channel <--> nodes[ i*mc+j ].gate++
                  if i * mc + j <= 17;
        }
```

In these above lines are created the connections of the nodes and in the end of the procedure will created the network tree. The channel of communication – Channel is at 100msec as in all tutorials of OMNeT++.

For running the simulation the .ned file collaborates with the omnetpp.ini by taking elements from it. In the omnetpp.ini file defined the two variables, the MC and network’s height and also defined the position of the nodes. In order to change the values should “clean” the project and rebuild it. The commands that should be used are the followings:

- *make clean; opp_msgc wsn_msg.msg; opp_makemake –f ; make ;*

Below is presented **the code of omnetpp.ini file**. In this case, given as example the height = 2 and the mc = 4.

```
General]
network = wsn_tree_net
**.height = 2
**.mc = 4

**.nodes[0].xpos = 480
**.nodes[0].ypos = 40

**.nodes[1].xpos = 110
**.nodes[1].ypos = 200

**.nodes[2].xpos = 350
**.nodes[2].ypos = 200

**.nodes[3].xpos = 590
**.nodes[3].ypos = 200

**.nodes[4].xpos = 830
**.nodes[4].ypos = 200

# node 1 childs
**.nodes[5].xpos = 30
**.nodes[5].ypos = 400

**.nodes[6].xpos = 90
**.nodes[6].ypos = 400

**.nodes[7].xpos = 150
**.nodes[7].ypos = 400

**.nodes[8].xpos = 210
**.nodes[8].ypos = 400

# node 2 childs
**.nodes[9].xpos = 270
**.nodes[9].ypos = 400

**.nodes[10].xpos = 330
**.nodes[10].ypos = 400

**.nodes[11].xpos = 390
**.nodes[11].ypos = 400

**.nodes[12].xpos = 450
**.nodes[12].ypos = 400
```

```
# node 3 childs
**.nodes[13].xpos = 510
**.nodes[13].ypos = 400

**.nodes[14].xpos = 570
**.nodes[14].ypos = 400

**.nodes[15].xpos = 630
**.nodes[15].ypos = 400

**.nodes[16].xpos = 690
**.nodes[16].ypos = 400

# node 4 childs
**.nodes[17].xpos = 750
**.nodes[17].ypos = 400

**.nodes[18].xpos = 810
**.nodes[18].ypos = 400

**.nodes[19].xpos = 870
**.nodes[19].ypos = 400

**.nodes[20].xpos = 930
**.nodes[20].ypos = 400
```

The source code of wsn_node.cc file:

```
#include"defines.hpp"
#include<omnetpp.h>
#include"wsn_msg_m.h"

class WSN_node : public cSimpleModule
{
public:
	WSN_node();
	~WSN_node();
	static string INIT_MESSAGE;
	static string INFO_MESSAGE;
	static string ACK_MSG;
	static string SPN_REQUEST;
	static string SPN_REPLY;
	static int MAX_CHILD_NODES;
	static int TREE_DEPTH;
	static int MAX_WAIT_ACK_RETRIES;
	static int NODE_INDEX_TO_FAIL;
	static int ACK_MESSAGE_TO_FAIL;
protected:
	virtual Wsn_msg* generateMessage(string, int, int);
	virtual void initialize();
	virtual void handleMessage(cMessage* msg);
	virtual void handleNormalMessage(cMessage* msg);
	virtual void handleTimeoutMessage(cMessage* msg);
	virtual void initiate_network();

private:
	int parent_id;			// the main parent of the
node
	int depth_id;			// the depth of the node
```

```
      bool network_initiaded;     // flag to indicate the network has
been initialized
      bool is_leaf_node;                // flag whether is a leaf or
another node
      double pqi;                     // path quality indicator
      int msg_num;
      bool node_failed;           // variable that indicates taht a
node has failed.
    simtime_t timeout;          // timeout time
    cMessage *timeoutEvent;     // holds pointer to the timeout self-
message
    int timeout_retries;        // timeout retries
    int timeout_factor;         // timeout increasing time factor
    vector<int> step_parents;   // vector to hold the indices
                                      // of the step parent
nodes of a node
    map<int,int> mp_step_childs;

    Wsn_msg* info_msg;      // information message
    Wsn_msg* ack_msg;         // Acknowledgment message
    Wsn_msg* spn_request_msg; // Step parent request message;
    Wsn_msg* spn_reply_msg; // Step parent request message;
};

Define_Module(WSN_node);

// VARIOUS SIMULATION RELATED DEFINITIONS
int WSN_node::MAX_CHILD_NODES = 4;
int WSN_node::TREE_DEPTH = 2;
string WSN_node::INIT_MESSAGE = "INIT_MSG";
string WSN_node::INFO_MESSAGE = "INFO_MSG";
string WSN_node::ACK_MSG      = "ACK_MSG";
string WSN_node::SPN_REQUEST  = "SPN_REQUEST";
string WSN_node::SPN_REPLY    = "SPN_REPLY";

int WSN_node::MAX_WAIT_ACK_RETRIES = 3;
int WSN_node::NODE_INDEX_TO_FAIL   = 3;
int WSN_node::ACK_MESSAGE_TO_FAIL  = 1;

WSN_node::WSN_node()
{
      this->network_initiaded = false;
      this->msg_num = 0;
      this->node_failed = false;

      timeout         = 0.1; // this time is simulation time and not
real time - e.g. seconds etc.
      timeout_retries = 0;
      timeout_factor  = 1;

      timeoutEvent    = NULL;
      info_msg        = NULL;
      ack_msg         = NULL;
      spn_request_msg = NULL;
      spn_reply_msg   = NULL;
}

WSN_node::~WSN_node()
{
      cancelAndDelete(timeoutEvent);
}
```

```
//=================================================================
==========

/*
 * Function that initializes the properties of each node of the
network,
 * always following a tree topology. All members of the node object
are
 * initialized. The function is called within the constructor.
 */
void WSN_node::initialize()
{
      int node_index = getIndex();
      if( node_index == 0 )
      {
            parent_id = -1;
            depth_id = 0;
            // Boot the process scheduling the initial message as a
self-message.
            Wsn_msg* msg = generateMessage(INIT_MESSAGE,0,0);
            scheduleAt(0.0, msg);
      }
      else
      {
            parent_id = node_index / MAX_CHILD_NODES;
            if( node_index % MAX_CHILD_NODES == 0 )
                  parent_id-=1;
            int min_node_id = 0;
            int max_node_id = 0;
            int height = 0;
            while(1)
            {
                  height++;
                  min_node_id = min_node_id * MAX_CHILD_NODES + 1;
                  max_node_id = max_node_id * MAX_CHILD_NODES +
MAX_CHILD_NODES;
                  if( node_index >= min_node_id && node_index <=
max_node_id )
                  {
                        depth_id = height;
                        break;
                  }
            }
      }

      is_leaf_node = ( depth_id == TREE_DEPTH );

      if( is_leaf_node == false )
            this->pqi = dblrand();

      if( is_leaf_node == true )
            timeoutEvent = new cMessage("timeoutEvent");

      //
      // [I] Filling the step parents information. The root node
(index 0) has no step parent, so the vector will contain no nodes.
      //
      if( parent_id != -1 )
      {
            // Below the root node and in the first level of the
tree,
            // the only step parent node is the root.
            if( depth_id == 1 )
                  step_parents.push_back(0);
            // For the rest of the nodes and depending on the tree
depth the vector
            // should contain all of the node indices but not those
of the directly connected
            // ones.
            else
            {
                  int sum = 0;
                  int prod = 1;

                  for( int j=-1; j < depth_id - 2; j++ )
                  {
                        sum += prod;
                        prod *= MAX_CHILD_NODES;
                  }

                  for( int k = sum; k < prod+sum; k++ )
                  {
                        if( k != parent_id )
                              step_parents.push_back(k);
                  }

                  EV << "Node: " << getIndex() << endl;
                  EV << "Step parents: ";

                  vector<int>::iterator it = step_parents.begin();
                  EV << *it;
                  it++;
                  for( ; it != step_parents.end(); it++ )
                  {
                        EV <<  ", " << *it;
                  }
                  EV <<  endl;
            }
      }

      //
      // [II] Filling the child node information for parent nodes.
      //
      if( parent_id != -1 && is_leaf_node == false )
      {
            int sum = 0;
            int prod = 1;
            for( int k = 0; k <= depth_id; k++ )
            {
                  sum += prod;
                  prod *= MAX_CHILD_NODES;
            }

            int gate_count = -1;

            for( int j = sum; j < sum + prod; j++ )
            {
                  if( j < node_index * MAX_CHILD_NODES + 1 ||
                        j > (node_index+1) * MAX_CHILD_NODES )
                  {
                        gate_count++;

      mp_step_childs.insert(pair<int,int>(j,gate_count));
                  }
            }

//          map<int,int>::iterator it;
//          EV << "Node: " << getIndex() << endl;
//          EV << "Step" << "Child gate" << endl;
//
//          for( it = mp_step_childs.begin(); it !=
mp_step_childs.end(); it++ ){
//                EV << it->first << " - " << it->second << endl;
//          }
      }

}

//=====================================================================
==========

Wsn_msg* WSN_node::generateMessage(string content, int destination,
int origin_id )
{
      Wsn_msg* a_msg = new Wsn_msg(content.c_str());
      a_msg->setSource(getIndex());
      a_msg->setDestination(destination);

      if( content == WSN_node::INFO_MESSAGE )
      {
            if( origin_id == getIndex() )
                  msg_num++;
            a_msg->setSeq_num(msg_num);
      }

      if( origin_id == getIndex() || (
                  content == WSN_node::ACK_MSG || content ==
WSN_node::INFO_MESSAGE )  )
      {
            a_msg->setOrigin(origin_id);
      }
      return a_msg;
}

//=====================================================================
==========

void WSN_node::initiate_network()
{
      int gates_connected = this->gateSize("gate");

      if( parent_id == -1 )
      {
            for( int j=0; j<gates_connected; j++ )
            {
                  Wsn_msg* a_msg = generateMessage(INIT_MESSAGE,
                                          getIndex() *
MAX_CHILD_NODES + j+1, -1 );
                  send(a_msg,"gate$o",j);
            }
      }
```

```
        else
        {
                for( int j=1; j<gates_connected; j++ )
                {
                        Wsn_msg* a_msg = generateMessage(INIT_MESSAGE,
                                                         getIndex() *
MAX_CHILD_NODES + j, -1 );
                        send(a_msg,"gate$o",j);
                }
        }
}


//=====================================================================
==========
//
// [X]
//
// Function that handles the two types of events a node can receive.
// Normal node-to-node events and self timeout events.
//
void WSN_node::handleMessage(cMessage* msg)
{
        // [Y] Timeout event handling
        if( msg == timeoutEvent )
        {
                handleTimeoutMessage(msg);
        }
        // [Z] Node-to-node event handling
        else
        {
                handleNormalMessage(msg);
        }

}

//=====================================================================
==========
//
// [Y]
//
// Function that handles all other types of messages but the timeout
ones.
//
void WSN_node::handleNormalMessage(cMessage* msg)
{
        Wsn_msg* the_msg = check_and_cast<Wsn_msg*>(msg);
        string content = the_msg->getName();
        //
        // [A] INITIALIZATION MESSAGE
        //
        if( content == INIT_MESSAGE && network_initiaded == false )
        {
                network_initiaded = true;
                if( parent_id != -1 )
                        bubble( "INIT_NET_MSG" );

                delete msg;

                if( is_leaf_node == false )
                        initiate_network();
```

```
            // a leaf node will send a INFORMATION MESSAGE and also
schedule the timeout timer.
            else
            {
                  info_msg =
generateMessage(INFO_MESSAGE,parent_id,getIndex());
                  send(info_msg,"gate$o",0);
                  // this event is independent from the event queue
that are send between the nodes
                  scheduleAt(simTime()+timeout, timeoutEvent);
            }
      }
      //
      // [B] INFORMATION MESSAGE
      //
      else if( content == INFO_MESSAGE )
      {
            if( the_msg->getDestination() == getIndex() )
            {
                  if( parent_id != -1 && is_leaf_node == false )
                  {
                        // FORWARDING THE INFORMATION MESSAGE -
MIDDLE NODE - TO ROOT
                        info_msg =
generateMessage(INFO_MESSAGE,parent_id,the_msg->getOrigin());
                        info_msg->setSeq_num(the_msg->getSeq_num());
                        delete msg;

                        send(info_msg,"gate$o",0);
                  }
                  else if( parent_id == -1 )
                  {
                        ack_msg = generateMessage(ACK_MSG,the_msg-
>getSource(),
                                    the_msg->getOrigin());
                        ack_msg->setSeq_num(the_msg->getSeq_num());
                        delete msg;
                        send(ack_msg,"gate$o",ack_msg-
>getDestination()-1);
                  }
            }
      }
      //
      // [C] ACKNOWLEDGEMENT MESSAGE
      //
      else if( content == ACK_MSG )
      {
            // Leaf node received its ACK message.
            if( the_msg->getOrigin() == getIndex() )
            {
                  EV << "Node: " << the_msg->getOrigin()
                              << " - Sequence num: " << the_msg-
>getSeq_num() << endl;
                  delete msg;
                  bubble("ACK_MSG - RECEIVED - NEW INFO MESSAGE");
                  // RESTART INFO MESSAGE TRANSMISSION
                  info_msg =
generateMessage(INFO_MESSAGE,parent_id,getIndex());
                  cancelEvent(timeoutEvent);
                  // reset timeout retries and wait time
                  timeout = 0.1;
```

```
                timeout_retries = 0;
                timeout_factor = 1;
                send(info_msg,"gate$o",0);
                scheduleAt(simTime()+timeout, timeoutEvent);
          }
          else
          {
                // middle node ACK message handling
                if( parent_id != -1 )
                {
                      if( the_msg->getOrigin() != getIndex() )
                      {
                            // simulate link failure

                            // In first ACK message sent by the
root, and if the node index is 3
                            // and the ACK message sequence is 1
the failure flag is raised to true
                            // and the node does not forward any
message. For any subsequent messages
                            // that will arrive to the node, the
failure flag will be true already...
                            if( node_failed == false )
                            {
                                  if( NODE_INDEX_TO_FAIL ==
getIndex() &&
                                              ACK_MESSAGE_TO_FAIL
== the_msg->getSeq_num() )
                                  {
                                        bubble("NODE FAILURE!!!");
                                        node_failed = true;
                                        delete msg;
                                        return;
                                  }
                            }
                            else
                            {
                                  // ...and so no further message
action is taken.
                                  delete msg;
                                  return;
                            }

                            // If node has not failed forwards the
ACK message to the proper child
                            // leaf node.
                            int msg_origin = the_msg->getOrigin();
                            int tmp = 0;
                            int port = 0;
                            for(int i=1; i<=MAX_CHILD_NODES; i++)
                            {
                                  tmp = MAX_CHILD_NODES *
getIndex() + i;

                                  if( tmp == msg_origin )
                                  {
                                        port = i;
                                        break;
                                  }
                            }
```

```
                                        ack_msg =
generateMessage(ACK_MSG,the_msg->getDestination(),
                                                the_msg->getOrigin());
                                        ack_msg->setSeq_num(the_msg-
>getSeq_num());
                                        delete msg;
                                        send(ack_msg,"gate$o",port);
                                }
                        }
                }
        }
        //
        // [D] STEP PARENT REQUEST MESSAGE
        //
        else if( content == SPN_REQUEST )
        {
                bubble("STEP PARENT REQUEST MESSAGE");
                // 1. Check if the maximum number of child nodes are
connected
                int child_nodes_connected = this->gateSize("gate");

                // subtract one because of the parent node connection
                child_nodes_connected--;

                EV << "Child nodes connected: " <<  child_nodes_connected
<< endl;
                // If the node is connected already to the maximum number
of child nodes
                // take no action than display an information message.
                if( child_nodes_connected == MAX_CHILD_NODES )
                {
                        bubble("NO CONNECTIONS AVAILABLE");
                        delete msg;
                        return;
                }
                else
                {
                        // in any other case a unicast message must be sent
to each SPN reply message
                        // received.
                        bubble("CONNECTIONS AVAILABLE");
                        spn_reply_msg = generateMessage(SPN_REPLY,the_msg-
>getSource(),-1);

                        // set PQI ( = Path Quality Indicator )
                        spn_reply_msg->setPqi(this->pqi);
                        spn_reply_msg->setDisplayString("i=msg/resp_s");


                        map<int,int>::iterator it;
                        int port_to_reply = -1;

                        for( it = mp_step_childs.begin(); it !=
mp_step_childs.end(); it++ ){
                                if( (*it).first == the_msg->getSource() )
                                {
                                        port_to_reply = (*it).second;
                                        break;
                                }
                        }
```

```
                  EV << "Child to reply: " << the_msg->getSource() <<
endl;
                  EV << "Gate to reply:  " << port_to_reply << endl;
                  // delete message avoid memory leaks
                  delete msg;
                  send( spn_reply_msg, "PAN_gate$o",port_to_reply );
            }
      }
      //
      // [E] STEP PARENT REPLY MESSAGE - In this simple case only
leaf nodes will receive
      //                                   such messages.
      //
      else if( content == SPN_REPLY )
      {
            bubble("PARENT NODE AVAILABLE");
            EV << "Step parent: "<< the_msg->getSource() << " - PQI:
" << the_msg->getPqi() << endl;
            //TODO: In order to select one of multiple parent nodes
to connect to with respect
            //      to PQI another timer must be used that actually
will wait for more parent
            //      nodes to reply. In case mor than two reply the
best PQI is the one that
            //      the child node must be connected to.

            cModule* spn_sender_module = the_msg->getSenderModule();
            cGate* spn_sender_gate = the_msg->getSenderGate();
            spn_sender_gate->disconnect();


            cModule* child_arrival_module = the_msg-
>getArrivalModule();
            cGate* child_arrival_gate = the_msg->getArrivalGate();
            child_arrival_gate->disconnect();

            // Simple case connect to the parent that replied

            int gate_type_connected = spn_sender_module-
>gateSize("gate");
            EV << gate_type_connected << endl;
            spn_sender_module-
>setGateSize("gate",gate_type_connected+1);

            cChannelType *channelType =
cChannelType::get("wsn_tree_net.immediate_channel");
            cChannel *channel_from_parent = channelType-
>create("channel");
            cChannel *channel_to_parent = channelType-
>create("channel");

            child_arrival_module->gate("gate$o",0)->disconnect();
            child_arrival_module->gate("gate$i",0)->disconnect();


      //=============================================================
========
            child_arrival_module->gate("gate$i",0)->connectTo(
                        spn_sender_module-
>gate("gate$o",gate_type_connected), channel_from_parent );
            child_arrival_module->gate("gate$o",0)->connectTo(
                                                    spn_sender_module-
>gate("gate$i",gate_type_connected), channel_to_parent );

            this->parent_id = spn_sender_module->getIndex();
            this->setIndex( spn_sender_module->getIndex() *
MAX_CHILD_NODES + gate_type_connected, 0 );
            info_msg =
generateMessage(INFO_MESSAGE,parent_id,getIndex());
            delete msg;
            timeout_retries = 0;
            send(info_msg,"gate$o",0);
            // this event is independent from the event queue that
are send between the nodes
            scheduleAt(simTime()+timeout, timeoutEvent);
      }
}

//=====================================================================
==========

//
// [Y] Function that handles the timeout events of self messages.
//
void WSN_node::handleTimeoutMessage(cMessage* msg)
{
      EV << "Timeout expired\n";
      // If the timeout retries are equal to the max then the Step-
Parent-Request
      // message must be broadcasted.
      if( timeout_retries == MAX_WAIT_ACK_RETRIES )
      {
            EV << "Parent node not functional - Broadcast step parent
message\n";

            // decrease the message sequence number since the node
has never received an ACK
            // message that the root has received the information. So
the message must be resent.
            // But before it is resent the node must be reconnected
with another parent node.
            this->msg_num--;

            // 1. Find out how many personal area parent nodes are
connected.
            int PAN_parents_connected = this->gateSize("PAN_gate");
            EV << "Step parents connected: " << PAN_parents_connected
<< endl;

            // For each step parent node sent a SPN_REQUEST message.
            for( int k = 0; k < PAN_parents_connected; k++ )
            {
                  Wsn_msg* a_msg =
generateMessage(SPN_REQUEST,step_parents[k],-1);
                  a_msg->setDisplayString("i=msg/bcast_s");
                  send(a_msg,"PAN_gate$o",k);
            }
      }
      else
      {
            // re-schedule a timeout event again
            EV << "Wait for acknowledgment, re-schedule timeout event 
timer\n";
            timeout_retries++;
            //timeout_factor++;
            timeout = timeout_factor * timeout;
            scheduleAt(simTime()+timeout, timeoutEvent);
      }
}
```

This file describes the node and is the "heart" of the simulation. The five variables that are presented below are static and are used for the types of messages which transferred between the nodes.

```
static string INIT_MESSAGE;
static string INFO_MESSAGE;
static string ACK_MSG;
static string SPN_REQUEST;
static string SPN_REPLY;
```

The two following variables correlated with max number of child node and the depth of the tree, as mentioned before.

```
static int MAX_CHILD_NODES;
static int TREE_DEPTH;
```

The following lines of the code present the attitude of the node. Specifically, the *protected* section has the functions of the node's attitude and the private section has members which define how a node should react in any type of message. Moreover, in the function ***virtual void initialize ();*** initialized all the members of the object which are necessary for the routing of the messages. Also in this function a self message is developed for the root node in order to start the procedure of sending and accepting the messages. Furthermore in the function ***virtual void initiate_network();*** a message is sent to the nodes in order to inform that the network is ready to begin.

```
class WSN_node : public cSimpleModule
{
public:
      WSN_node();
      static string INIT_MESSAGE;
      static string INFO_MESSAGE;
      static string ACK_MSG;
      static string SPN_REQUEST;
      static string SPN_REPLY;
      static int MAX_CHILD_NODES;
      static int TREE_DEPTH;

protected:
      virtual Wsn_msg* generateMessage(string, int, int);
      virtual void initialize();
      virtual void handleMessage(cMessage* msg);
      virtual void handleNormalMessage(cMessage* msg);
      virtual void handleTimeoutMessage(cMessage* msg);
      virtual void initiate_network();

private:
      int parent_id;                  // the main parent of the node
      int depth_id;                   // the depth of the node
      bool network_initiaded; // flag to indicate the network has
been initialized
      bool is_leaf_node;              // flag whether is a leaf or
another node
      int pqi;                // path quality indicator
      int msg_num;
      bool node_failed;       // variable that indicates taht a node
has failed.
    simtime_t timeout;        // timeout time
    cMessage *timeoutEvent;   // holds pointer to the timeout self-
message
    int timeout_retries;      // timeout retries
    int timeout_factor;       // timeout increasing time factor
    vector<int> step_parents; // vector to hold the indices
                                          // of the step parent nodes
of a node
    map<int,int> mp_step_childs;

    Wsn_msg* info_msg;       // information message
    Wsn_msg* ack_msg;          // Acknowledgment message
    Wsn_msg* spn_request_msg; // Step parent request message;
    Wsn_msg* spn_reply_msg; // Step parent request message;
};
```

The source code of wsn_msg_m.cc file:

```
//
// Generated file, do not edit! Created by opp_msgc 4.1 from
wsn_msg.msg.
//

// Disable warnings about unused variables, empty switch stmts, etc:
#ifdef _MSC_VER
#  pragma warning(disable:4101)
#  pragma warning(disable:4065)
#endif

#include <iostream>
#include <sstream>
#include "wsn_msg_m.h"

// Template rule which fires if a struct or class doesn't have
operator<<
template<typename T>
std::ostream& operator<<(std::ostream& out,const T&) {return out;}

// Another default rule (prevents compiler from choosing base class'
doPacking())
template<typename T>
void doPacking(cCommBuffer *, T& t) {
    throw cRuntimeError("Parsim error: no doPacking() function for
type %s or its base class (check .msg and _m.cc/h
files!)",opp_typename(typeid(t)));
}

template<typename T>
void doUnpacking(cCommBuffer *, T& t) {
    throw cRuntimeError("Parsim error: no doUnpacking() function for
type %s or its base class (check .msg and _m.cc/h
files!)",opp_typename(typeid(t)));
}




Register_Class(Wsn_msg);

Wsn_msg::Wsn_msg(const char *name, int kind) : cMessage(name,kind)
{
    this->source_var = 0;
    this->destination_var = 0;
    this->seq_num_var = 0;
    this->origin_var = 0;
    this->pqi_var = 0;
    this->displayString_var = "i=msg/floppy_s,kind";
}

Wsn_msg::Wsn_msg(const Wsn_msg& other) : cMessage()
{
    setName(other.getName());
    operator=(other);
}

Wsn_msg::~Wsn_msg()
{
}

Wsn_msg& Wsn_msg::operator=(const Wsn_msg& other)
{
    if (this==&other) return *this;
    cMessage::operator=(other);
    this->source_var = other.source_var;
    this->destination_var = other.destination_var;
    this->seq_num_var = other.seq_num_var;
    this->origin_var = other.origin_var;
    this->pqi_var = other.pqi_var;
    this->displayString_var = other.displayString_var;
    return *this;
}

void Wsn_msg::parsimPack(cCommBuffer *b)
{
    cMessage::parsimPack(b);
    doPacking(b,this->source_var);
    doPacking(b,this->destination_var);
    doPacking(b,this->seq_num_var);
    doPacking(b,this->origin_var);
    doPacking(b,this->pqi_var);
    doPacking(b,this->displayString_var);
}
```

```
void Wsn_msg::parsimUnpack(cCommBuffer *b)
{
    cMessage::parsimUnpack(b);
    doUnpacking(b,this->source_var);
    doUnpacking(b,this->destination_var);
    doUnpacking(b,this->seq_num_var);
    doUnpacking(b,this->origin_var);
    doUnpacking(b,this->pqi_var);
    doUnpacking(b,this->displayString_var);
}

int Wsn_msg::getSource() const
{
    return source_var;
}

void Wsn_msg::setSource(int source_var)
{
    this->source_var = source_var;
}

int Wsn_msg::getDestination() const
{
    return destination_var;
}

void Wsn_msg::setDestination(int destination_var)
{
    this->destination_var = destination_var;
}

int Wsn_msg::getSeq_num() const
{
    return seq_num_var;
}

void Wsn_msg::setSeq_num(int seq_num_var)
{
    this->seq_num_var = seq_num_var;
}

int Wsn_msg::getOrigin() const
{
    return origin_var;
}

void Wsn_msg::setOrigin(int origin_var)
{
    this->origin_var = origin_var;
}

double Wsn_msg::getPqi() const
{
    return pqi_var;
}

void Wsn_msg::setPqi(double pqi_var)
{
    this->pqi_var = pqi_var;
}
```

```
const char * Wsn_msg::getDisplayString() const
{
    return displayString_var.c_str();
}

void Wsn_msg::setDisplayString(const char * displayString_var)
{
    this->displayString_var = displayString_var;
}

class Wsn_msgDescriptor : public cClassDescriptor
{
  public:
    Wsn_msgDescriptor();
    virtual ~Wsn_msgDescriptor();

    virtual bool doesSupport(cObject *obj) const;
    virtual const char *getProperty(const char *propertyname) const;
    virtual int getFieldCount(void *object) const;
    virtual const char *getFieldName(void *object, int field) const;
    virtual int findField(void *object, const char *fieldName) const;
    virtual unsigned int getFieldTypeFlags(void *object, int field)
const;
    virtual const char *getFieldTypeString(void *object, int field)
const;
    virtual const char *getFieldProperty(void *object, int field,
const char *propertyname) const;
    virtual int getArraySize(void *object, int field) const;

    virtual std::string getFieldAsString(void *object, int field, int
i) const;
    virtual bool setFieldAsString(void *object, int field, int i,
const char *value) const;

    virtual const char *getFieldStructName(void *object, int field)
const;
    virtual void *getFieldStructPointer(void *object, int field, int
i) const;
};

Register_ClassDescriptor(Wsn_msgDescriptor);

Wsn_msgDescriptor::Wsn_msgDescriptor() : cClassDescriptor("Wsn_msg",
"cMessage")
{
}

Wsn_msgDescriptor::~Wsn_msgDescriptor()
{
}

bool Wsn_msgDescriptor::doesSupport(cObject *obj) const
{
    return dynamic_cast<Wsn_msg *>(obj)!=NULL;
}

const char *Wsn_msgDescriptor::getProperty(const char *propertyname)
const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    return basedesc ? basedesc->getProperty(propertyname) : NULL;
```

```
}

int Wsn_msgDescriptor::getFieldCount(void *object) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    return basedesc ? 6+basedesc->getFieldCount(object) : 6;
}

unsigned int Wsn_msgDescriptor::getFieldTypeFlags(void *object, int
field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldTypeFlags(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static unsigned int fieldTypeFlags[] = {
        FD_ISEDITABLE,
        FD_ISEDITABLE,
        FD_ISEDITABLE,
        FD_ISEDITABLE,
        FD_ISEDITABLE,
        FD_ISEDITABLE,
    };
    return (field>=0 && field<6) ? fieldTypeFlags[field] : 0;
}

const char *Wsn_msgDescriptor::getFieldName(void *object, int field)
const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldName(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static const char *fieldNames[] = {
        "source",
        "destination",
        "seq_num",
        "origin",
        "pqi",
        "displayString",
    };
    return (field>=0 && field<6) ? fieldNames[field] : NULL;
}

int Wsn_msgDescriptor::findField(void *object, const char *fieldName)
const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    int base = basedesc ? basedesc->getFieldCount(object) : 0;
    if (fieldName[0]=='s' && strcmp(fieldName, "source")==0) return
base+0;
    if (fieldName[0]=='d' && strcmp(fieldName, "destination")==0)
return base+1;
    if (fieldName[0]=='s' && strcmp(fieldName, "seq_num")==0) return
base+2;
    if (fieldName[0]=='o' && strcmp(fieldName, "origin")==0) return
base+3;
```

```
    if (fieldName[0]=='p' && strcmp(fieldName, "pqi")==0) return
base+4;
    if (fieldName[0]=='d' && strcmp(fieldName, "displayString")==0)
return base+5;
    return basedesc ? basedesc->findField(object, fieldName) : -1;
}

const char *Wsn_msgDescriptor::getFieldTypeString(void *object, int
field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldTypeString(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static const char *fieldTypeStrings[] = {
        "int",
        "int",
        "int",
        "int",
        "double",
        "string",
    };
    return (field>=0 && field<6) ? fieldTypeStrings[field] : NULL;
}

const char *Wsn_msgDescriptor::getFieldProperty(void *object, int
field, const char *propertyname) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldProperty(object, field,
propertyname);
        field -= basedesc->getFieldCount(object);
    }
    switch (field) {
        default: return NULL;
    }
}

int Wsn_msgDescriptor::getArraySize(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getArraySize(object, field);
        field -= basedesc->getFieldCount(object);
    }
    Wsn_msg *pp = (Wsn_msg *)object; (void)pp;
    switch (field) {
        default: return 0;
    }
}

std::string Wsn_msgDescriptor::getFieldAsString(void *object, int
field, int i) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldAsString(object,field,i);
        field -= basedesc->getFieldCount(object);
    }
    Wsn_msg *pp = (Wsn_msg *)object; (void)pp;
    switch (field) {
        case 0: return long2string(pp->getSource());
        case 1: return long2string(pp->getDestination());
        case 2: return long2string(pp->getSeq_num());
        case 3: return long2string(pp->getOrigin());
        case 4: return double2string(pp->getPqi());
        case 5: return oppstring2string(pp->getDisplayString());
        default: return "";
    }
}

bool Wsn_msgDescriptor::setFieldAsString(void *object, int field, int
i, const char *value) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->setFieldAsString(object,field,i,value);
        field -= basedesc->getFieldCount(object);
    }
    Wsn_msg *pp = (Wsn_msg *)object; (void)pp;
    switch (field) {
        case 0: pp->setSource(string2long(value)); return true;
        case 1: pp->setDestination(string2long(value)); return true;
        case 2: pp->setSeq_num(string2long(value)); return true;
        case 3: pp->setOrigin(string2long(value)); return true;
        case 4: pp->setPqi(string2double(value)); return true;
        case 5: pp->setDisplayString((value)); return true;
        default: return false;
    }
}

const char *Wsn_msgDescriptor::getFieldStructName(void *object, int
field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldStructName(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static const char *fieldStructNames[] = {
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
        NULL,
    };
    return (field>=0 && field<6) ? fieldStructNames[field] : NULL;
}

void *Wsn_msgDescriptor::getFieldStructPointer(void *object, int
field, int i) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldStructPointer(object, field, i);
        field -= basedesc->getFieldCount(object);
    }
    Wsn_msg *pp = (Wsn_msg *)object; (void)pp;
    switch (field) {
        default: return NULL;
    }
}
```

The source code of wsn_msg_m.h file:

```
//
// Generated file, do not edit! Created by opp_msgc 4.1 from
wsn_msg.msg.
//

#ifndef _WSN_MSG_M_H_
#define _WSN_MSG_M_H_

#include <omnetpp.h>

// opp_msgc version check
#define MSGC_VERSION 0x0401
#if (MSGC_VERSION!=OMNETPP_VERSION)
#    error Version mismatch! Probably this file was generated by an
earlier version of opp_msgc: 'make clean' should help.
#endif



/**
 * Class generated from <tt>wsn_msg.msg</tt> by opp_msgc.
 * <pre>
 * message Wsn_msg {
 *     int source;
 *     int destination;
 *     int seq_num = 0;
 *     int origin;
 *     double pqi;
 *     string displayString = "i=msg/floppy_s,kind";
 * }
 * </pre>
 */
class Wsn_msg : public ::cMessage
{
  protected:
    int source_var;
    int destination_var;
    int seq_num_var;
    int origin_var;
    double pqi_var;
    opp_string displayString_var;
    // protected and unimplemented operator==(), to prevent
accidental usage
    bool operator==(const Wsn_msg&);

  public:
    Wsn_msg(const char *name=NULL, int kind=0);
    Wsn_msg(const Wsn_msg& other);
    virtual ~Wsn_msg();
    Wsn_msg& operator=(const Wsn_msg& other);
    virtual Wsn_msg *dup() const {return new Wsn_msg(*this);}
    virtual void parsimPack(cCommBuffer *b);
    virtual void parsimUnpack(cCommBuffer *b);

    // field getter/setter methods
    virtual int getSource() const;
    virtual void setSource(int source_var);
    virtual int getDestination() const;
    virtual void setDestination(int destination_var);
    virtual int getSeq_num() const;
    virtual void setSeq_num(int seq_num_var);
    virtual int getOrigin() const;
    virtual void setOrigin(int origin_var);
    virtual double getPqi() const;
    virtual void setPqi(double pqi_var);
    virtual const char * getDisplayString() const;
    virtual void setDisplayString(const char * displayString_var);
};

inline void doPacking(cCommBuffer *b, Wsn_msg& obj)
{obj.parsimPack(b);}
inline void doUnpacking(cCommBuffer *b, Wsn_msg& obj)
{obj.parsimUnpack(b);}


#endif // _WSN_MSG_M_H_
```

The source code of Makefile file:

```
#
# OMNeT++/OMNEST Makefile for ergasia1
#
# This file was generated with the command:
#  opp_makemake -f --deep -O out
#

# Name of target to be created (-o option)
TARGET = ergasia1$(EXE_SUFFIX)

# User interface (uncomment one) (-u option)
USERIF_LIBS = $(ALL_ENV_LIBS) # that is, $(TKENV_LIBS) $(CMDENV_LIBS)
#USERIF_LIBS = $(CMDENV_LIBS)
#USERIF_LIBS = $(TKENV_LIBS)

# C++ include paths (with -I)
INCLUDE_PATH = -I.

# Additional object and library files to link with
EXTRA_OBJS =
# Additional libraries (-L, -l options)
LIBS =

# Output directory
PROJECT_OUTPUT_DIR = out
PROJECTRELATIVE_PATH =
O = $(PROJECT_OUTPUT_DIR)/$(CONFIGNAME)/$(PROJECTRELATIVE_PATH)

# Object files for local .cc and .msg files
OBJS = $O/wsn_node.o $O/wsn_msg_m.o

# Message files
MSGFILES = \
    wsn_msg.msg

#----------------------------------------------------------------------
----------

# Pull in OMNeT++ configuration (Makefile.inc or configuser.vc)

ifneq ("$(OMNETPP_CONFIGFILE)","")
CONFIGFILE = $(OMNETPP_CONFIGFILE)
else
ifneq ("$(OMNETPP_ROOT)","")
CONFIGFILE = $(OMNETPP_ROOT)/Makefile.inc
else
CONFIGFILE = $(shell opp_configfilepath)
endif
endif

ifeq ("$(wildcard $(CONFIGFILE))","")
$(error Config file '$(CONFIGFILE)' does not exist -- add the OMNeT++
bin directory to the path so that opp_configfilepath can be found, or
set the OMNETPP_CONFIGFILE variable to point to Makefile.inc)
endif

include $(CONFIGFILE)

# Simulation kernel and user interface libraries
OMNETPP_LIB_SUBDIR = $(OMNETPP_LIB_DIR)/$(TOOLCHAIN_NAME)
OMNETPP_LIBS = -L"$(OMNETPP_LIB_SUBDIR)" -L"$(OMNETPP_LIB_DIR)"
$(USERIF_LIBS) $(KERNEL_LIBS) $(SYS_LIBS)

COPTS = $(CFLAGS)  $(INCLUDE_PATH) -I$(OMNETPP_INCL_DIR)
MSGCOPTS = $(INCLUDE_PATH)

#----------------------------------------------------------------------
----------
# User-supplied makefile fragment(s)
# >>>
# <<<
#----------------------------------------------------------------------
----------

# Main target
all: $(TARGET)

$(TARGET) : $O/$(TARGET)
      $(LN) $O/$(TARGET) .

$O/$(TARGET): $(OBJS)  $(wildcard $(EXTRA_OBJS)) Makefile
```

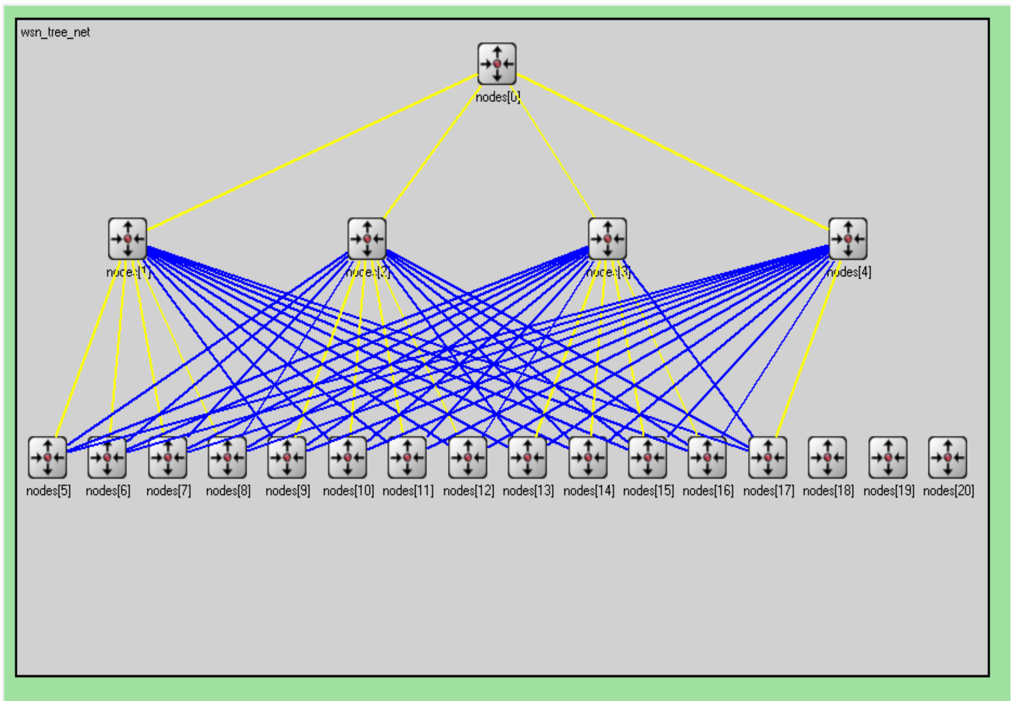
```
	@$(MKPATH) $O
	$(CXX) $(LDFLAGS) -o $O/$(TARGET)  $(OBJS) $(EXTRA_OBJS)
$(WHOLE_ARCHIVE_ON) $(LIBS) $(WHOLE_ARCHIVE_OFF) $(OMNETPP_LIBS)

.PHONY:

.SUFFIXES: .cc

$O/%.o: %.cc
	@$(MKPATH) $(dir $@)
	$(CXX) -c $(COPTS) -o $@ $<

%_m.cc %_m.h: %.msg
	$(MSGC) -s _m.cc $(MSGCOPTS) $?

msgheaders: $(MSGFILES:.msg=_m.h)

clean:
	-rm -rf $O
	-rm -f ergasia1 ergasia1.exe libergasia1.so libergasia1.a
libergasia1.dll libergasia1.dylib
	-rm -f ./*_m.cc ./*_m.h

cleanall: clean
	-rm -rf $(PROJECT_OUTPUT_DIR)

depend:
	$(MAKEDEPEND) $(INCLUDE_PATH) -f Makefile -P\$$O/ --
$(MSG_CC_FILES)  ./*.cc

# DO NOT DELETE THIS LINE -- make depend depends on it.
$O/wsn_node.o: wsn_node.cc
$O/wsn_msg_m.o: wsn_msg_m.cc \
	wsn_msg_m.h
```

The source code of wsn_msg.msg file:

```
// 1. Source node id
// 2. Destination node id
// 3. Sequence number of message sent by the same node
// 4. The origin node of the message. This information is used for
the acknowledgement
//    type of messages so as to be forwarded to the correct origin
node.
// 5. PQI (=Path Quality Indicator).
// 6. Message's attribute which has the ability to check the kind of
// the bitmap will appeared in the simulation in order differentiate
// the types of the messages
message Wsn_msg {
    int source;
    int destination;
    int seq_num = 0;
    int origin;
    int pqi;
    string displayString = "i=msg/floppy_s,kind";
}
```

This file describes the messages that are sent between the nodes. It contains important information like source and destination address, the origin of the message and the number of the messages. In order to be created this class should be given the command of >***opp_msgc wsn_msg.msg.***

In general the simulation process is described below as shown in the Figures 38, 39, 40, 41, 42 and 43. In this case where appeared below are used as value of height = 2, MC = 4. In this certain example the difference from the paper is that the node 4 has only one child node. As mentioned before if anyone wants to change the values and *x*pos, *y*pos should re-assign them in omnetpp.ini file.



<u>Figure 38:</u> *an example of simulation's topology*

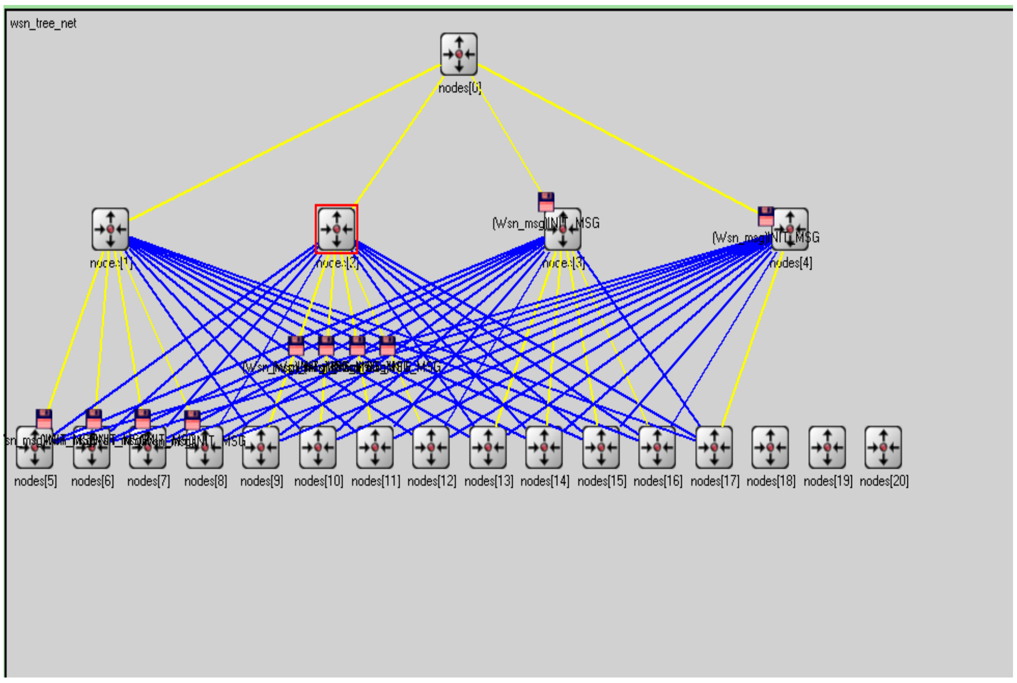

wsn_tree_net
nodes[0]
nodes[1]
nodes[2]
nodes[3]
nodes[4]
(Wsn_msg)INIT_MSG
nodes[5] nodes[6] nodes[7] nodes[8] nodes[9] nodes[10] nodes[11] nodes[12] nodes[13] nodes[14] nodes[15] nodes[16] nodes[17] nodes[18] nodes[19] nodes[20]

Figure 39: *a shot of simulation where the node sends a message*

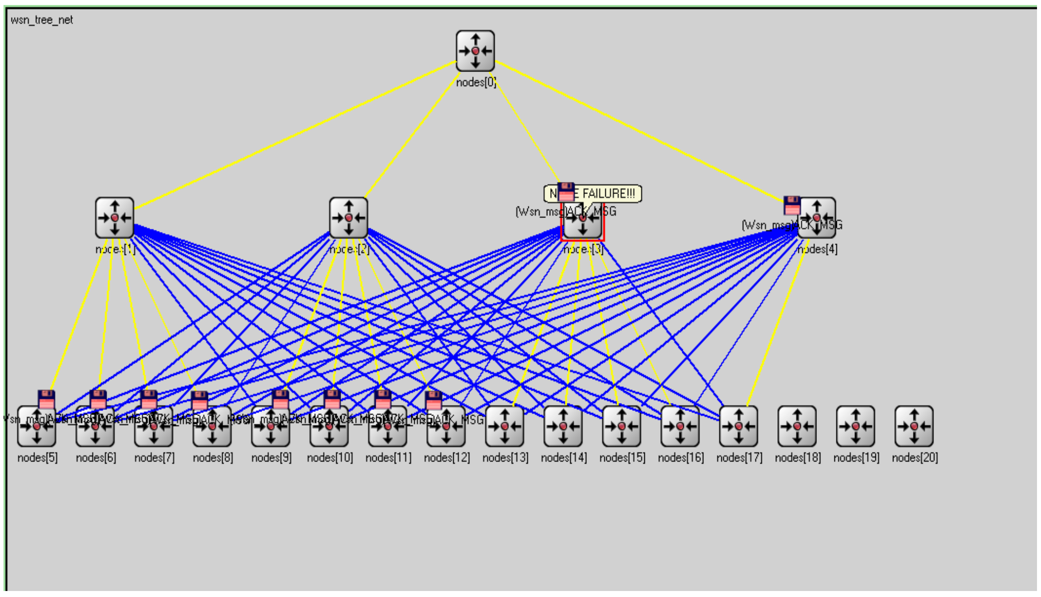

wsn_tree_net
nodes[0]
nodes[1]
nodes[2]
NODE FAILURE!!!
(Wsn_msg)ACK_MSG
nodes[3]
nodes[4]
nodes[5] nodes[6] nodes[7] nodes[8] nodes[9] nodes[10] nodes[11] nodes[12] nodes[13] nodes[14] nodes[15] nodes[16] nodes[17] nodes[18] nodes[19] nodes[20]

Figure 40: *a shot of simulation, in this instant the node 3 is failed*

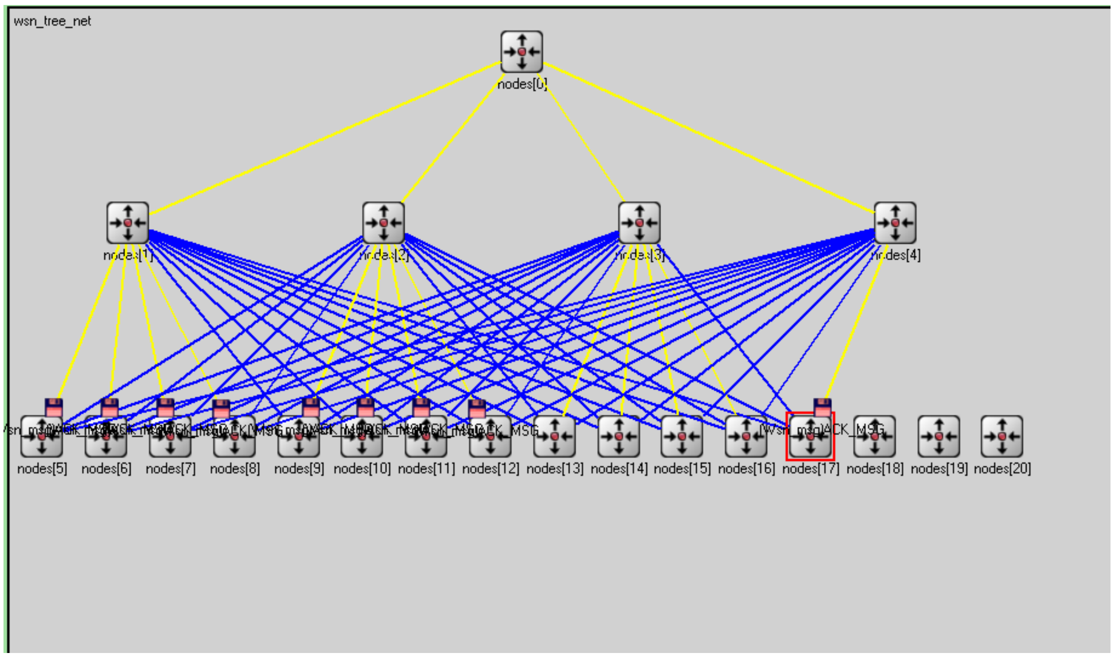


Figure 41: *a shot of simulation, in this instant all the child nodes have accepted an ACK message except for the child nodes of node 3*

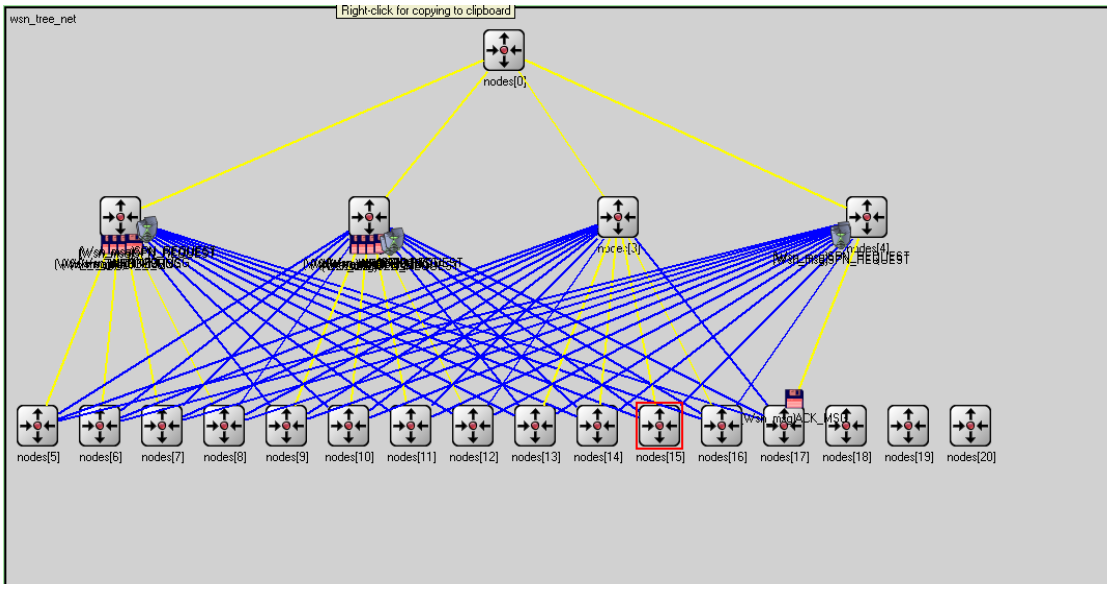


Figure 42: *a shot of simulation, in this instant spn_requests are send from the child nodes of the failure node to the other*

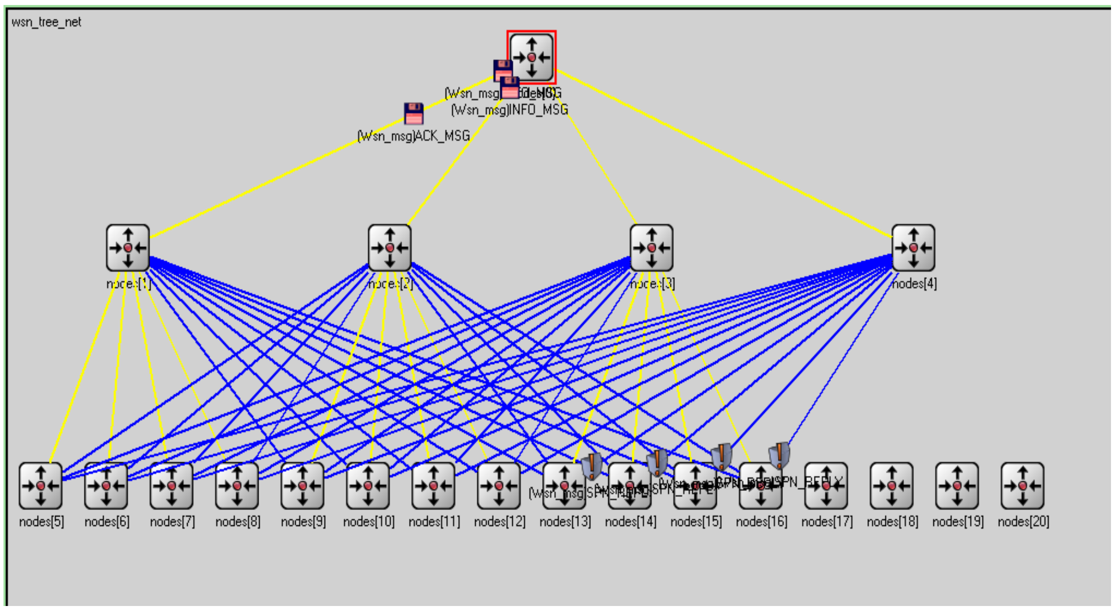
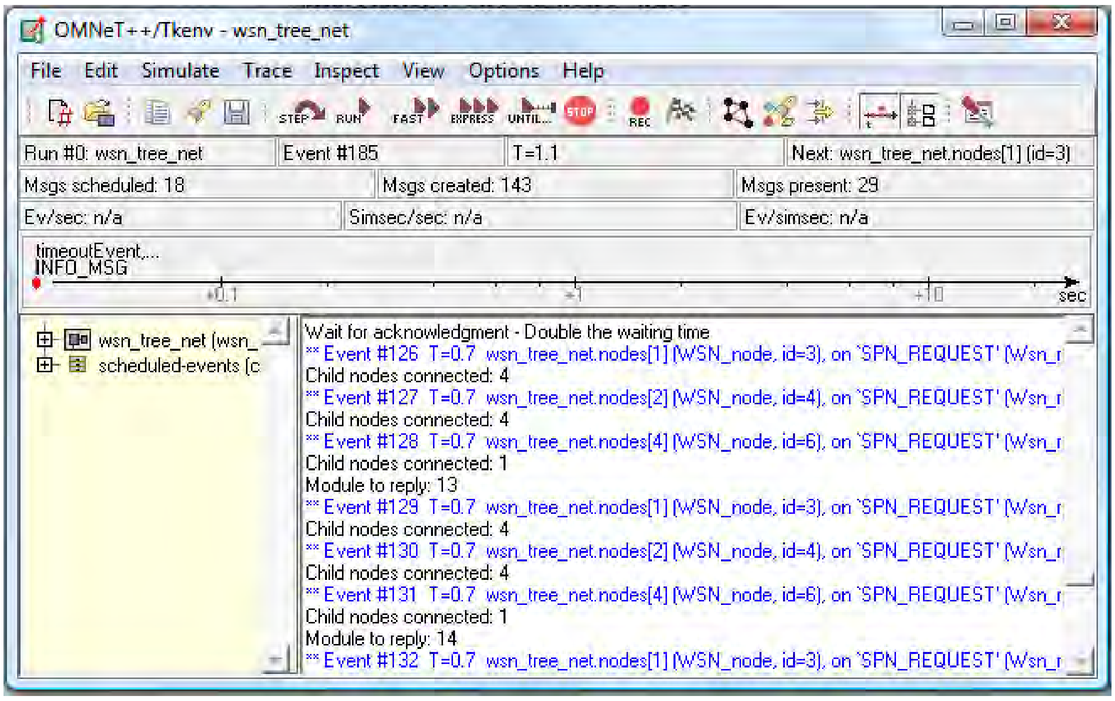


Figure 43: *a shot of simulation, in this instant spn_replies are send from the node4 to the child nodes of the failure node, the other possible step parents inform that are not available*

Then the node 4 becomes the step parent of the child nodes of the failure node 3.


OMNeT++/Tkenv - wsn_tree_net
File Edit Simulate Trace Inspect View Options Help
STEP RUN FAST EXPRESS UNTIL... STOP REC
timeoutEvent,...
INFO_MSG
+0.1
+1
+10
sec
wsn_tree_net (wsn_
scheduled-events (c

```
Run #0: wsn_tree_net    Event #185    T=1.1    Next: wsn_tree_net.nodes[1] (id=3)
Msgs scheduled: 18    Msgs created: 143    Msgs present: 29
Ev/sec: n/a    Simsec/sec: n/a    Ev/simsec: n/a
```

```
Wait for acknowledgment - Double the waiting time
** Event #126  T=0.7  wsn_tree_net.nodes[1] (WSN_node, id=3), on `SPN_REQUEST' (Wsn_r
Child nodes connected: 4
** Event #127  T=0.7  wsn_tree_net.nodes[2] (WSN_node, id=4), on `SPN_REQUEST' (Wsn_r
Child nodes connected: 4
** Event #128  T=0.7  wsn_tree_net.nodes[4] (WSN_node, id=6), on `SPN_REQUEST' (Wsn_r
Child nodes connected: 1
Module to reply: 13
** Event #129  T=0.7  wsn_tree_net.nodes[1] (WSN_node, id=3), on `SPN_REQUEST' (Wsn_r
Child nodes connected: 4
** Event #130  T=0.7  wsn_tree_net.nodes[2] (WSN_node, id=4), on `SPN_REQUEST' (Wsn_r
Child nodes connected: 4
** Event #131  T=0.7  wsn_tree_net.nodes[4] (WSN_node, id=6), on `SPN_REQUEST' (Wsn_r
Child nodes connected: 1
Module to reply: 14
** Event #132  T=0.7  wsn_tree_net.nodes[1] (WSN_node, id=3), on `SPN_REQUEST' (Wsn_r
```

Figure 44: *this window runs simultaneously with tree network and shows what happens in the network every instant of the simulation*

8. Results

In this simulation is tried to create a spn algorithm similar to this of the paper but with some important differences. The scenarios that have been used are more complex. (e.g. some nodes may have only one child node or some may not have any child node).This has as an aim the evaluation of the different reactions may have the network. In general has as an aim the evaluation of the spn algorithm which proposed Gee Keng Ee, Chee KYun Ng, Nor Kamariah Noordin and Borhanuddin Mohd Ali in their paper. After the writing of a great piece of code are tried many examples. A lot of effort has been given in order to thoroughly evaluate this simulation. In specific, a range of values have been used in the 'wsn_tree_net.height' and in the 'wsn_tree_net.mc', in the omnetpp.ini file which as mentioned in the previous chapter. As mentioned in the previous chapter if anyone want to change the values of variables should change the values from the omnetpp.ini file and then rebuild it. Thus, initially is clearly observed that there are not great differences in the reaction of the system. The network reacts with the same way in any value of height and in any value of MC. It follows the same procedure.

In specific, the different values of the MC affects on the time. This means that when the value of MC is high the procedure of communication via the nodes lasts more than in the case where the value of MC is low. The same happens when the value of net's height is high. On the other hand, when these two variables have high value, the node which search for a step parent node has much more possibilities to find the right node easier and faster.

Furthermore, the value of the MC and network's height affects the possibility of errors. When a node which search for a step parent, has to send request messages to many neighbors, the great number of neighbors may increase the possibility of an error. In specific, the request sender may send to a wrong node a request or the node which send the reply may send wrong information about its condition (if has the ability to be a step parent) or may send the reply to a wrong destination.

Finally when the value of MC is high then the data rate is high, while more data packets should be send in the parent nodes in the same time.

9. Conclusion and Future Work

By trying to follow the initial program the dissertation over the 6LoPAN has finished. The 6LoWPAN routing is a very new area which aggregates the interest of many experts. For this reason, in order to reach the 6LoWPAN routing, have to make a mention of 6LoWPAN and the two parts which compose this type of network, the WSN and the IPv6. In general the combination of WSNs and IPv6 is also a quite new and very interesting subject. Thus, initially in this paper, a brief and explicative commentary with the basic points, of the general concepts, WSNs, IPv6 and 6LoWPAN, are presented. Thereafter, the paper and the simulation focus on the main subject of the project, as mentioned before, on the 6LoWPAN routing.

The simulation has as a starting point the SPN algorithm. In this paper, the simulation tries to give different values to the variables of tree network topology, in order to evaluate the reaction of the network in any different case. As far as the network's reaction is concerned there are not remarkable changes, in the different values. The differences have to do with the time period, the error rate and overhead.

Further work, will address the optimization of the SPN algorithm by reducing the percentage of the breaking links and of the error messages. Moreover, a further study could be the development of a new algorithm and why not of a new protocol over 6LWPAN because there are not many.

10. References

[1] Joel J. P. C. Rodrigues, Paulo A. C. S. Neves, "*A survey on IP-based wireless sensor network solutions*", published online in Wiley InterScience www.interscience.wiley.com, 2010

[2] www.ipv6.com

[3] C. S. Raghavendra, Krishna M. Sivalingam, Taieb Znati, "*Wireless Sensor Networks*", 2004

[4] F. L. Lewis, "*Wireless Sensor Networks*", appeared in *Smart Environments: Technologies, Protocols and Applications*, New York, 2004

[5] Jason Lester Hill, "*System Architecture for Wireless Sensor Networks*", Spring 2003

[6] Gregory J. Potie, "*Wireless Sensor Networks*", Ireland, June 1998

[7] Pedro Silva Girão, George Alexandru Enache, "*WIRELESS SENSOR NETWORKS: STATE OF THE ART AND FUTURE TRENDS*" 2nd National Conference, October 2007

[8] John A. Stankovic, "*Wireless Sensor Networks*", University of Virginia, October 2008

[9] Karl Mayer, Wolfgang Fritche, "*IP-Enabled Wireless Sensor Networks and their Integration into the Internet*", proceeding of the First International Conference on Integrated Ad Hoc and Sensor Networks, May 30 – 31 2006, Nice, France

[10] Chris Townsend, Steven Arms, "*Wireless Sensor Networks: Principles and Applications*", Chapter 22, 2004

[11] Ian F. Akyildiz, Mehmet Can Vuran, "*Wireless Sensor Networks*", WILEY 2010

[12] Ian F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirichi, "*Wireless Sensor Networks: a survey*", January 2002

[13] "*An Overview of IPv6*", Chapter 2, 1997

[14] Jari Arkko, Tuomas Aura, James Kempf, Vesa-Matti Mäntylä, Pekka Nikander, Michael Roe, "*Securing IPv6 Neighbor and Router Discovery*", Atlanta 2002

[15] Robert M. Hinden, "*IP Next Generation Overview*", Communications of ACM, June 1996, Vol. 39, No. 6

[16] J. Hui, P. Thubert, "*Compression Format for IPv6 Datagrams in Low Power and Lossy Network (6LoWPAN)*", 2011

[17] Naveed A. Abbasi, "*6LoWPAN: IPv6 for Battery-less Building Networks*", August 2009

[18] Florent Parent, Regis Desmueles, "IPv6 Tutorial", 13 March 2000

[19] Jordi Palet, "*IPv6 Tutorial*", December 2006

[20] Geoff Mulligan, Carsten Borman, "*IPv6 over Low power WPAN WG (6lowpan)*", http://6lowpan.tzi.org, 2009

[21] Jonathan W. Hui, David E. Culler, "*IP is Dead, Long Live IP for Wireless Sensor Networks*", November 2008

[22] Charles "Chuck" Sellers, "IPv6 *Embedded Systems and Sensor Networks*", December 2009

[23] Mark Blanchet, "IPv6 Primer for Sensor Networks", 2006

[24] Jonathan W. Hui, David E. Culler, "*Extending IP to Low-Power, Wireless Personal Area Networks*", July – August 2008

[25] J. Hui, P. Thubert, "*Compression Format for IPv6 Datagrams on 6LowPAN Networks*", April 2010

[26] K. Kim, H. Kim, S. Daniel Park, J. Lee, "*Interoperability of 6LowPAN*"

[27] Md. Sakhawat Hossen, A. F. M. Sultanul Kabir, Razib Hayat Khan, Abdullah Azfar**,** "*Interconnection between 802.15.4 Devices and IPv6: Implications and Existing Approaches*", International Journal of Computer Science Issues, Vol. 7, Issue 1, No. 1, January 2010

[28] S. Chakrabarti, E. Nordmark, "*LowPAN Neighbor Discovery Extensions*"

[29] Kevin Dominik Korte, "*Evaluation of 6LowPAN Implementations*", May 2009

[30] Lars Schor, "*IPv6 for Wireless Sensor Networks*", Zurich, June 2009

[31] Gee Keng Ee, Chee Kyun Ng, Nor Kamariah Noordin, and Borhanuddin Mohd. Ali, "*A Review of 6LoWPAN Routing Protocols*", Department of Computer and Communication Systems, Faculty of Engineering, Universiti Putra Malaysia

[32] M. Harvan, J. Sch¨onw¨alder, "*A 6lowpan Implementation for TinyOS 2.0*", 2007

[33] Alessandro Ludovici, Anna Calveras and Jordi Casademont, "*Forwarding Techniques for IP Fragmented Packets in a Real 6LoWPAN Network*", Barcelona January 2011

[34] E. Kim, D. Kaspar, C. Gomez, C. Bormann, "Problem Statement and Requirements for 6LoWPAN Routing", draft – expires August 2011

[35] S. Chakrabarti, E. Nordmark, "*LowPAN Neighbor Discovery Extensions*"

[36] N. Kushalnagar, G. Montenegro, C. Schumacher, "*IPv6 over Low Power Wireless Personal Area Network (6LowPAN): Overview, Assumptions, Problem Statement, and Goals*"

[37] M. Harvan, J. Sch¨onw¨alder, "*A 6lowpan Implementation for TinyOS 2.0*", 2007

[38] Vassil Stefanov, "*Mesh routing for IPv6 over 802.15.4 on TinyOS*", Computer Science Jacobs University Bremen Germany, May 2008

[39] Ricardo Silva, Jorge Sá Silva and Fernando Boavida, "*Evaluating 6lowPAN implementations in WSNs*", Department of Informatics Engineering University of Coimbra, Portugal

[40] Vladimir Iliev, "*Mesh Routing for Low-Power Mobile Ad-HocWireless Sensor Networks Using LOAD*", Computer Science Jacobs University Bremen, 2007

[41] Ki-Hyung Kim, S. Daniel Park, "*Routing Protocol Comparison for 6LoWPAN*", *6LoWPAN WG, IETF64, Vancouver*, 2005

[42] Iliyan Zarov, "Mesh Routing for Low-Power Mobile Ad-HocWireless Sensor Networks Using DYMO-low", Computer Science Jacobs University Bremen, Germany 2007

[43] C. Gomez, P. Salvatella, O. Alonso, J. Paradells, "*Adapting AODV for IEEE 802.15.4 Mesh Sensor Networks: Theoretical Discussion and Performance Evaluation in a Real Environment*", Proceedings of the 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06

[44] Ki-Hyung Kim, S. Daniel Park, G. Montenegro, S. Yoo, "*6LoWPAN Ad Hoc On-Demand Distance Vector Routing (LOAD)*", 2005

[45] Ki-Hyung Kim, S. Daniel Park, G. Montenegro, I. Chakeres, S. Yoo, "*Dynamic MANET On-demand for 6LoWPAN (DYMO-low) Routing*", 2005

[46] Lingeswari V Chandra, Kok-Soon Chai and Sureswaran Ramadass, Gopinath Rao Sinniah, "*Mechanism to Prevent Disadvantageous Child Node Attachment in HiLOW*", International Journal of Computer Science and Information Security (IJCSIS), Vol. 008 No. 003, 2010

[47] Stefan Dulman, "*WSN Simulation Template for OMNeT++*", University of Twente, department of computer science