Staffordshire University

TEI of Larissa

Heuristic Approaches For Near Optimality In Sensor Placements

Linardos Evangelos

Advisor: Kokkinos Konstantinos

Master of Science in Network Computing

June 2014

Table of Contents

1. Introduction 4
2. Wireless Sensor Networks 7
   2.1 Introduction 7
   2.2 The description of Wireless Sensor Networks 9
   2.3 The incentive and the short story of Wireless Sensor Networks 11
   2.4 Wireless Sensor Networks and their Applications 13
   2.5 Application Development Environments 15
   2.6 Network Design, Challenges and the Future 17
3. Literature Review 20
4. Problem Formulation and Methodology 30
   4.1 Incentives 30
   4.2 Definition of the problem 31
   4.3 Description of Sensing Regions 31
   4.4 Methodology 32
5. The placement algorithm 33
   5.1 A small description 33
   5.2 The main code 35
6. The ComCost Algorithm 40
   6.1 Network model 40
   6.2 Omnet++ 41
   6.3 ComCost Algorithm Implementation 43
      6.3.1 Input 44
      6.3.2 Assumptions 44

6.3.3 The main description 45
6.4 Simulation tests and results 47
Conclusion 54
Future Work 54

References 55

Appendix 59

List of figures

Figure 2.1: The pervasive computing as a combination of science 8
Figure 2.2: A typical example for a WSN 9
Figure 2.3: The pilot experiment of the MIT Lincoln laboratory 12
Figure 3.1: An $r - strip$ (b) the pattern P_1 (c) the pattern P_2 21
Figure 3.2: (a) Overall sensing coverage of a network with 350 nodes in a 100X100m^2 region, (b) scheduling result with r as 6 m, (c) network is connected before node scheduling with R = 9 m, (d) active nodes are disconnected with R = 9 m, (e) network is connected before node scheduling with R = 12 m and (f) active nodes are disconnected with R = 12 m 24
Figure 3.3: A grid network with unreliable nodes 26
Figure 3.4: Voronoi Diagram of a Set of Randomly Placed Points in a Plane 28
Figure 6.1: Network representation 41
Figure 6.2: A hierarchical model-network 42
Figure 6.3: The ComCost network in Omnet 47
Figure 6.4: The ComCost network 48
Figure 6.5: The Simple ComCost network 48
Figure 6.6: The Heavy ComCost network 48
Figure 6.7: Full ComCost network 51

1. Introduction

The history of the sensor networks is long and is placed on back as far as the 1950's. As the authors in [1, 2] mentioned the first obvious sensor network was the Sound Surveillance System (SOSUS). It is described that the US deployed the SOSUS which was made up of an array of acoustic sensors that were interconnected by wired cables in deep ocean basins during the Cold War to detect and track Soviet submarines. The army and generally the military use mainly contributed to the development of sensor networks, in its early stages, in which sensor nodes were wired together to provide battlefield surveillance.

Recent advances in micro-electro-mechanical systems (MEMS) technology, wireless communications, and digital (MEMS) technology, wireless communications and digital low-power have cause the strong interest in sensor networks [3, 4] and as a result have succeeded to produce multifunctional sensor nodes that are small in size and have the ability to communicate in short distances [5]. The definition of a sensor network is that consists of sensor nodes with a short-range radio and on-board processing capability. With the leverage of these sensor nodes wireless sensor networks are becoming increasingly popular and being developed for use in monitoring a host of environmental characteristics and spatial phenomena across the area (WSNs) of deployment, such as light, temperature, sound, and many others [6]. Most of these data have the common characteristic that, they are useful only when considered in the context of where the data was taken from and so most sensor data will be stamped with position information. Because the sensors nowadays have low cost, we can nowadays deploy wireless sensor networks with a very high node density. The authors in [7] have mentioned that this density can reach up to 20 nodes/m^3. When these sensors intend to send their report messages simultaneously, the network would suffer from excessive energy consumption. The processing of some high-level sensing tasks in a collaborative fashion, and the periodically querying by an external source for reporting a summary of the sensed data/tasks is the purpose of a sensor network. To be explained better that I say I will use an example. Can you imagine a large number of sensors can be scattered in a battlefield for surveillance purposes to detect certain objects of interest, like tanks. The reporting of the number of tank sightings at 15 minute intervals for the next 48 hours in a specific region within the battlefield could be a typical query.

For sensor networks have emerged several new design themes. On the one hand as we say because of the sensor has only limited battery energy, network as a whole must minimize total energy usage in order to enable untethered and unattended operation for an extended time. Another disadvantage in this case scenario is the phenomenon of packet collision when packets from various

sensors are sent to sinks simultaneously. However, when we are limited by the number of sensors that can be placed, and this number is rather small, it is significant to deploy them at most informative locations. So, it is understood that localization plays an important role in wireless sensor network applications when the positions of nodes cannot be decided before hand or, if nodes are mobile. Moreover, due to the nature of wireless communication, poor link qualities, such as those between sensors which are too far apart, or even nearby nodes that have obstacles in front of them (e.g. walls, radiation from appliances e.tc), require a large number of retransmissions in order to collect the data effectively. These retransmissions consume a lot of battery power, and they decrease the overall lifetime of the sensor network. Also, the communication cost is a fundamental constraint which must be taken into account when placing wireless sensors. Furthermore, the network must be self-configuring and highly fault-tolerant as the sensors may be deployed in an "ad hoc" fashion. The self-organizing of the network would be a technique to optimize energy usage during query execution. By this way could become feasible to response to a query, into a logical topology involving a minimum number of sensor nodes that is sufficient to process the query. During the query execution would participate, namely communicate with each other only the sensors in the logical topology. More specifically when there are many sensors in the network if it is used this very effective strategy for energy conservation is necessary to process a given query.

The number of control messages used in the self-organization process must be small, so that the overhead of the technique does not offset the expected benefit completely rather than the above technique to be of value. It is important to be noted that the overhead is paid only once for a given query, but the benefit is reaped during each execution of the query.

The existing work on sensor placement under communication constraints focus on the geometric perspectives of the problem [8, 9]. In these researches, the sensors can only communicate with other sensors which are in a specified distance apart at most R. Analyzing the above assumptions, we must deal with the following two sub-problems: Firstly, the notion of a convex implies that sensors can perfectly observe everything within the convex and nothing outside, which is not always true. Secondly, the assumption that two sensors which are at some specific location and either communicate (connected), or not communicate (disconnected) is not always a good assumption, because here we do not consider the variability in the link quality because of obstacles as we mentioned above. Note that, the authors in [10] have analyzed exactly this situation and have come to the conclusion that this is a rather important issue that reflects to the quality of communication. In order to avoid the sensing region assumption above, we found a work by [11], where the authors established probabilistic models for this reason. More specifically, these

models predict the sensing quality of the network by assuming a correlation between the sensor locations.

In this Thesis, we design and analyze two competing algorithms, one for the spatial coverage of an area of sensors into an optimal logical and another for monitoring and reducing of communication costs. More specifically, the approximation algorithm that we have designed for the spatial coverage of an area of sensors constructs a topology that her size is about $O(\log n)$ factor of the size of an optimal topology, where n is the number of sensors in the network.

In the rest of our Thesis we refer on the literature review in which is described all the scientific works upon the fundamental topics. Then we provide a formulation of the problem and we analyze the methodology of the work which is one of the most significant tasks with examples and we describe the motivations. Moreover, we use a network simulator that is named Omnet++ and with the help of the programming language C++ we make some simulations and we analyze their results on the communication costs of the examples that we use.

2. Wireless Sensor Networks

2.1 Introduction

In the late 1990s developed into the world of calculation users and IT, both visions of "Ambient Intelligence" and "Pervasive Computing." These concepts refer to an abstract environment which is able to "feel" and understand people, and general entities that are in it [12]. More specifically, in such an environment there are arbitrarily many devices which communicate and work together with a common goal of a human service. The service has to do the daily tasks such as automatic arming or disarming alarm, automatic watering, automatic opening of lights or windows, and adjusting the temperature of the house when the owner enters, but more advanced technologies such as tackling a fire in a public building or demonstrating new products appropriate for the consumer habits of a family.

It is therefore evident that this vision, although quite promising, in final form includes many technologies and disciplines together. We can describe all these like a relatively new interdisciplinary field. For the implementation of this vision are needed basic tools (knowledge and algorithms) of disciplines such as information theory, fault tolerance, networks, the wireless mobile networks, Wireless Sensor Networks, Distributed Systems, Security and many others. A combination of descriptive these sciences with the end result of these pervasive computing is shown in Figure 1.1.

One of the most important tools of the diffuser computation is the Wireless Sensor Networks (WSNs). The appointing authority consisting are a large number of nodes covering adequately the environment which is under surveillance. That is the subsystem which record all information from the environment and at the same time is the same footnotes system that performs the functions necessary every time. One of the most critical components of the WSNs is the sensor which is the subsystem of node which converts a physical quantity such as humidity, temperature, etc. into electronic data. The scientific community since the early 2000, after having studied in detail the prior art, in relation to WSNs, such as networks, Wireless Networks, Distributed Systems, etc. has focus of much attention in Wireless Sensor Networks after an indispensable tool for pervasive computing.

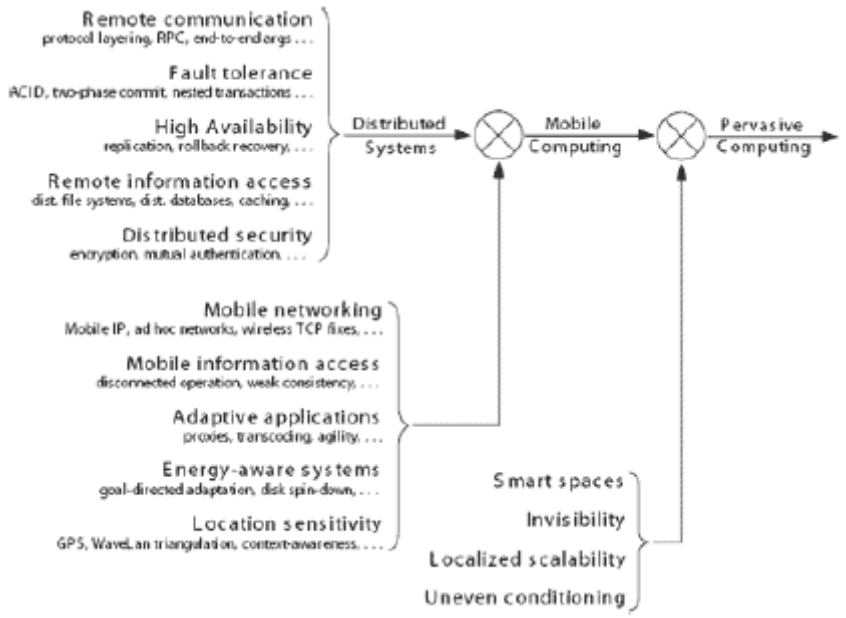


Figure 2.1: The pervasive computing as a combination of science

These last 10 years, WSNs experienced tremendous progress over time. Mathematical models are invented which simulate the full WSNs [13], algorithms that exploit fully their properties in order to have better performance, upper and lower bounds in yields algorithms and more. The most important limitation of their which found the beginning of the investigation is the limited resources of nodes energy expert. The limited energy capacity of a node is the key point which actually differentiates WSNs from traditional wireless networks. Therefore the more research on the WSNs, even 10 years later, is related to the minimization of energy consumption. It is now accepted that with today's technology the WSNs cannot be operated indefinitely without human intervention which significantly affects the vision of pervasive computing. Because if each node in a WSN requires human intervention at regular intervals then the cost of the overall system increases significantly, a sector of the start of mass production of such systems is taken seriously and ultimately determines their success. But even apart from the cost factor, an appointing authority where nodes due to limited energy closed down unexpectedly after the network becomes unreliable failing its primary goal which is to help people. However, a new technology invented by scientists at MIT [14] is to fundamentally change expectations in the field of WSNs. This technology enables wireless transmission of energy from a source to a receiver with very small energy losses.

2.2 The description of Wireless Sensor Networks

A Wireless Sensor Network (WSNs) consist of autonomous distributed nodes (or sensors) where everyone takes measurements for proximal environment, such as temperature, humidity, noise, seismic, pressure and even motion provided always that there is necessary hardware on the node that you can take each measurement. The nodes take measurements periodically with a period that depends on the type of the network and is regulated by the original designer of the network. For a network nodes can take measurements every hour while in another network sensors may take measurements per second. In a WSN almost always considered that there is another very important point: the Source. Each node taking and record a size measurement sends these data, structured in packets through a routing protocol, the source using as intermediaries other nodes. It should be noted that each node knows the total path follow packages but knows only the first neighbor to which sent each time a packet is in the source. An example of a wireless sensor network is shown in Figure 1.2.

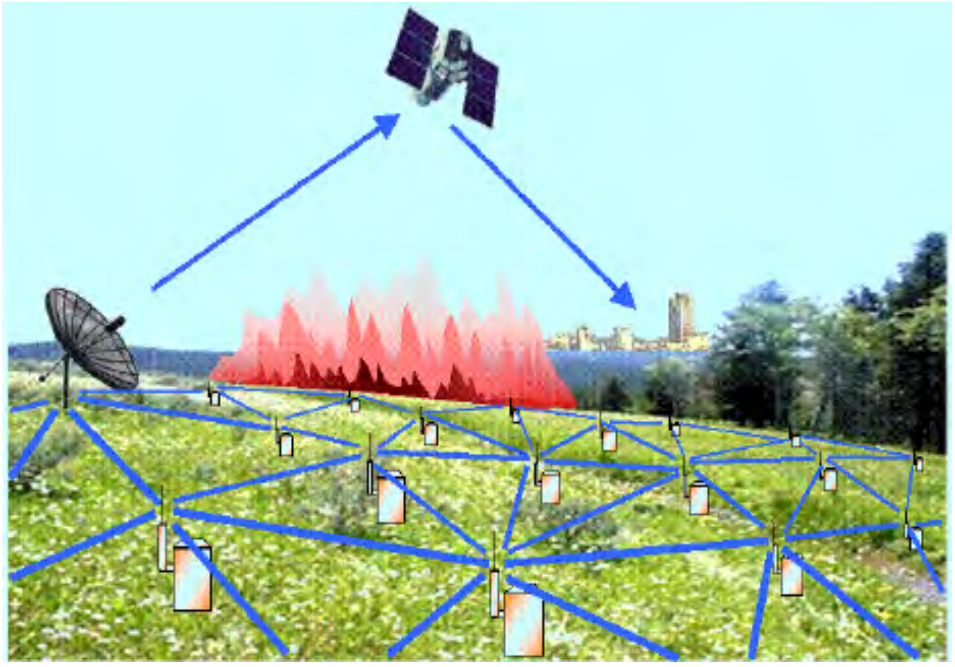


Figure 2.2: A typical example for a WSN

The main criteria that characterize a network as WSN are:

- ✓ major limitations on the energy consumption for nodes

- ✓ ability of the network to be able to deal efficiently glitches and bugs nodes
- ✓ ability to cope with failures and communication problems between nodes
- ✓ heterogeneity in nodes
- ✓ locality to information
- ✓ network scalability to very large sizes
- ✓ Easy installation and use of the overall network of an operator

Regarding the node typically consists of some typical parts:

- ✓ transmitter and receiver which uses low power wireless protocols
- ✓ microcontrollers of low energy consumption, from the very simple 8bit up and modern with great computing power. The main feature is the low power consumption
- ✓ battery and generally an energy resource
- ✓ sensors such as thermometer, barometer, camera etc.

Although there is the technology to build a hardware node in size of "lice" batteries dramatically increase the size so that a node works flawlessly for a long time. It should also be noted that, although we believe that there are limited resources in hardware for example, the capabilities of the microprocessor and thus microcontroller that carries a node, WSNs are not studied cases where nodes do not have control of their movement or no memory and ability to perform basic operations. Such cases are subject to other models such as the Population Protocols [15].

The Source is a separate network element, which has a huge computational capacity in relation to hubs, have no limiting resources while having much greater communication range of the nodes. In some networks Source participates in routing protocol or partially helps the whole network using the particular characteristics of. The source eventually reaches all the information and is responsible for manage data information such as performing specific queries onto data safe and fast conclusions about the network.

The basic Wireless Sensor Networks have been stopped at this point. The Manipulation of the data and the export-after information from these data are lots of interdisciplinary fields of Pervasive Computing and Ambient Intelligence. It is therefore clear because the appointing authority is the most critical component in the above disciplines.

Finally, noting that in WSNs generally considered that the sensors have cooperative tendencies, namely the total network has a common goal. Rather rarely studied where each node faces individually and selfishly his interests. The analysis of these cases uses usually game theory [16], and indicates a fully heterogeneous network different from the original philosophy of WSNs.

2.3 The incentive and the short story of Wireless Sensor Networks

The main motive of WSNs was primarily military applications. Overview of the battlefield, target recognition, monitoring and recording of military forces and the available ammunition was of the first applications that have fueled the posting of tubers on governments of leading countries, for the inspection of such mechanisms. Specifically, the first real research on sensor networks started within in the 1980s to work with the U.S. Department of Defense, the DARPA (Defense Advanced Research Projects Agency). This service started a new program that is called Distributed Sensor Networks (DSN). Simultaneously the ARPANET (Advanced Research Projects Agency Network) network was fully operational with 200 universities and research centers connected. The basic idea of the DSN was a network many distributed nodes connected who cooperate but each node running independently. The routing protocol used was based on the method of flooding. If taken into account that at that time there were no personal computers, let alone laptops, while Ethernet was just beginning and acquire fame, the DSN program was probably too ambitious. The technology for the realization of this project had been a relatively recent conference [17]. Essentially included a sound sensor, ability to send and receiving signals of low energy and the necessary software distributed nature. Scientists from the University Carnegie Mellon (CMU), focused their interest on the construction of an operating system which will be addressed in such devices, namely devices connected to a network in which there is easy and unified access to distributed resources of a reliable system DSN. The result of this system was to provide the Mach operating system for which season was quite pioneering data [18] and even had some limited commercial success. Later, scientists at MIT university focused on recognition (hostile) helicopter through editing audio signals. The system they used was distributed microphones scattered in an area which send their information to a central computer. They used heuristic algorithms and techniques for matching be able to achieve results in the recognition accepts helicopters. Moreover they extended the system DSN adding algorithms for signal processing and matching techniques [19]. To demonstrate the system at MIT Lincoln Laboratory developed a real test scenario for acoustic recognition helicopters and low flying planes [20]. Used nodes actually were microphones which wireless technology to send audio signals to 3 desktop computers (processor MC68000, 256KB memory and shared memory 512KB). In Figure 1.3 [21] shows the test scenario. Finally, the overall system worked successfully managed to identify low-flying helicopters and aircrafts.

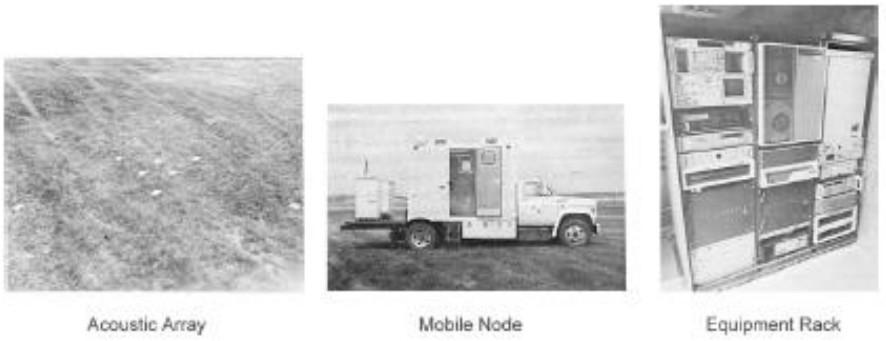


Figure 2.3: The pilot experiment of the MIT Lincoln laboratory

While researchers have understood that such systems should include many small sensor networks which have hundreds nodes, the technology for these systems was not fully ready. However military factors had recognized the immense usefulness of these systems and the superiority of network arms: thousands sensors together and gather information in relation to the enemy and send the operations center, which may be miles away from the battlefield. That pattern would work fatal in the course of a battle.

The first real system was implemented with this purpose and has enough relative to the appointing authority is the CEC (Cooperative Engagement Capability) which was constructed by the U.S. Navy in the mid of 1990s. The system consists of several radar which collects information on air targets such as aircraft, helicopters, missiles, etc. These measurements are sent to a leader node (essentially a source) that processes and filters the information. The node that is common to all other nodes that collect information. The important element of the system is that all nodes have access to all the information, creating a truly distributed system which gives the same image to all military who is in a node everyone. This system followed by other strategic systems similar objectives as the REVMASS (Remote Battlefield Sensor System) and TRSS (Tactical Remote Sensor System). At the same time the technology in computers is evolving rapidly. They were manufactured wireless networks, had created the Internet, microprocessors now had enough computing power while there was the possibility of constructing microcomputer systems were size as a palm. These developments and visions of Ambient Intelligence and Pervasive computing made scientists to envision a different side of the sensor networks: wireless sensor networks which will collect information in order to help the man. The WSNs namely aimed to make easier the life of man from each side taking some of these functions in accordance with the information collect but man can interact with the overall system. But it was immediately apparent that such systems would be readily susceptible to energy management. Because of all other technologies have evolved rapidly, progress in battery technologies were much less progress while such systems should operate almost forever without human interaction. Also found that algorithms

for conventional wireless networks (such as ALOHA, slotted ALOHA, etc.) will not be could be used as wireless networks are designed to maximize performance while appointing authority aimed at maximizing intervals between nodes with limited energy resources, send information to the source.

From 2000 onwards, the attention of the scientific community mainly focused on minimizing the consumption of energy in appointing or this is achieved through the routing protocols or by way development of sensors or any other technique that could make the appointing authority to resist even more time. Also started to massively manufactured sensors were commercially available for practical and research purposes while simultaneously created first operating system specifically for the WSNs, the TinyOS. Directly the first experimental applications created with the WSNs which were almost all areas of human activity.

2.4 Wireless Sensor Networks and their Applications

The technology of wireless sensor networks can be applied to many real-world applications and bring up some completely new. A critical and primary component of nodes of wireless sensor networks is the sensor. For many aspects of the natural environment is appropriate sensor technology that can incorporated in a WSN. The most widely used are sensors temperature, humidity, sound, pressure, and chemical sensors. A brief list of the most basic applications is presented below:

- **Disaster Prevention:** One of the most frequently reported applications of SAA is to prevent disasters. A typical scenario for applications of this class is to detect fires. The nodes sensors are equipped with thermometers and can calculate their position running a positioning algorithm (localization). These nodes we can stretch out in a forest, throwing from a plane. This forms a temperature map for the region and in case of high temperature and low humidity suggesting fire informs firefighters.
- **Control of the environment and biodiversity:** The WSN can be used to control the environment for chemical pollutants or even to form an image on the number different species of flora and fauna of a region.
- **Intelligent Buildings:** Large buildings often consume large amounts of energy due to improper use of appliances Air Condtitioning (HVAC). An efficient, real-time and accurate monitoring of

temperature, humidity and other parameters can reduce energy consumption. Also can be used for monitoring of mechanical stresses in buildings or bridges located in seismic active zones, while other types of sensors can be used for detecting the trapped people in cases of earthquake. The sensors can be installed in buildings the time of construction or have been built since. In these applications, energy savings for sensors is very important requirement.

- **Facility Management:** The WSNs can be used for applications to manage large facilities, such as security. The input of people on site can be done without keys, but using a transmitter and can detect potential intruders. Also in the chemical plants WSNs could be used for leak detection.
- **Maintenance Machines:** Sensors can be placed in inaccessible places machines to control vibrations indicate need for maintenance. Examples of such machines are automatic machinery or axles of trains.
- **Applications in Agriculture:** The application of WSNs in croplands by placing sensors for humidity measurements and analysis of soil structure allows for more accurate and efficient lubrication and irrigation of land. Also, animal husbandry can benefit the placing of sensors in animals that control the state of health them.
- **Applications in health care:** The use WSNs in healthcare can prove very beneficial. But there are several ethical dilemmas on this topic. Potential applications range from direct placing sensors in the patient for monitoring the health and perhaps automatic administer medicinal products up monitoring of doctors and patients in hospitals.
- **Intelligent road systems:** In the intelligent road systems, the sensors are placed in the streets, even the curbs of streets which collect information on traffic and road conditions generally and communicate with drivers by giving them useful information.
- **Military Applications:** WSNs can be an integral part of military systems. The characteristics of WSNs, as is the quick placement, self-organization and tolerance errors, convert them into a promising technology for military systems. Some of the potential military applications are to monitor the status of equipment and munitions, close monitoring of the battlefield, identification of enemy forces, the assessment of disaster after battle and the identification and recognition of chemical, biological or atomic attack.

2.5 Application Development Environments

A network of sensors to be easily programmable and gives multitude of options on the programmer and the user should running an operating system (OS) which is made especially for systems WSNs. Note that the operating system should cover both users (e.g. who will want to run a specific application for cultivation of the plant) and developers who want to build powerful and reliable applications easily and quickly interval. Generally an operating system intended for wireless networks sensors should have the following characteristics:

- **Small area code:** Given the limited memory of a node, the kernel function should be implemented with the minimum possible code.
- **Low energy consumption:** Due to the nature and limitations WSNs of an operating system destined for WSNs should be done only by the proper management of energy resources.
- **Reliable architecture:** The monolithic OS now considered obsolete because of their architecture. The modern OS because of offering reliability must have microkernel architecture. With this architecture only the basic components of the OS loaded core while all other (file system, communication system, etc.) running as servers. So if a subsystem fails, as the file system, while in a monolithic OS would be set off the entire node, in an OS architecture micronucleus the file system would do a reboot and node would continue to function.
- **Easy programming model:** The programming model has significant influence on the creation of applications. The most familiar programming model is multithreaded with low resource requirements [22].
- **Efficient scheduling:** The scheduling defines the device by entering the core processes of the central processor. But because WSNs are used in numerous applications, there are applications that require flexible scheduling which conserves energy while others, given the nature their real-time scheduling, which depletes the energy of a knot faster. The OS should allow the programmer type of scheduling you want to use.
- **Abstract Communication Interface:** The interface refers both communication processes within a node and the communication between nodes. Because the nodes may be completely heterogeneous among themselves, with other hardware architecture and each of them, the operating system should remove such details from the interface of the programmer.

As revealed in chapter 2.3 of the start of WSNs scientists were trying to create an operating system that has characteristics similar to those reported. The first operational which was massively used is TinyOS in 2000. Thereafter created and other OS for sensor networks, each with a different motivation and target. Summarizes the most common operating systems for sensor networks are:

- **TinyOS:** Developed by the University of Berkeley in cooperation with Intel and Crossbow Technology. It is open source and the first version was released in 2000. Supports a huge number of hardware platforms and the requirements in RAM is only 2KB. The application development language in TinyOS is nesC (Network Embedded Systems C) a variant of C but with object characteristics and intended especially for WSNs [23].
- **Contiki:** It is a small, open source, and fully portable multithreaded OS designed specifically for devices with limited resources. A typical installation takes just 2KB RAM and 40 KB ROM. Is written in C, fully supports IPv6 and can be installed graphical interface, web browser, web server, and much more. The first version was released in 2005 and features a large community that deals with its further development [24].
- **Mantis:** Multithreaded operating system specially designed for microcontrollers with very limited resources. It can run even 500Bytes RAM and requires only 14KB memory ROM. The router makes efficient use of the available energy in putting sleep mode the microcontroller whenever needed. It is written in language C [25].
- **SOS:** The operating system was developed within a project of the UCLA campus in collaboration with other universities. The main motivation for its development was the fact that an application for an operating system WSNs was directly related to the actual OS. Therefore, the transfer in another OS was prohibitive. The OS of SOS has created interfaces that encountered in modern operating systems such as runtime error checking, garbage collection, etc. He moved to microcontrollers but its development is slow progress mainly due to limited use [26].
- **Nano-RK:** Developed by the Carnegie Mellon University with full support multi-hop network. Supports platforms FireFly and MicaZ. Includes core very small but several possibilities and can run on systems with 2KB RAM and 18KB ROM. Supports fixed priority preemptive router thus ensuring that all deadlines are met. The OS can reduce energy consumption through the property providing

applications can define their requirements to resources that will be needed during the execution [27].

- **Maté:** The basic idea of this project is to build a virtual machine which can be installed on top of OS sensor networks. So the Maté would function as the Java running on modern operating systems. But while Java offers plurality of classes creates a very large file size bytecode prohibitive for microcontrollers used in WSNs. Instead, the idea is that in Maté allows the user to select the programming language in which he will write the program, which classes include etc. to reduce the size of the virtual machine and the final file bytecode [28].

Of course there are other tools for rapid application development operating systems in WSNs. For example, have developed simulators discrete time as ns-2 [29] and the youngest ns-3 [30], the OMNeT + + which we will use and analyze in the next chapter in more detail [31], o NetSim [32] and J -Sim [33]. They have also developed frameworks that ready contain implementations of algorithms and models for sensor networks and wireless networks in general. One of the most famous is the wiselib [34], a library which contains functions for routing algorithms, localization, distributed algorithms, etc. while providing full support for almost all platforms sensor networks.

2.6 Network Design, Challenges and the Future

When designing a new sensor network designer has a multitude of choices to make in order to achieve the best outcome in relation to requirements. When the designer mixes various algorithms and models in the design (e.g. an algorithm for routing of packets and a model for the topology of the network) may have worse outcomes than expected. The most important parameters that a designer should to set are:

- ***Development of nodes:*** Since the nodes in a WSN are becoming smaller, their development within a space can be done in many ways. For example, can grow uniformly random (e.g. thrown from a plane) or settle in specific places which are more difficult especially if the number of nodes is large. But the development of nodes may be contiguous. For example in a WSN can be seen that after some time operating a point perhaps needs more nodes to supervise. The way I

finally placed nodes (i.e. if it is uniform or nonuniform their distribution) significantly affects the network performance.

- ***Mobility:*** Mobility can be unexpected (e.g. through air or water) or be predesigned for some nodes. The movement of some nodes on a network can significantly improve the network performance since through their motion can relaxes other nodes going very close to them and getting their data. But moving nodes should have large pores or energy can be recharged with a way because otherwise their energy will finish much faster than the energy of the remaining nodes.
- ***Cost, number of nodes and Available Resources:*** The three concepts are inextricably linked. Larger grid size, i.e. more nodes, or a larger size available resources directly implies dramatic increase in the cost. Keeping the cost of network fixed the designer must choose among a large number of nodes and the available resources, especially energy.
- ***Type of nodes:*** Another factor which has a direct relation to the cost of the network but also with the function and purpose the WSN. Features include node architecture (e.g., processor, RAM sizes and ROM) and the potential their communication as Wifi, Zigbee, Laser, Bluetooth, IrDA, etc.
- ***Uniformity nodes:*** If the appointing authority and usually made of the same type nodes this is not the norm. A network designer can prefer to import 80% of our nodes and 20% more expensive that but have special features like movement or GPS that eventually network has better performance.
- ***Area coverage:*** Depending on the purpose of a WSN different coverage policies are necessary. For example, if a need to WSN detects moving entities within the site then you should each point to at least 3-times capped (i.e. at least 3 nodes can perceive via sensors) so that they can localization algorithms work and be able to find the exact position of the entity. Conversely if this is not necessary since one-time or 2-times coverage is sufficient for the needs of the network.

The design of an appointing authority has a direct relationship with performance. However, the performance has many meanings depending on the purpose of the network. The main performance metrics are as follows:

- **Lifetime:** The most critical but also the most controversial performance metric. Abstract The lifetime of an appointing authority set the time until the network to become useless. In the literature, the definitions differ significantly. Defined as the time to death the first

network node and there are definitions that define it as the time to death for 70% of network nodes.

- **Uniformly energy distribution:** Metric that is directly related to the lifetime of a WSN. A nonuniform distribution of energy leads directly some nodes deplete their energy much faster than others. Therefore created holes (energy holes) network which can break the network into smaller cohesive components and the effect of lost packets.
- **Delay (Latency):** defined as the time it takes from the moment you create an event on the network to learn the source. Depends on hops network but is generally a fraction of second.
- **Reason for Success (Success rate):** Defined as the percentage of received source events in terms of percentage of total events that were created in the space covered by the WSN. Or else the number of packets that were sent correctly to the total number of packets that were sent.
- **Average degree of nodes:** Defined as the average number neighbors of a node. It is very important because if the average number neighbors are one with little dispersion then die if a node with high possibility that the network will break into two smaller subnets.

Of course all these will be counted each time and the standard deviation of each metric.

The vision that there is for WSNs for the future is every man can be purchased and easily installed hundreds of nodes at points he needed. These nodes will automatically set protocols and algorithms to use for the operation have selected. Each node should be immortal, that you do have never been replaced for a very long time. Also WSNs will very use to monitor planets. There, each node will must remain alive for years, maybe decades. Therefore, the full exploitation of mechanisms that help to keep the nodes energy for years is consistent placement of the scientific community.

3. Literature Review

The immediate basic and important challenge in this chapter of our work is thorough search, study and recording of research that has been done so far by other researchers / authors. Firstly, as we mentioned in the chapter of introduction, the existing work on sensor placement under communication constraints focus on the geometric perspectives of the problem [8, 9]. In these researches, the sensors can only communicate with other sensors which are in a specified distance apart at most R. Analyzing the above assumptions, we must deal with the following two sub-problems: Firstly, the notion of a convex implies that sensors can perfectly observe everything within the convex and nothing outside, which is not always true. Secondly, the assumption that two sensors which are at some specific location and either communicate (connected), or not communicate (disconnected) is not always a good assumption, because here we do not consider the variability in the link quality because of obstacles as we mentioned above.

More specifically, the authors in [8] that have the title "Node Placement for Connected Coverage in Sensor Networks" address the problem of optimal node placement for ensuring connected coverage in sensor networks. To reach any firm conclusions consider two different practical scenarios. They underline that for the first case they give solutions that are within a small factor of the optimum. About the second case they mention that are presented an algorithm that runs in polynomial time, and guarantees a constant factor approximation ratio. Furthermore they focus on two important issues in the deployment of wireless sensor networks. The first issue they mention is the coverage in which each sensor device typically has a physical sensing range within which it is able to perform its operation and one goal of a sensor network is that each location in the physical space of interest should be within the sensing range of at least one of the sensors. The other important issue that they mention is named connectivity and about this they add that it is typically more energy efficient to aggregate all the sensor data at a few specific wireless nodes (gateways) from where the data can be uploaded to the remotely located monitoring station. Therefore the sensors need to organize themselves into a connected ad-hoc network. Since the wireless radio in each sensor node also has a transmission range, the location and placement of the sensors determine the connectivity of the sensor network. They underline that wireless sensor networks need to meet both these requirements of coverage and connectivity.

As we say in the above paragraph, they mention that they consider two different practical scenarios. In the first scenario, a set of regions are to be provided connected coverage, while in the second case, a given set of n points are to be covered and connected. They add that for the first case, they provide solutions that are within a small factor of the optimum and for the second case, they present an algorithm that runs in polynomial time, and guarantees a constant factor approximation ratio. General, the optimization goal is to minimize the number of sensors used as they emphasize. For that reason they assume that each sensor is capable of detecting signals within a fixed radius r around it, i.e., each sensor covers all points that are within a circle of radius r

around it. Moreover, they describe that the communication range of any sensor is also r, i.e., a sensor is "connected" only with sensors that are within a fixed radius r from it. Also they add that the sensing and the communicating radii are in general different. In their paper they focus only on the case where the sensing and communication radii are equal and identical for all the sensors.

Additionally they address several important and interesting versions of the optimal connected coverage problem. Firstly they analyze the case in which the region to be covered is the entire two-dimensional plane. They support that the solution for this simple special case provides some valuable insights for approaching the more complex region-coverage problems. They describe that to cover the entire two dimensional plane with r-disks, they need an infinite number of disks and they add that the appropriate optimization metric in this case should therefore be the number of disks used per unit area which they call density. To describe the solution, they first definite a pattern referred here as an r-strip. They describe that an r-strip is a string of r-disks placed along a line such that the distance between the centers of any two adjacent r-disks is r and also the nodes in an r-strip form a single connected component. They continue saying that for every even integer k, place an r-strip oriented parallel to the x-axis such that the point is the center of an r-disk constituting the strip. Also, for every odd integer k, place an r-strip oriented parallel to the x-axis such that the point is the center of an r-disk in the strip.

Finally they describe the connected coverage for a region of finite size. Firstly they assume region that has a convex shape and they describe the way that can be provided connected coverage. Firstly they place the shape on the plane tiled by the r-strips. They include all the non-shaded disks of the pattern P_1 that intersect the region, and exclude the rest. Then they take another r-strip and place it in such a way that it intersects all the other r-strips intersecting the region. This guarantees connectivity as they mention. In the end they note that since the region has a convex shape, it is always possible to place the last r-strip in such a way that it intersects all the parallel r-strips covering the region.

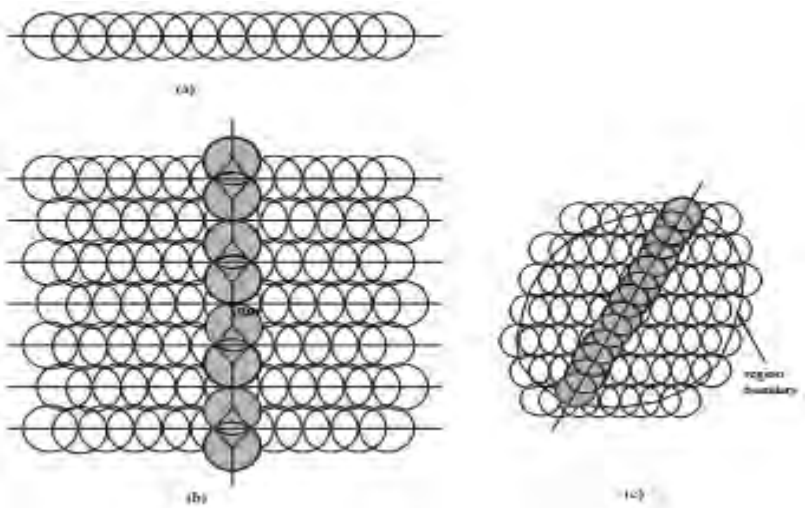


Figure 3.1: (a) An $r-strip$ (b) the pattern P_1 (c) the pattern P_2

In [9] the authors deal with one of the main problems of the wireless sensor networks which is the energy conservation. Since sensors are devices with limited battery life and it is infeasible to replace their batteries we are interested to preserve their energy for the most possible time. The researchers insist that an effective solution for this problem of energy conservation is to schedule sleep intervals for some sensors, while the remaining sensors stay active providing continuous service. This work is similar to the previous work mentioned in the first paper. Again the effort of the authors is concentrated to solve the important problem of maintaining coverage and connectivity in wireless sensor networks and their goal is to keep the minimum number of sensor nodes in active mode. In this way they achieve to maximize the network lifetime, because as they claim, if all the sensor nodes simultaneously operate in active mode, an excessive amount of energy is wasted and the data collected is redundant because of the retransmissions. Also they add that multiple packet collisions may occur when all the sensors in a certain area try to transmit. This is a usual phenomenon in sensor networks as a result of a triggering event. Furthermore they mention that there are some protocols that they try to solve the problem of sensing coverage but they do not guarantee network connectivity. On the other hand, they add that many protocols have been designed to maintain network connectivity but they do not ensure sensing coverage. So for the reason that satisfying only coverage or connectivity alone is not sufficient since nodes may not be able to coordinate effectively or monitor the environment with the required accuracy they study the problem of providing coverage and connectivity in a united framework. First of all, however, they make some assumptions. These assumptions are that sensors have fixed locations and also that the communication range is twice the sensing range. Furthermore, another innovation that they try to complete is to show how to connect a set of sensors that already provides coverage under a less restricting assumption that the communication range equals the sensing range.

Generally they try to create a natural greedy sector cover algorithm that has running time $O(n^2 \log n)$ and which is known to have an approximation factor of $\log m$., namely how many sensors are used in the worst case compared to the optimal solution, where m is the maximal number of sectors covered by a single sensor. Also they underline that the approximation factor of the greedy sector cover is no better than $\Omega(\log m)$. Furthermore to achieve better approximation factors, they propose a simple grid placement algorithm which selects at most $6\pi(\approx 18.84)$ times the number of sensors of the optimum cover, while not guaranteeing full coverage of the region of interest. They believe that this algorithm works very well in dense sensor distributions and they add that in terms of the chosen grid can be bounded the area that remains uncovered. From the scope of a different approach that it based on the assumption that the region of interest P is also covered when decreasing the sensing ranges of all nodes by a factor of$(1-\epsilon)$, it developed another algorithm that is called fine grid and which has a theoretical running time of an exponential dependency on $1/\epsilon^2$. This algorithm, as the researchers mention, produces a solution with full coverage, and whose cardinality is at most 12 times the size of an optimum solution using the reduced sensing

ranges. The covering of convex regions with fat objects are extended from the techniques that are used in this algorithm and which might interesting given that sensing areas in reality are not perfect disks. Additionally they create and present another algorithm that has an approximation factor of 18 and which provides approximate coverage and is named distributed dominating cover algorithm. The time complexity of this algorithm is $O(n)$ and the time complexity is $O(n \log n)$ as they say. Also this algorithm is created to be applied in networks without centralized control. They emphasize on that the coverage that is provided by this algorithm is less accurate compared to that of the centralized algorithms.

More especially, they try to solve the problem as follows: At first they assume that sensors have fixed locations and that the communication range is twice the sensing range. In order to obtain better approximation factors they emphasize on that they concentrate on algorithms that always guarantee connectivity, but provide approximate coverage. The first algorithm that they refer on is named *greedy sector cover algorithm*. They analyze it on two steps. In the first step they use the algorithm of another paper that named "Obnoxious Facility Location: Complete Service with Minimal Harm, International Journal of Computational Geometry and Applications" [9] to check whether the sensors cover the region of interest. Then they derive a simple algorithm called *grid placement*. About this algorithm they support that a specific instance of grid is defined by its position and then they choose exactly one sensor in each cell of the grid to be in the covering set. Finally they add extra sensors to make the covering set connected. Also they mention that the selection of the cell size implies that each sensor covers its cell completely and sensors in neighboring cells are able to communicate with each other. In the end to obtain better coverage, they aim to minimize the number of such cells. For this reason they run the MST connection algorithm. The MST connection algorithm is used for optimizing the grid placement.

In [35] the authors deal with one of the main design challenges is to save severely constrained energy resources and obtain long system lifetime. Because of sensors having a low cost are deployed a large number of sensor nodes as they emphasize. Also they support that in a single wireless sensor network, sensors are performing two operations, sensing and communication a solution is to let sensors work alternatively by identifying redundant nodes in high density networks and assigning them an off-duty operation mode that has lower energy consumption than the normal on-duty mode. They support that they provide the conclusion that the communication range is twice of the sensing range which is the sufficient condition and the tight lower bound to ensure that complete coverage preservation implies connectivity among active nodes if the original network topology, consisting of all the deployed nodes, is connected.

In their paper they firstly describe their communication model and they characterize the wireless sensor network as a communication graph. Then they introduce the concept of sensing coverage into a network graph G. About this sensing model they support that they define A as the region where sensors are initially deployed in and are supposed to monitor afterwards and they add that each sensor can do 360 observation. In the end they denote the maximal

circular area centered at a sensor that can be covered by that sensor as its sensing area *S(v)*.

After all these they support the general networks that may have small sensing holes geographically scattered throughout the deployed area. They argue that the latter is more realistic situation because wireless sensor networks use a random node deployment strategy and such a strategy implies that it is difficult or impossible to completely ensure that each point in the region A can be covered by at least one sensor node even at a very high node density. Although they add that it is possible that after removing some sensing redundant nodes, the remaining nodes can cover the same area as initially, without introducing any coverage loss.

In the end they prove that $R \geq 2r$ is the sufficient condition and the tight lower bound to ensure connectivity of any sub-graph G (V_{active}) induced from a connected network graph *G (V, E)* since they have shown that above, some existing node scheduling algorithms have the capability of complete coverage preservation. To do this they make two assumption that the first say that there may be a large number of sensor nodes in the network. However, the number is always finite and the second support that there are no two nodes located at the exactly same position. Then they make the proofs and some experiments in existing algorithm.

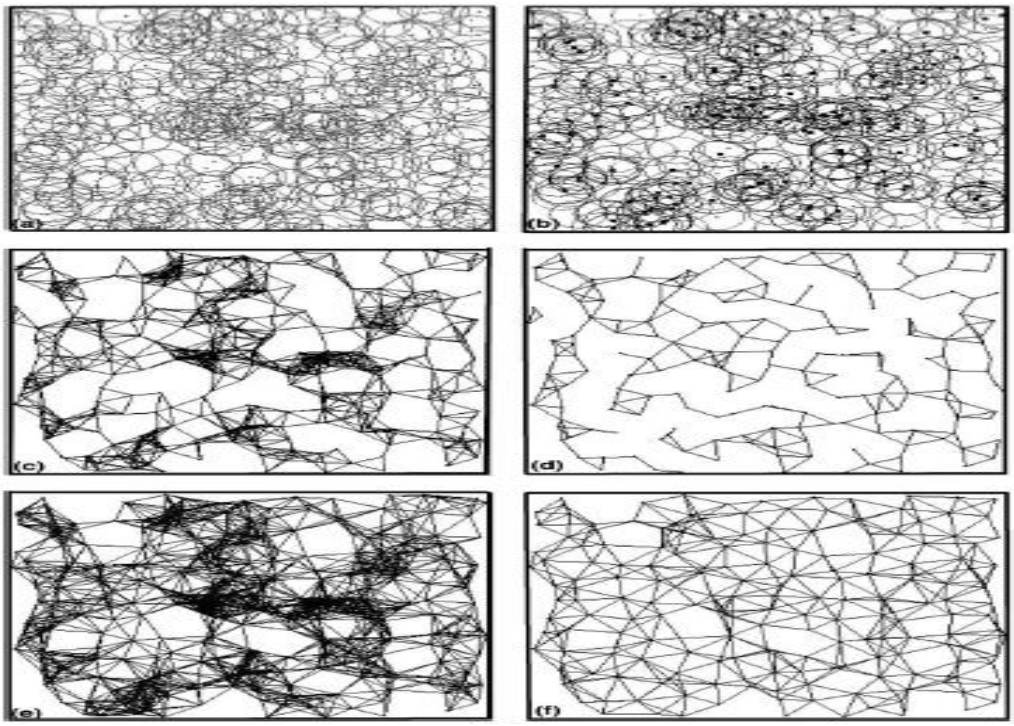


Figure 3.2:(a) Overall sensing coverage of a network with 350 nodes in a 100X100m^2 region, (b) scheduling result with r as 6 m (black circles representing inactive nodes), (c) network is connected before node scheduling with R = 9 m, (d) active nodes are disconnected with R = 9 m, (e) network is connected before node scheduling with R = 12 m and (f) active nodes are disconnected with R = 12 m.

In [36] we can read another scientific research that is most closely related to ours. In this work the authors consider power efficient organization of sensor networks. Through their scientific proposal they introduce a heuristic

that selects mutually exclusive sets of sensors, the members of each of those sets together completely cover the monitored/query area. Their technique results in energy savings while preserving coverage, because they describe that only one set of sensors need to be active at any time. They add that every set is activated when it passes a certain time and while the first one is deactivated. Furthermore they mention that the whole process repeats until the sensors are out of power and this means that the one round is resulted and all sets are used. The algorithm that they create and present in their work does not extend easily to a distributed algorithm. Furthermore, it is important to be noted that they do not consider the communication cost incurred in the execution of their heuristic and hence they do not require the selected sets to be connected.

To support the reason that they write this paper, they analyze their motivation by comparing the controlled node placement with the random node placement. They focus on two main reasons to explain the reason that the deterministic placement of sensor nodes is impractical. The first has to do with the deployment of the sensor and with the fact that take place in remote or inhospitable areas, which prevent individual sensor node deployment. Another fact that comes to reaching that their reasoning is that the sensor nodes are not capable of dynamic adjustment of their positions. The other reason that they describe in their paper is dealing with the number of sensor nodes. Because of this number is large in a network, the cost of a deterministic placement is increased and latency in the deployment of the network. So, they underline about the preferred method of sensor placement is that bulk dispersion of sensor nodes from an aircraft.

More especially, the generalized goal of the algorithm that they present is to maximize the number of the sets. Their algorithm is composed by two phases. The determination of how many different sensor nodes cover the different parts of the monitored area by the algorithm considered as the first phase of the algorithm. About the second phase the authors describe that includes the allocation of sensor nodes into mutually independent sets. The assumption regarding the purpose of the simulation according to the authors is that the sensing area of a sensor is a circle with the radius of the circle equal to the sensing range of a sensor but not required something like this by the heuristic algorithm as the authors underline. Another assumption for each sensor in which the algorithm is based on is that the determination of the sensing range of any shape can take place either before the deployment as a static approximation of the sensor's capabilities or as a function of a specific location and surrounding environment after the deployment of a sensor node. Also they finish to that the observed target can affects the sensing range.

They end up making a report of the simulation results and emphasize on that the conclusion is that the determination of the possible utilization of the

available sensor nodes is made from the initial distribution of the sensor nodes in the monitored area.

Another paper that includes a closely related work to ours is that in [37] in which its authors consider an unreliable sensor grid-network and describe that the main features for the definition of the sufficient conditions for the coverage of the region and connectivity of the network are the transmission radius, sensing radius, and failure rate of the sensors.

The unreliability of the sensor networks is the element that made the authors to deal with such an issue. They describe that the network that they use in the context of simulations and approach the problem consists of n nodes arranged in such a manner so as to create a grid over a square region of unit area, as shown in Figure 3.3.

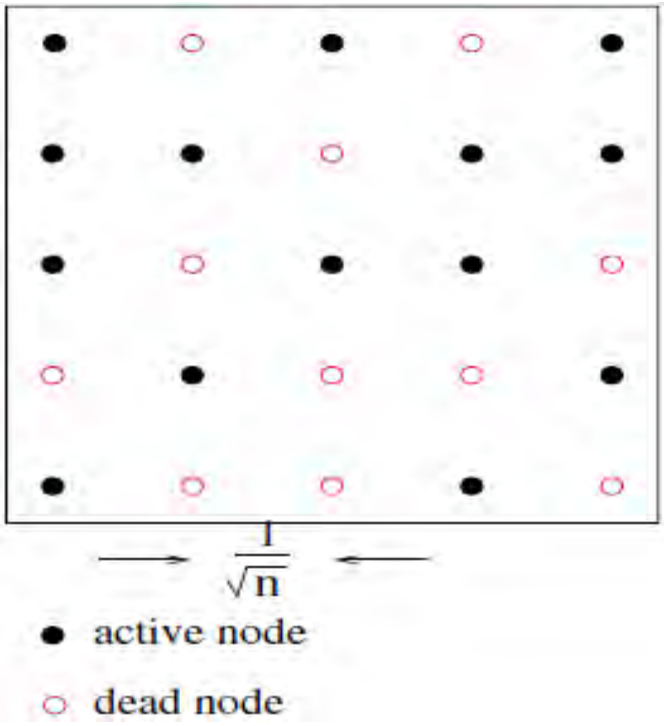


Figure 3.3: A grid network with unreliable nodes

As they describe the separation that they use in their example between adjacent nodes is $1/\sqrt{n}$ units. They give a definition about the sensing radius saying that because of every node is a sensor can detect events within some distance from it. They add that every sensor has a probability $p(n)$ and is active but because of the nodes are prone in failure and for that reason everyone has an independent fail probability of $1 - p(n)$. Also they mention that in the case of the distance between a pair of nodes is less than some specified value, this pair can communicate with each other. Furthermore an assumption that they make in their paper is that they consider that sensing radius is the same as the transmission radius. They support that the terminus

seeing final results of the simulations made is that if the sensing radius is different form the transmission radius their results can easily be extended to that case. Also they develop scaling laws for the quantities $p(n)$ that we describe above and $r(n)$ which is the transmission or the sensing radius of a node and parallel they demonstrate the dependence of these two quantities.

Some other important things that they support in their work are that connectivity and coverage are two important performance metrics for such a network. In the case that an active node can communicate with any other active node they say that the network is connected. To understand better the readers the importance of these two performance metrics, they say that in an intrusion detection system the coverage ensures that the intrusion is detected while the connectivity ensures that messages are propagated to the appropriate authority. In the end they prove that in the case that the node success probability $p(n)$ is small enough the connectivity does not imply coverage.

Something very different to our work is described in [38]. In this paper the authors present and analyze the coverage problems to maximize and improve the surveillance that is provided by a sensor network. More especially, about finding the lowest and highest observabilities in a sensor network, they study the problem of finding maximal paths for these qualities. They support that as a definition of the coverage can be considered the measure of quality of service of a sensor network. Additionally, suggestions about future deployment or reconfiguration schemes for improving the overall quality of service can be created from the coverage formulations which can try to find weak points.

In continue they mention that in the most sensor networks there are two seemingly contradictory viewpoints of coverage and that are the worst and the best case coverage. The primary concern is the finding of areas of high observability and identifying the best support and guidance regions from sensors. From the other hand, about the worst case coverage, they mention that are made efforts to quantify the quality of service by finding areas of lower observability from sensor nodes and detecting breach regions. For that reasons they aimed to create robust, efficient and scalable algorithms to be used in wireless multi-sensor integration.

Because of, from another point of view, the algorithmic, the main contribution is provably optimal polynomial time algorithm for coverage in sensor networks the authors add that they use a combination of the existing computational geometry techniques and constructs such as the Voronoi diagram as we can see in Figure 3.4 which its use transforms the continuous geometric problem into a discrete graph problem and also it enables direct application of search techniques in the resulting graph representation, with graph theoretical algorithmic techniques.

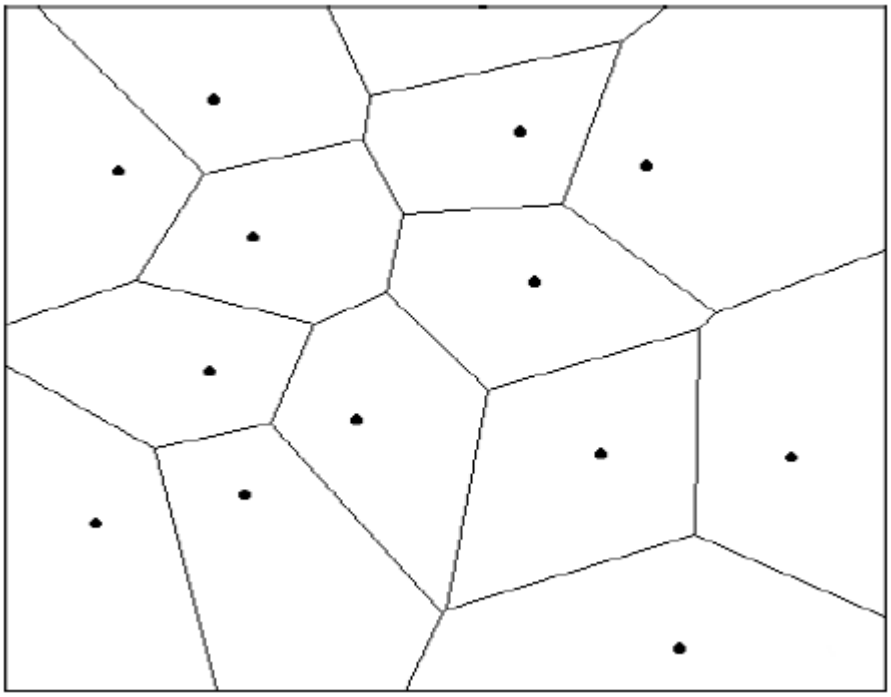


Figure 3.4: Voronoi Diagram of a Set of Randomly Placed Points in a Plane

Moreover, in their simulations they make another assumption. Because of sensing ability is dependent on distance and that is a fact that is common in most models they assume that if distance from sensors increases, the sensor coverage decreases.

Furthermore, they describe that they make an implementation concerning the location procedure as the initial step before the coverage algorithm because their coverage algorithms based on geolocation information. They mention on their geolocation approach that they call beacons a few of the sensor nodes which they know their coordinates in advance, either from satellite information (GPS) or pre-deployment. To approximate the neighbor distances, the geolocation scheme relies on signal strength information embedded in the inherent radio frequency communication capabilities of the nodes. Also they add that for becoming a beacon, a node firstly can hear from a minimum of three beacon neighbors and also can determine its own location by trilateration. For the location of as many possible nodes are used iterative trilaterations.

In the end they support that the final simulations show that in a reasonably dense network, by having 1% or less of the nodes as initial beacons, almost all other nodes can locate themselves at the end of the location process and also they mention that they only use nodes that have valid location information.

Finally a work that does not deal at all with the concepts of spatial coverage and connectivity is that is described in [39]. In this paper, the authors suggest a randomized clustering algorithm for selecting cluster heads in a sensor network resulting that the energy budget for relaying information to a gateway

is distributed evenly among sensor nodes. They support that because of there is a transmission for the data being sensed by the nodes in the network to a control center or base station, where the end-user can access the data, there are many possible models for these microsensor networks. In their work they use microsensor networks which have two main characteristics. Firstly, they describe that the base station is fixed and located far from the sensors and secondly all nodes in the network are homogeneous and energy constrained.

More especially, their work focuses on the framework of the expensive communication between the sensor nodes and the base station and in the fact that there are no high-energy nodes through which communication can proceed. The purpose of this framework is to suggest energy-optimized solutions at all levels of the system hierarchy, from the physical layer and communication protocols up to the application layer and efficient DSP design for microsensor nodes.

For characterizing the methods of performing data aggregation and the classification algorithm they say that are application-specific. They add that they can achieve large energy gains by performing the data fusion or classification algorithm locally. As a result of these gains is required much less data to be transmitted to the base station.

They mention that they developed LEACH (Low-Energy Adaptive Clustering Hierarchy), a clustering-based protocol that minimizes energy dissipation in sensor networks from the resolution of the advantages and disadvantages of conventional routing protocols. They underline the three main characteristics of this protocol which are firstly to reduce global communication it uses local compression, secondly for the cluster base stations and cluster-heads and the corresponding clusters is used a randomized rotation and finally for cluster set-up and operation is used localized coordination and control.

Is noteworthy that they mention that because of the LEACH allows the energy requirements of the system to be distributed among all the sensors, this protocol outperforms classical clustering algorithms by using adaptive clusters and rotating cluster-heads. Furthermore they add that there is reduction of the amount of data that must be transmitted to the base station from the performance of local computation in each cluster from this protocol. As a result of all those mentioned above achieved a reduction in the energy dissipation because the computation is much cheaper than communication.

4. Problem Formulation and Methodology

In this chapter we describe the general problem that we want to solve in our work, presenting motivations, and give a formal definition of the problem. Firstly, we will give a description of a sensor network model.

A wireless sensor network consists of a large number of sensors, deployed in a region of interest. Each device has a certain amount of processing and wireless communication capabilities, which enable it to gather information about the environment and to generate and deliver report messages to the remote base station which is controlled by a remote user. More specifically, each sensor has a unique identifier (ID) and is capable of sensing a well-defined convex region S around itself called the sensing region. We will describe more the sensing region in the next subsection. Every sensor can communicate directly with some of the sensors around it, because each sensor has also a radio interface. Also, in wireless sensor networks, the energy source provided for sensors is usually through batteries, which of course have a limited lifetime in the absence of recharging. Furthermore, due to the limited communication capability, each sensor node has to act as router to help others to forward data packets to remote base stations. That means that the routing of communication packets is multi-hoped.

Our work tries to face trying to address the problem the following optimization problem that exists in wireless sensor networks: having a set of sensors with different covering abilities (range), we need to place them in an area so that it will be efficiently covered.

4.1 Incentives

For the following two characteristics resulting from the sensor networks get the motivation to deal with this problem:

- ***Sensing Region:*** With the concept sensing region in a sensor we define the area that the sensor can take the full responsibility for sensing a given physical phenomenon within a desired confidence. So we can easily understand that to being fully exploit this criterion by placing the sensors is very important in order to save as many sensors can.

- ***Limited Battery Power:*** Every sensor have a limited battery power because of the fact that is a very small computing device. The energy budget for communication is many times more than computation with

the available technology as is described in [40]. So minimizing communication cost is very important because with that way will be created bigger lifetime sensor networks.

4.2 Definition of the problem

Now, in this section we define the problem that we have to address in wireless sensor networks.

Given a set of sensors, select a minimum number of sensors, such that: throughout the geographical area to be covered from the sensing regions of the selected number of sensors. Furthermore for that placed sensors and generally for that wireless network we want to extend the network lifetime achieving lower communication costs.

Generally this is a NP-hard problem and for solving this we break it into two pieces and we dealt with each one separately.

4.3 Description of Sensing Regions

As we already said in the previous subsection, the sensing region in a sensor is an area for which the sensor can take the full responsibility for sensing a given physical phenomenon within a desired confidence. The real definition of a sensing region in an application is specific. To explain this we will use an example. So, for example in detection applications, when we use the term sensing region we mean that the sensor have a region around him which named sensing region and can detect a target with a pre-determined minimum confidence on this region. So we understand that in this case the sensing region for a sensor is a region with a fixed distance inside which can detect everything the sensor. In some other applications the sensing regions are defined as the correlation of the sensed data. And in this case we will use an example to make it clearer. The applications that deal with temperature measurements in a geographical region and are monitored by a sensor network, many times the temperature values at any two points can be highly correlated. In these cases we can define sensing regions of circular radius d around each sensor.

The meaning of sensing region is used in [36, 37] which are works similar to our and their authors support that the sensing region for each sensor is used as a static approximation of the sensor's location and capabilities.

In our work the sensing regions can take any shape and in the created algorithm that is predetermined.

4.4 Methodology

As we already said for solving this problem of the placement of sensors in any area with as few as possible sensors employed we break it into two pieces and we dealt with each one separately.

Firstly, we create a placement algorithm to solve the problem of that having a set of sensors with different covering abilities (range) we try to place them in an area so that it will be efficiently covered. The algorithm selects the sensor with the largest range and places it at or near the center of the area. As long as there is uncovered area, he places a new sensor in the area. The point, where the sensor will be placed, has to be inside the range of one of the placed sensors and the sensor will cover larger uncovered area. So to create this algorithm we based on the covering abilities of every sensor to cover with as few as possible sensors area we want. In this algorithm are not taken into account the sensors communication costs. Also the geometrical calculations are out of the scope of this algorithm and haven't been applied.

The parameter of communication costs is taken into account in the next algorithm that we created with the help of the Omnet++ simulation tool in C++ programming language. For creating this algorithm we based on two parameters to ensure that the purpose of this algorithm is to maximize the network lifetime. Firstly, we define the weight that every connection between sensors has and is an integer number representing the power that a node needs to consume in order to transmit the packet through this connection. It is obvious that the greater the weight of a connection between two sensors both reduced lifetime of the network due to the fact that will be consumed by the sensors is greater energy to send packets to other nodes. The other parameter is the queue that has every sensor and more especially the queue threshold of every sensor minimum packages must have the tail of a sensor to start sending packets. To how big the threshold depends on the network you want to implement the algorithm. The values of threshold should not be very large because it may delay the packets to reach the final recipient and so have reached the first packages that were created later and thus created some confusing information. Furthermore, the algorithm is based on the tail of every sensor and its threshold and for another reason. If you had any tail pack and will come directly evict thus consumes energy then until the end of the network which would be short because of minimizing the lifetime of the network. Conversely using these mentioned above keeps sending packets gathered with less energy.

5. The Placement Algorithm

In this part of our work we will describe the algorithm that we create to help us with the sensor placement in any area that we want to use. So, generally the problem that we want to solve is that having a set of sensors with different covering abilities (range), we need to place them in an area so that it will be efficiently covered.

A briefly description that we can give is that we select the sensor with the largest range and we place it at or near the center of the area. As long as there is uncovered area, we place a new sensor in the area. The point, where the sensor will be placed, has to be inside the range of one of the placed sensors and the sensor will cover larger uncovered area.

5.1 A small description

As we say need to place in an area a set of sensors with different covering abilities, namely range. Also, as we have already pointed out that generally every time the algorithm selects the sensor with the largest range and we place it at or near the center of the area and it is making that action as long as there is uncovered area.

But before describing the algorithm created in more detail we would like to mention two assumptions did the writing and completion of this algorithm. Firstly we have to underline that the algorithm doesn't take into account the sensors communication costs. This very important admission that we want to maximize the battery life of the sensors network is described in the next chapter that we look at creating and using another algorithm. Our only expectation from this algorithm is to find the better positions in the area that we want to put the sensors for covering this with as few as possible sensors area we want. Furthermore another assumption that we make is as follows. The implementation of the algorithm depends on the actual shape and geometry of the area to be covered by the sensors. The geometrical calculations are out of the scope of this algorithm and haven't been applied.

Now we will give a more detailed description about the placement algorithm and the fact that how he puts sensors in every area that we want. Firstly, we define the following data structures and object classes:

- ***Point:*** It represents a point with 2 integer coordinates.
- ***Sensor:*** The sensor consists of a point, where it is placed, and the range it covers.
- ***Area:*** An Area object is defined by a set of points. These points define the

boundaries of the area.

In order to apply the algorithm, the input data are an Area object and a set of sensors, which will be placed in the area. The expected output is a set of sensors with complete information of their placements.

The algorithm starts by placing the first sensor at or near the center of the area. The steps to do so are:

- Select the sensor with the largest range from the input set of sensors.
- Calculate the center of the area [(maximum x coordinate – minimum x coordinate)/2 (maximum y coordinate – minimum y coordinate)/2]
- Set the above point to the selected sensor
- Set the sensor to the output set of sensors
- Remove the sensor from the input set of sensors.

As long as there is uncovered area by sensors and there are more sensors in the input set, the algorithm executes the following:

- Select the sensor with the largest range from the input set of sensors.
- Calculate the variable benefit for each point inside the range of the placed sensors. The variable benefit is assigned with a double value representing the uncovered surface of the area which will be covered if we place the selected sensor at the specific point.
- If the benefit is larger than the value of the variable max_benefit, we assign the value of benefit to the max_benefit and the point coordinates at the xBeneficial and yBeneficial variables.
- Set the point, derived by xBeneficial and yBeneficial, to the selected sensor.
- Set the sensor to the output set of sensors.
- Remove the sensor from the input set of sensors.

5.2 The main code

In this section of our work we represent the main code of our placement algorithm. We quote the infrastructure with the states that are made, the main algorithm and the required functions. We aim through this algorithm we can cover any area we want to request or as few sensors.

Infrastructure

```
struct Point{
        int xCoordinate
        int yCoordinate
}

struct Sensor{
        Point placement
        int range
}

class Area{
        Point[] points //array of significant points defining the area

        int maxX(){
                //Returns the maximum x coordinate of the area
                int x=points[0].xCoordinate;
                for p=1 to points.length(){
                        if points[p]. xCoordinate>x
                                x=points[p]. xCoordinate;
                }
```

```
        return x;
}

int minX(){
        //Returns the minimum x coordinate of the area
        int x=points[0].xCoordinate;
        for p=1 to points.length(){
                if points[p]. xCoordinate<x
                        x=points[p]. xCoordinate;
        }
        return x;
}

int maxY(){
        //Returns the maximum y coordinate of the area
        int y=points[0].yCoordinate;
        for p=1 to points.length(){
                if points[p]. yCoordinate>y
                        y=points[p]. yCoordinate;
        }
        return y;
}

int minY(){
        //Returns the minimum y coordinate of the area
```

```
            int y=points[0].yCoordinate;
            for p=1 to points.length(){
                if points[p]. yCoordinate<y
                    y=points[p]. yCoordinate;
            }
            return y;
        }
}
```

Algorithm

Input: Area area

Sensor[] sensorsToPlace; //array of available sensors

Output: Sensor[] placedSensors //array of sensors placed in the area

BEGIN

int sensorNo=0;

//Place 1st sensor on center of the area

Point center=new Point(area.maxX()-area.minX(), area.maxY()-area.minY());

placedSensors[sensorNo]=largestRangeSensor();

sensorsToPlace.remove(largestRangeSensor());

placedSensors[sensorNo].placement=center;

```
while (uncoveredAreaExists() && sensorsToPlace.length()>0){
        //Find the most beneficial sensor placement
        sensorNo++;
        double benefit=0.0;
        double max_benefit=0.0;
        int xBeneficial=0;
        int yBeneficial=0;

        for i=0 to placedSensors.length(){
                for x=placedSensors[i].placement.xCoordinate-range to
placedSensors[i].placement.xCoordinate+range{
                        for y=placedSensors[i].placement.yCoordinate-range to
placedSensors[i].placement.yCoordinate+range{
                                benefit=calculateBenefit(new Point(x,y),
largestRangeSensor().range);
                                if benefit>max_benefit{
                                        max_benefit=benefit;
                                        xBeneficial=x;
                                        yBeneficial=y;
                                }
                        }
                }
        }

        placedSensors[sensorNo]=largestRangeSensor();
        sensorsToPlace.remove(largestRangeSensor());
        placedSensors[sensorNo].placement=new Point(xBeneficial,yBeneficial);
```

```
}

END
```

Required Functions

```
boolean uncoveredAreaExists(){
        //Returns true if there is area uncovered by sensors.
        //Implementation depends on the geometry of the area
}

double calculateBenefit(Point point, int range){
        //Returns the area which will be covered if we place a sensor at a given point.
        //Maximum return value is PI*range*range;
        //Implementation depends on the geometry of the area
}

Sensor largestRangeSensor(){
        //Returns the first available sensor with the largest range
        int sensorToPlace=0;
        for i=1 to sensorsToPlace.length(){
                if  sensorsToPlace[sensorToPlace].range< sensorsToPlace[i].range
                        sensorToPlace=i;
        }
        return sensorsToPlace[sensorToPlace];
}
```

6. The ComCost Algorithm

In this part of our work we will describe another algorithm that we use and create in programming language C++ with the help of a programming environment that is named Omnet++ for taking some simulation results. Because of in the previous chapter we described an algorithm that ensures to us the most informative positions in any area that we want to put sensors, now we make some simulations using another algorithm that we present this in C++ considering that we have taken these informative positions from the previous algorithm.

6.1 Network model

In this section we mention the model that we use to represent a wireless sensor network and we will present the algorithm implemented in this model with the objective of maximizing the lifetime of the network. We consider a wireless sensor network consisting of nodes which gather information from their surroundings and which after being placed in an area that is not moving, is namely static. These nodes are equipped with a battery of finite energy E which is the same for all nodes. The time to the moment when for the first time will run out of energy E of one of the nodes is defined as the lifetime of the network. Information which is gathered has as single destination another node that is part of the network, called node collector. This node, which we denote by *s*, does not move in our model and collect all the information which comes to him in order to process them further if it has more processing power than ordinary nodes. Alternatively, the information is transmitted to a wider network which will make proper use of a remote user. The information is transmitted and reach the collector node with packet format directly, that we define as single hops of the node that is collected or multiple retransmissions which we name multiple hops through other nodes.

Our network is therefore represented by a graph $G = (N, C)$(Figure 6.1) where N represents the number of nodes and C represents the connections between them respectively. A connection between a node a and a node b is characterized by the edge (a, b) where $a, b \in N$. Note that the edge (a, b) is different from the edge (b, a), that the graph is directed and that the topology of network we study is such as to permit the existence of only one edge of each two adjacent nodes.

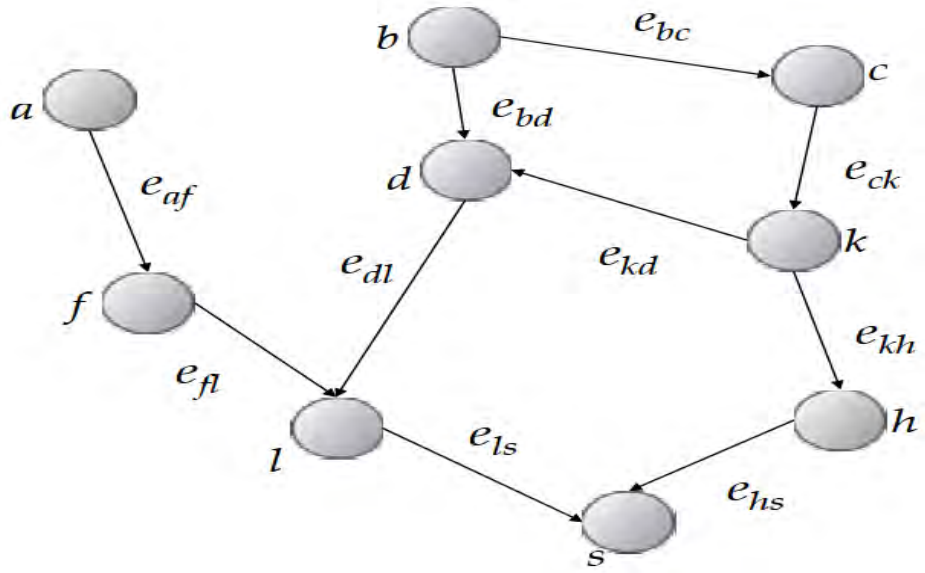


Figure 6.1: Network representation

Also, as the authors in their work in [41] support, the model assumes that the network operates in discrete timeslots. This means that the decisions taken by the algorithm are valid for the entire duration of the time slot while slot to slot these decisions change depending on whether the algorithm works dynamically or static. So we can say that all nodes operate with a common clock. Also, in our model we consider that we haven't got changes in the transmission channels.

Furthermore, we define that our model packets entering through the network layer and placed in a queue that has each node which stores both the packets from the same node and the application layer and the packets arriving at the node from neighboring nodes and also does not have a specific limit on the number of packets that can be saved so you may need to discard incoming packets, for the reason that we support that there is no flow control.

6.2 Omnet++

The Omnet++ [42] is a discrete event simulation system that is creating networks, based on C++, so is suitable for the network that we want to simulate since it works in slots. It follows the logic of creating autonomous modules (modular approach) for representation of network components, and the internal foundation within them all parameters and methods that implement the simulation. The Omnet++ is not limited to a type / model network, but is able to simulate all types of networks, wired / wireless

communication networks, transport networks, routers, token rings etc. Also this simulation program contains a wide library of object classes in C++, which represent the different elements of a discrete-time system. Furthermore each class has a number of methods that give substantial functionality to the objects thereof.

Now we will give a further description of the architecture of Omnet. In Omnet + + simulators consider various models. These models are referred to as networks (networks). Each model-network consists of hierarchically structured modules (modules). The top unit in the hierarchy is the system unit and is equivalent to the network simulating as an entity object. It contains other subsections, the so-called modules (submodules), which may also in turn contain other subunits. There is no limit to the range of the division of a unit.

The structure of a model is described by the NED's programming language Omnet++. The units containing other subunits called compound units or compound modules. The units located in the lower level of the hierarchy are called single units or simple modules. The single units are those containing the algorithms of the model and determine the behavior and operation of the simulator. The simple modules implemented in C++.

Both simple and compound units are incarnation's module types. The user when designing the model defines types of units. Embodiments of these are parts of more complex types of units, and ultimately the user designs the network / system as a type of unit to which all other types of incarnating as subunits.

When a type of unit is used in building the network, not considering whether it is simple or complex, and not treated differently in each case. That is the same way of using a composite type using a single unit type. This allows you to split a simple module into several smaller units that are simple to the longer form as a complex unit so we can organize our system more efficient in small portions with individual character. Of course it is possible and vice versa, i.e. to join some simple units form a complex in a simple to implement the same functions as the first embodiment, and without affecting the management unit of system.

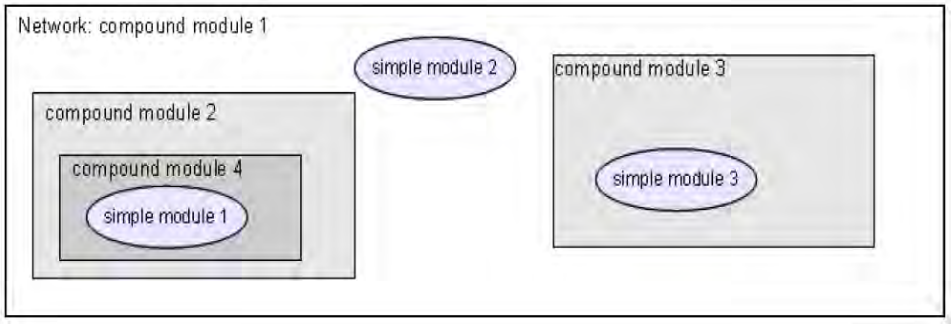


Figure 6.2: A hierarchical model-network

6.3 ComCost Algorithm Implementation

In this section of our work we analyze the algorithm that we create on Omnet to maximize the network lifetime, when the first algorithm give to us the most informative positions on any area we want to place for the sensors. Generally, in our model we create two new classes: the class Node and the class Sink which their code we can see in Appendix. The first defines the function of simple nodes having them performed essentially the algorithm and the second function of the collector node. The collector node all you do is deleting the network packets that arrive there and count the total number of packets that have come.

For creating this algorithm in the C++ programming language we based on two parameters to ensure that the purpose of this algorithm is to maximize the network lifetime. Firstly, we define the weight that every connection between sensors has and is an integer number representing the power that a node needs to consume in order to transmit the packet through this connection. It is obvious that the greater the weight of a connection between two sensors both reduced lifetime of the network due to the fact that will be consumed by the sensors is greater energy to send packets to other nodes. The other parameter is the queue that has every sensor and more especially the queue threshold of every sensor minimum packages must have the tail of a sensor to start sending packets. To how big the threshold depends on the network you want to implement the algorithm. The values of threshold should not be very large because it may delay the packets to reach the final recipient and so have reached the first packages that were created later and thus created some confusing information. Furthermore, the algorithm is based on the tail of every sensor and its threshold and for another reason. If you had any tail pack and will come directly evict thus consumes energy then until the end of the network which would be short because of minimizing the lifetime of the network. Conversely using these mentioned above keeps sending packets gathered with less energy.

6.3.1 Input

ComCost Algorithm is applied on a network consisting of sensors (nodes) and one sink. The sensors produce information packets, which need to be transmitted to the sink (final destination). Each node is connected to other nodes, while some of them are connected to the sink. Each connection has a weight, an integer number representing the power that a node needs to consume in order to transmit the packet through this connection as we said in the previous subchapter. Weight values may depend on the distance between the nodes, noise of the physical links, errors during transmission, etc. Also, each node maintains a queue, where the packets are placed and wait their turn to proceed through a connection to another node or the sink.

In order to use Omnet++ to simulate the network, we have created 3 Network Description files (Node.ned, Sink.ned and ComCostNetwork.ned) and one simulation configuration file (omnetpp.ini). Finally, we have implemented the Node and Sink classes in C++, header and class code files for each.

The Sink.ned file contains only the declaration for the sink input gates vector.

The Node.ned file includes declarations for the following:

- Input gates vector.
- Output gates vector.
- Parameters, representing Node features, such as total Energy and maximum power per connection.

The ComCostNetwork.ned file describes the network's topology (connections among nodes) while in the network configuration file (omnetpp.ini) the connections' weights are assigned.

Finally, the classes implement the way the nodes and the sink handle the received packets. While the sink simply discards the packets it receives, the Node class handles the packets, according to the algorithm implementation.

6.3.2 Assumptions

For our simplified things, we use three assumptions in our simulations. Firstly, Information packets are transmitted between 2 nodes only in one direction. Also a node can send packets to up to 4 other nodes (including the sink, if there is a connection) while there is no restriction on the number of

nodes it receives packets from. And finally, a connection weight can have one of the values: 1,2,3,4 or 6.

6.3.3 The main description of the algorithm

As mentioned above, the implementation of the algorithm is done in the Node class. Specifically, the algorithm is applied in the handleMessage (cMessage *msg) method of the Node class.

This method handles messages that the node receives. The messages are handled in specific time intervals. For simulation purposes, the time intervals are set in 2 seconds (simulation time). There are 3 types of messages that a node has to handle:

- Packet From Sensor Received. This message is received by the Node object when information packets are created by its sensor. Those packets are placed in the queue.

- Find Optimal Routing. When a node receives this message it calculates the optimal routing for sending packets.

- Send Packets. When a node receives this kind of message, it sends a number of packets through the connection with the optimal weight. There is a possibility that an optimal route doesn't exist, so the node keeps the packet in its queue.

  Simulation starts with the creation of a random number of sensor packets. These packets are placed in the queues. Whenever the sensors create packets, their number is random generated by Poisson distribution.

  After the packets are placed in the queue, the node schedules a message “Find Optimal Routing” to be received in the next time interval.

  When the above message is received, the node evaluates the optimal routing for sending out the packets of its queue. In order to do that, it calculates a number of variables for each connection:

- Difference of its queue length and the connected node's queue length. If the length of its queue is larger, it stores the difference in the variable optimalRoute. If not, zero value is assigned to the variable.

- If the node is connected to the sink, then the variable optimalRoute is assigned with the value of the node's queue length.

- Then, the variable routeCriterion is assigned with the value produced by the

formula: optimalRoute/connection_weight - queueThreshold. Connection_weight is the weight value of the specific route and queueThreshold is a constant variable set in the Node.ned file, representing how many packets can be kept and wait in the queue. For the specific simulation, queueThreshold has a value of 20.

After the above calculations, the node schedules a message "Send Packets" to be received in the next time interval.

When the above message is received, the node checks the value of routeCriterion for each connection. If the value is positive then it sends packets through the connection. The number of packets is calculated by the formula: maximum_power/connection_weight. Maximum_power is a value derived by the Node.ned file. For this simulation the value is set to 12.

If the number of packets is larger than the number of the packets in the queue, the node sends the packets of the queue and creates and sends the appropriate number of blank packets.

After sending the packets, the node calculates the energy that has consumed and the remaining energy it has. If the remaining energy is less or equal to zero then is the network considered dead and the simulation stops.

If not, the node schedules a message "Packet From Sensor Received" to be received in the next time interval and the whole procedure restarts. When this message is received the sensors create a random number of packets for each node.

When a node receives a packet, it checks if it is a blank one or one created by a sensor. If it is blank it discards it. If not, it places it in the queue.

6.4 Simulation Tests And Results

The algorithm tries to move as more packets as possible from the sensors to the sink and in the same time to extend the life of the network by minimizing the consumed power of the nodes.

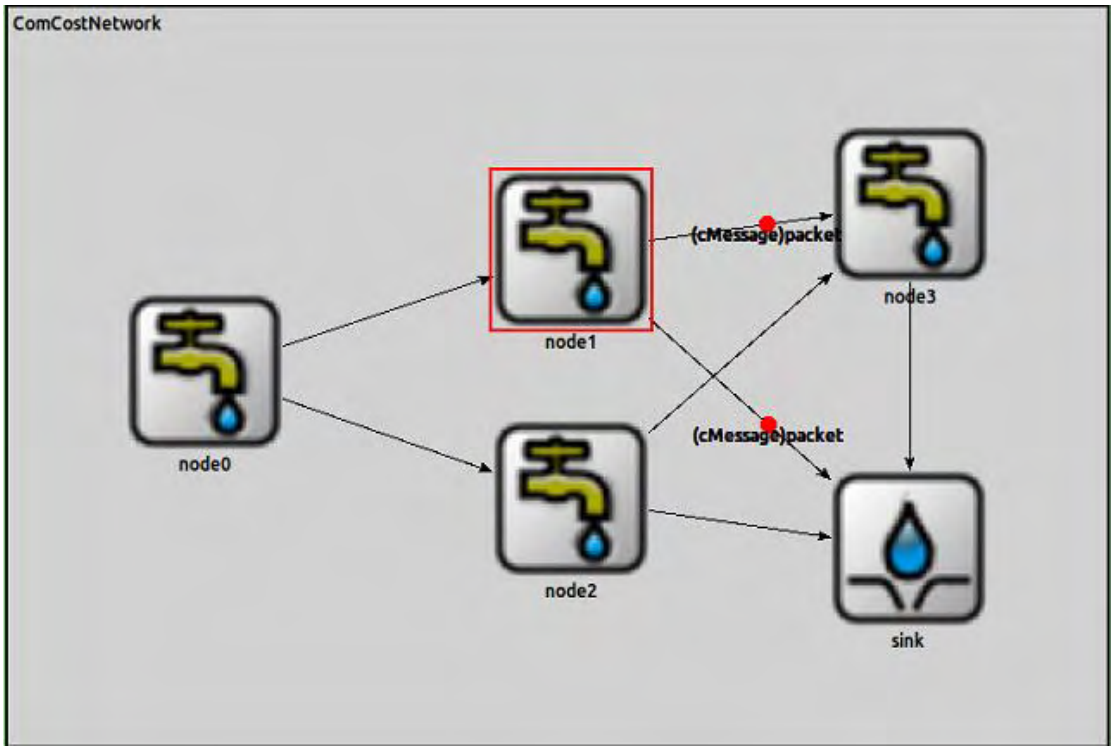


Figure 6.3: The ComCost network in Omnet

In order to succeed it, each node has a queue, where it places the incoming packets. Main functionality is that the node doesn't send all the packets of the queue in each time interval. The number of the packets that a node will send depends on the weight – cost of the connection, the power needed to send a packet through a connection and a queue threshold that we define. The queue threshold variable represents the least number of packets that a queue should hold, before the node starts sending them through a connection. For example if we assign the value 10 to the variable queue threshold, each node will not use a connection until the packets of its queue are more than 10.

It is obvious that the value of the queue threshold impacts on the algorithm's behavior and efficiency. The appropriate value is network-oriented, which means that it depends on the network's topology (number of nodes, connections, connections' costs, etc.).

In order to specify the appropriate queue threshold, the behavior of a specific network was tested in different threshold values. The network used for testing is the following:

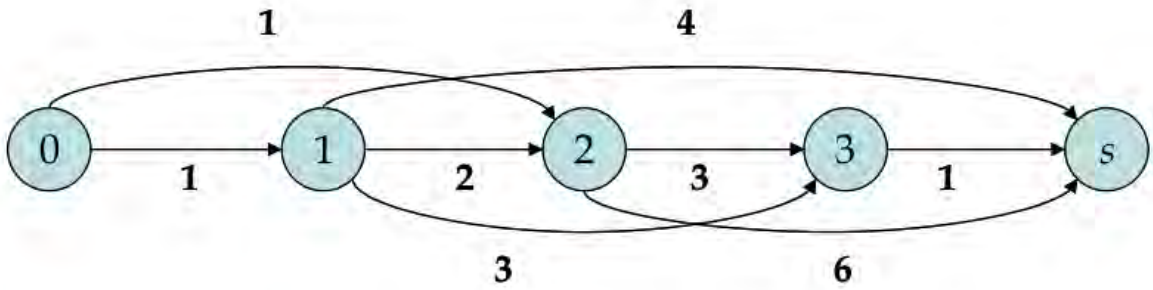


Figure 6.4: The ComCost network

Furthermore, in order to better understand the behavior of the algorithm in different values of queue thresholds and the communication costs, respective tests were done in the following networks:

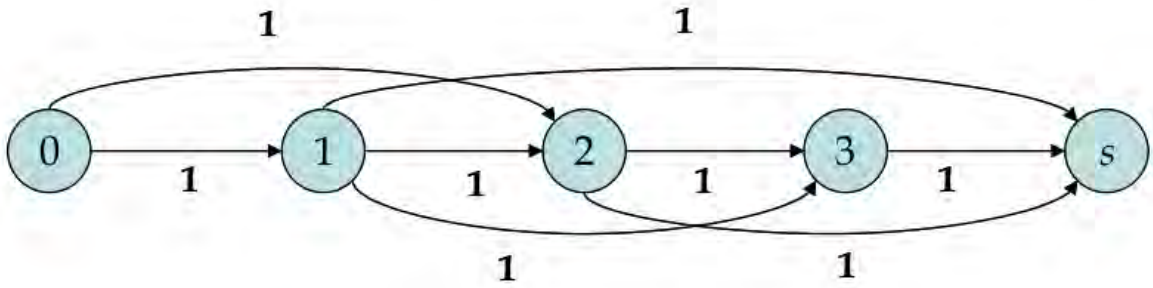


Figure 6.5: The Simple ComCost network

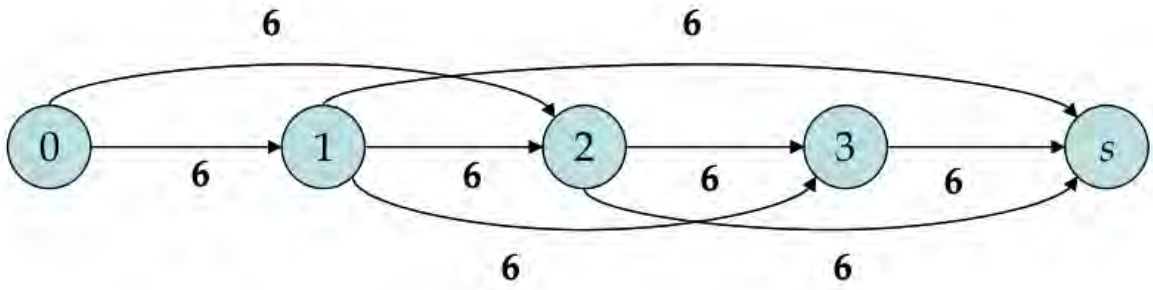


Figure 6.6: The Heavy ComCost network

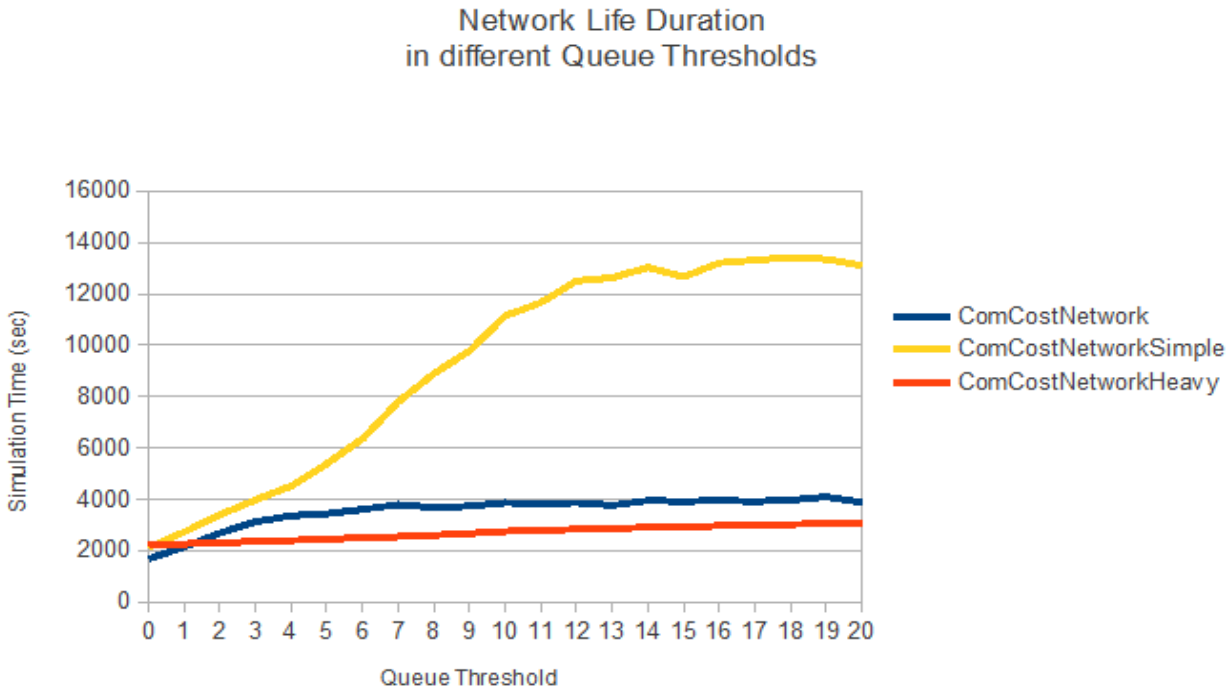
The above networks have the same topology (number of nodes, connections) with the tested network. Their difference is the communication costs of their connections. As it can be seen in the above pictures, the first

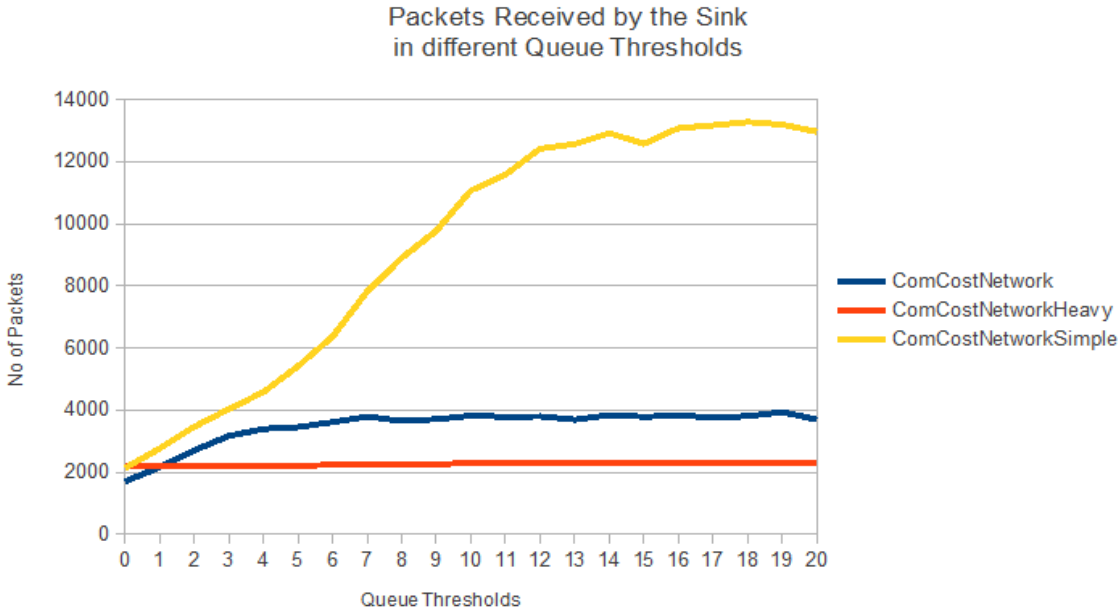
network has no communication costs on its connections (ComCostNetworkSimple) while the other one has the maximum communication costs on them (ComCostNetworkHeavy).

All the above networks were tested by Omnet++ simulator. Tests were done using queue threshold values from 0 to 20. It was observed that queue threshold values more than 20 have not further significant impact on the algorithm's behavior while there are packets received by the sink with large delay, especially those originated by nodes far away from the sink, due to their excessive waiting time at the nodes' queues.

Tests results were based on the life time duration of the networks and the number of information packets received by the sink.

The respective results of the tests can be seen in the following diagrams:





The diagrams show that the algorithm improves lifetime duration and the amount of information in a network. Improvement depends on the queue threshold of the nodes and the communication costs. So, it can be observed that higher queue thresholds improve a lot the behavior of nodes with low communication costs and less the ones with high communication costs.

On the other hand, it can be observed that after a specific value of the queue threshold the behavior of the algorithm remains the same in terms of lifetime duration and packets received by the sink. The exact queue threshold value is not specific and depends on the network topology.

From the simulation results, it can be observed that the optimum queue threshold value for the ComCostNetwork is 7.

The above observations raised another quote. How the algorithm will behave, if we replace the connection costs of the network with the appropriate number of sensors with no connection costs? For example a connection cost with value 3 will be replaced by 2 supplementary nodes (3 connections with cost value 1 each.) The network derived by these changes can be seen in the following figure 6.7.

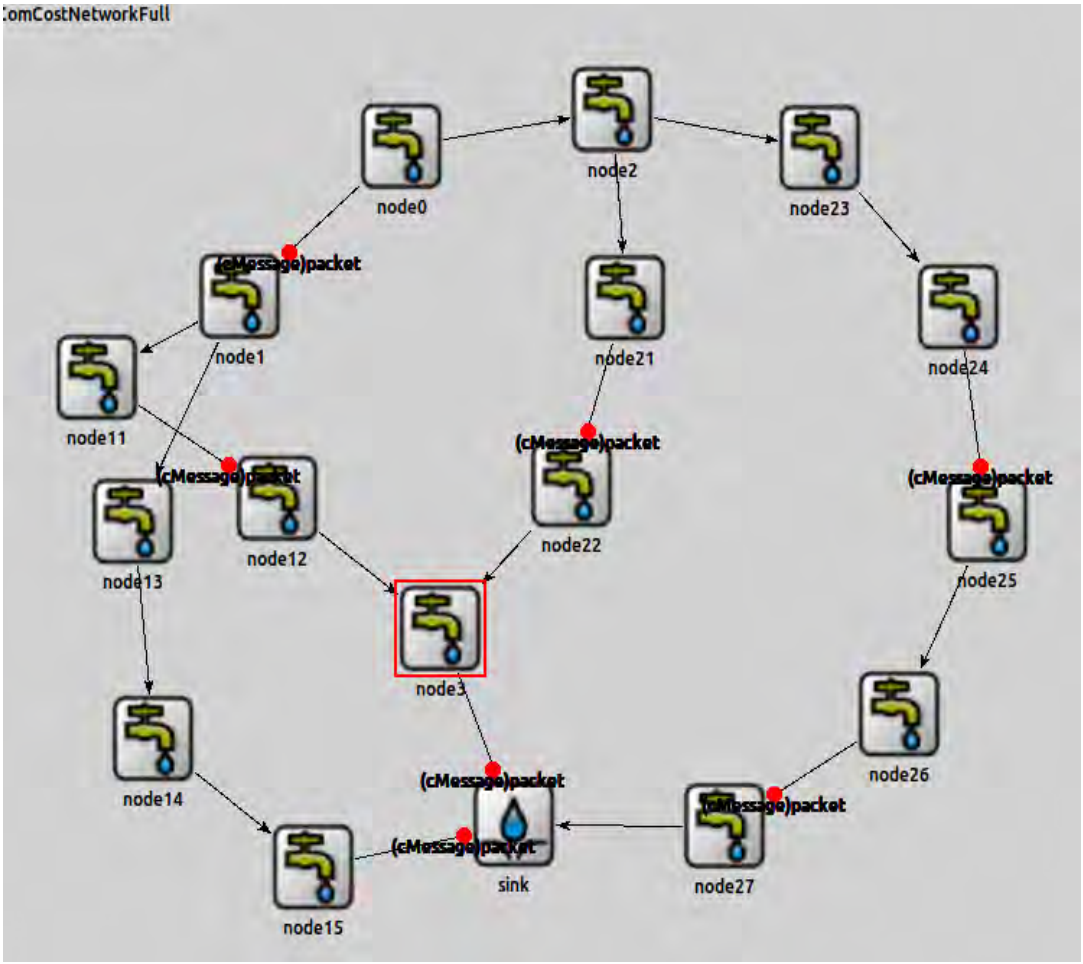
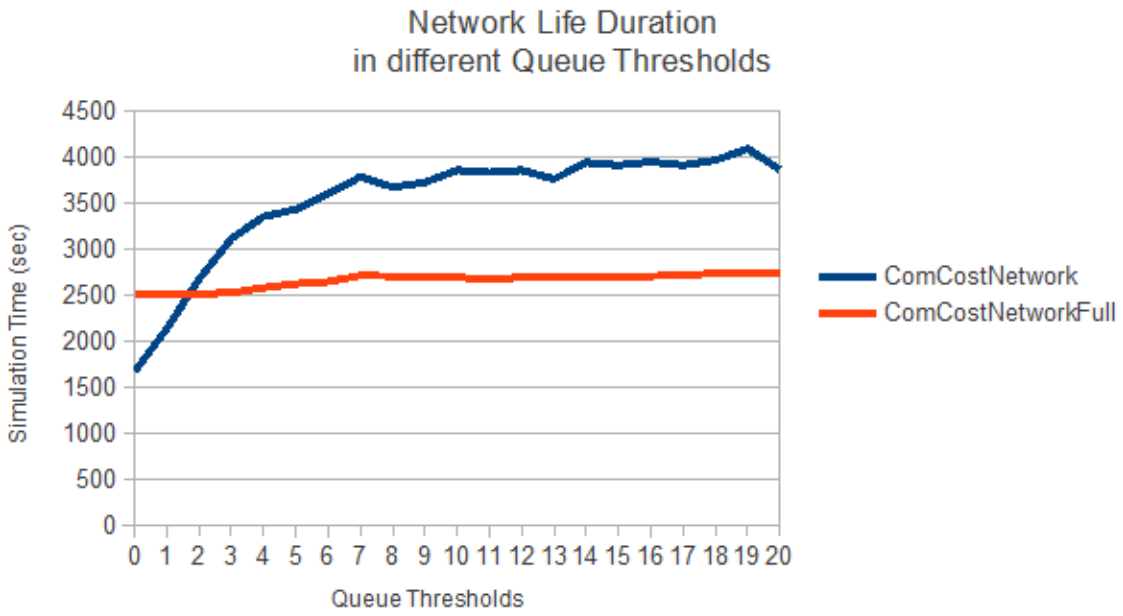
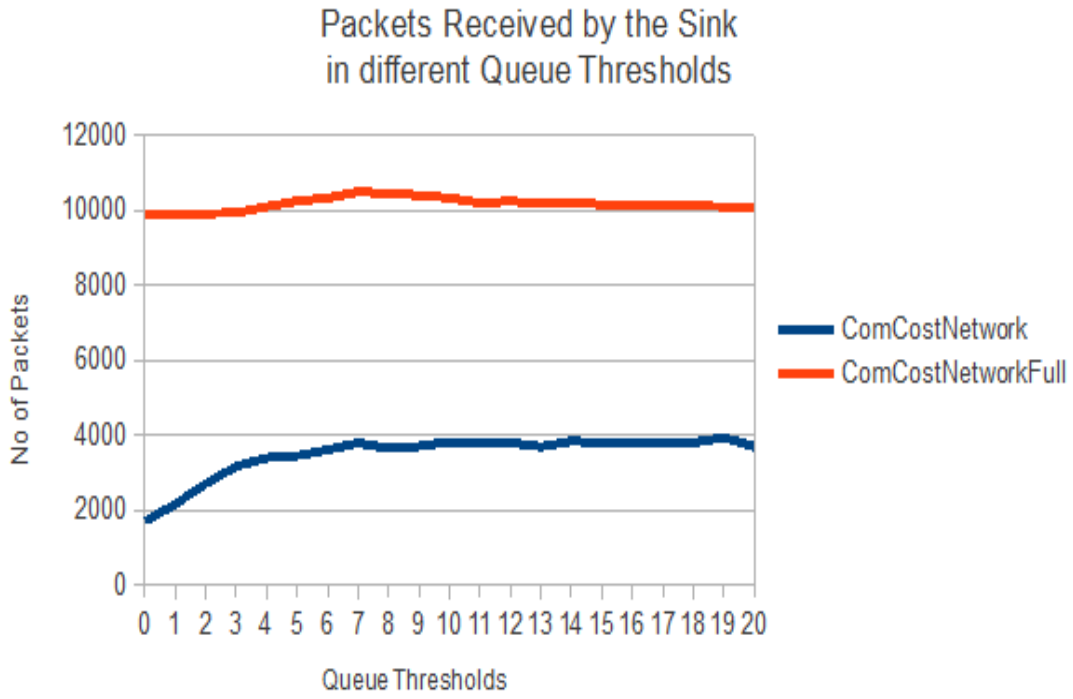


Figure 6.7: Full ComCost network

The results of the simulations for different queue threshold values can be seen in the following diagrams:





It's obvious that the number of information packets received by the sink is larger in the new network. This is an expected result because the number of nodes – sensors is larger.

However, the impact of the algorithm to the lifetime duration is more interesting. With queue threshold near zero, the lifetime duration of the new network is larger than the original one. However, the algorithm increases the original network's lifetime duration for queue threshold values larger than 2.

The above observations raise issues for future work on the optimization of the algorithm. Implementing nodes – sensors with individual queue threshold depending on their connections costs could improve further the algorithm's efficiency.

Also, the algorithm has to take into account priorities of the information packets. Higher priority packets have to transfer earlier to the sink than the lower priority ones.

Finally, the packets could be time-stamped, so that they could be discarded if they haven't reached the sink in time.

Conclusion

We have designed an efficient algorithm for the selecting of the most informative positions in any area that anybody wants to place sensors. We said that every time we select the sensor with the largest range and we place it at or near the center of the area. As long as there is uncovered area, we place a new sensor in the area. The point, where the sensor will be placed, has to be inside the range of one of the placed sensors and the sensor will cover larger uncovered area.

From the other hand, because we want not only the set of sensors to be informative but also we want to maximize the network lifetime with the help of Omnet++ we create an algorithm that extends the life of the network. The algorithm tries to move as more packets as possible from the sensors to the sink and in the same time to extend the life of the network by minimizing the consumed power of the nodes.

It's obvious that there is a tradeoff between the number of nodes and the weights of their connections. More nodes increase the possibility of a network to die early while “heavy” connections decrease the amount of information. The simulation gives the opportunity to experiment among different networks and discover their efficiency in terms of the amount of information and life duration.

Future Work

As future work we could create these two algorithms to take results which take into consideration both parameters we set, namely the most informative positions and also the extension of the network life time. Furthermore we could to upgrade the placement algorithm to take into consideration the sensors communication cost or the geometrical calculations or both of them.

REFERENCES

[1] Whitman, E.C . Sosus: The “Secret Weapon” of Undersea Surveillance. Undersea Warfare 2005, 7. Available online: http://www.navy.mil/navydata/cno/n87/usw/issue 25/sosus.htm (accessed September 17, 2009).

[2] Chong, C.Y.; Kumar, S.P. Sensor Networks: Evolution, Opportunities, and Challenges. Proc.IEEE 2003, 91, 1247–1256.

[3] D. Estrin, R. Govindan, and J. Heidemann. Special Issue on Embedding the Internet, Communications of the ACM, volume 43, 2000.

[4] B. Badrinath, M. Srivastava, K. Mills, J. Scholtz, and K. Sollins. Special Issue on Smart Spaces and Environments, *IEEE* Personal Communications, 2000.

[5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," IEEE Communications Magazine, August 2002, pp. 102-114.

[6] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Modeldriven data acquisition in sensor networks. In VLDB, 2004.

[7] E. Shih, S. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, A. Chandrakasan. Physical layer driven protocol and algorithm design for energy-efficient wireless sensor networks, in: Proceedings of the ACM Special Interest Group on Mobility of Systems Users, Data, and Computing (SIGMOBILE01), Italy, 2001, pp. 272–286.

[8] K. Kar and S. Banerjee. Node placement for connected coverage in sensor networks. In WiOpt, 2003.

[9] S. Funke, A. Kesselman, F. Kuhn, Z. Lotker, and M. Segal. Improved approximation algorithms for connected sensor cover. In ADHOC, 04.

[10] A. Cerpa, J. L. Wong, L. Kuang, M. Potkonjak, and D. Estrin. Statistical model of
lossy links in wireless sensor networks. In IPSN, 2005.

[11] N. Garg. Saving an epsilon: a 2-approximation for the k-mst problem in graphs. In STOC, 2005.

[12] E.J. Pauwels, A.A. Salah, and R. Tavenard. Sensor networks for ambient intelligence. In Multimedia Signal Processing, 2007. MMSP 2007. IEEE 9th Workshop on, pages 13 --16, oct. 2007.

[13] H. Kenniche and V. Ravelomananana. Random geometric graphs as model of wireless sensor networks. In Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on, volume 4, pages 103 --107, feb. 2010.

[14] Aristeidis Karalis, J. D. Joannopoulos, and Marin Soljačić. Efficient wireless nonradiative mid-range energy transfer. Annals of Physics, 323(1):34--48, January 2008.
[15] James Aspnes and Eric Ruppert. An introduction to population protocols. In Benoît Garbinato, Hugo Miranda, and Luís Rodrigues, editors, Middleware for Network Eccentric and Mobile Applications, pages 97--120. Springer-Verlag, 2009.

[16] Renita Machado and Sirin Tekinay. A survey of game-theoretic approaches in wireless sensor networks. Comput. Netw., 52(16):3047--3061, November 2008.

[17] CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE. Proceedings of a Workshop on Distributed Sensor Nets Held at Pittsburgh, Pennsylvania on December 7-8, 1978. Defense Technical Information Center, 1978.

[18] Richard Rashid, Robert Baron, Ro Forin, David Golub, and Michael Jones. Mach: A system software kernel. In In Proceedings of the 1989 IEEE International Conference, COMPCON, pages 176--178. Press, 1989.

[19] Chee-Yee Chong, Kuo-Chu Chang, and Shozo Mori. Distributed tracking in distributed sensor networks. In American Control Conference, 1986, pages 1863--1868, june 1986.

[20] Richard T. Lacoss. Distributed mixed sensor aircraft tracking. In American Control Conference, 1987, pages 1827 --1830, june 1987.

[21] MIT Lincoln Laboratory. Distributed sensor networks. http://robotics.eecs.berkeley.edu/~pister/290Q/Papers/Historyontext/DistributedSensorNetworks-LL-1986.

[22] Muhammad Omer Farooq and Thomas Kunz. Operating systems for wireless sensor networks: A survey. In Sensors 11, 2011.

[23] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. Tinyos: An operating system for sensor networks. In in Ambient Intelligence. Springer Verlag, 2004.

[24] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, LCN '04, pages 455--462, Washington, DC, USA, 2004. IEEE Computer Society.

[25] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. In ACM/Kluwer Mobile Networks & Applications (MONET), Special Issue on Wireless Sensor Networks, page 2005, 2005.

[26] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In Proceedings of the 3rd international conference on Mobile systems, applications, and services, MobiSys '05, pages 163-176, New York, NY, USA, 2005.

[27] A. Eswaran, A. Rowe, and R. Rajkumar. Nano-rk: an energy-aware resourcecentric rtos for sensor networks. In Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International, pages 10 pp. --265, dec. 2005.

[28] Philip Levis and David Culler. Maté: a tiny virtual machine for sensor networks. SIGOPS Oper. Syst. Rev., 36(5):85--95, October 2002.

[29] D. Mahrenholz and S. Ivanov. Real-time network emulation with ns-2. In Distributed Simulation and Real-Time Applications, 2004. DS-RT 2004. Eighth IEEE International Symposium on, pages 29 -- 36, oct. 2004.

[30] Gustavo Carneiro, Helder Fontes, and Manuel Ricardo. Fast prototyping of network protocols through ns-3 simulation model reuse. Simulation Modelling Practice and Theory, 19(9):2063 -- 2075, 2011.

[31] Xiaodong Xian, Weiren Shi, and He Huang. Comparison of omnet++ and other simulator for wsn simulation. In Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on, pages 1439 --1443, june 2008.

[32] M. Lord and D. Memmi. Netsim: a simulation and visualization software for information network modeling. In e-Technologies, 2008 International MCETECH Conference on, pages 167 --177, jan. 2008.

[33] A. Sobeih, Mahesh Viswanathan, D. Marinov, and J.C. Hou. J-sim: An integrated environment for simulation and model checking of network protocols. In Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pages 1 - 6, march 2007.

[34] Tobias Baumgartner, Ioannis Chatzigiannakis, Sandor P Fekete, Christos Koninis, Alexander Kroeller, and Apostolos Pyrgelis. Wiselib: A generic algorithm library for heterogeneous sensor networks. *Computer Engineering*, page 16, 2011.

[35] Di Tian, Nicolas D. Georganas. Connectivity maintenance and coverage preservation in wireless sensor networks. Elsevier, 2004.

[36] S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. In Proc. Of IEEE Intl. Conf. on Communications (ICC), 2001.

[37] S. Shakkottai, R. Srikant, and N. Shroff. Unreliable sensor grids: Coverage, connectivity and diameter. In Proceedings of INFOCOM (to appear), 2003.

[38] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava. Coverage problems in wireless ad-hoc sensor networks. In Proc. of the Conf. on Computer Communications (INFOCOM), 2001.

[39] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In Proc. Of Intl. Conf. on System Sciences (HICSS), 2000.

[40] G. Pottie and W. Kaiser. Wireless sensor networks. Communications of the ACM, 43, 2002.

[41] I. Papadimitriou and L. Georgiadis, "Maximum lifetime routing to mobile sink in wireless sensor networks", SoftCOM, September 2005.

[42] OMNeT++ Community Site, http://www.omnetpp.org.

APPENDIX

Here are their codes Omnet the classes Node and Sink. To run the program requires a topology file with extension *. Ned and a configuration file for omnetpp.ini.

Node

```
#include "Node.h"


Define_Module(Node);

Node::Node() {
        packetFromSensorReceived=0;
        findOptimalRouting=0;
        sendPackets=0;

        packetToBeRouted=0;

}

Node::~Node() {
   cancelAndDelete(packetFromSensorReceived);
        cancelAndDelete(findOptimalRouting);
        cancelAndDelete(sendPackets);
        //simulation
        cancelAndDelete(packetToBeRouted);
}


void Node::initialize()
{

        iaTimeHistogram.setName("interarrival times");
        arrivalsVector.setName("arrivals");
        arrivalsVector.setInterpolationMode(cOutVector::NONE);

        for(int k=0;k<gateSize("out");k++) {
                outgoingNeighbour[k]=gate("out",k)->getNextGate()->getOwnerModule();
                outgoingNode[k]=dynamic_cast<Node *> (outgoingNeighbour[k]);
        }
        for (int n=0;n<gateSize("out");n++)
                queueDifference[n]=optimalRoute[n]=routeCriterion[n]=0.0;
        power_per_pps[0]=par("power_per_pps_0");
        power_per_pps[1]=par("power_per_pps_1");
```

```
        power_per_pps[2]=par("power_per_pps_2");
        power_per_pps[3]=par("power_per_pps_3");
        queueThreshold=par("queueThreshold");
        //simulation
        poisson_incoming_packets=poisson(1.5,0);
        //simulation
        max_power=par("max_power");
        total_energy=par("total_energy");
        WATCH(total_energy);
        //simulation
        timeInterval=2.0;
        //simulation
        packetFromSensorReceived=new cMessage("Packets From Sensor Received");
        findOptimalRouting=new cMessage("Find Optimal Routing");
        sendPackets=new cMessage("Send Packets");

        throughput=0;
        idle_throughput=0;

        WATCH(throughput);
        WATCH(idle_throughput);

        scheduleAt( 0.0 , packetFromSensorReceived );



}

void Node::handleMessage(cMessage *msg)
{
        if ( simTime().dbl()==0.0){
                scheduleAt ( simTime() + (3*timeInterval) , packetFromSensorReceived );
                ev << "Starting the simulation." << "\n";
        }//simulation
        else {
                //simulation
                //simulation
                if (msg->isSelfMessage()) {
                        //simulation
                        //simulation
                        //simulation
                        if (!strcmp("Packets From Sensor Received",msg->getName())) {
                                int counter=0;
                                value_of_poisson=poisson(1.5,0);
                                for(int i=0;i<value_of_poisson;i++) {//new line of code
                                        cMessage *packet=new cMessage("packet");
                                        queue.insert(packet);
                                        ev << "Inserting packet that has just arrived from the
sensor application." << "\n";
                                        counter++;
                                }
```

```
                                ev << "A total of " << counter << " packets have been
inserted in the queue." << "\n";

                                scheduleAt( simTime() + timeInterval , findOptimalRouting
);
                        }
                        //simulation
                        //simulation
                        //simulation
                        if(!strcmp("Find Optimal Routing",msg->getName())) {
                                for(int l=0;l<gateSize("out");l++) {
                                        if(outgoingNode[l]) {
                                                queueDifference[l]= (this->queue.length() -
outgoingNode[l]->queue.length());
                                                optimalRoute[l]=queueDifference[l] > 0 ?
queueDifference[l] : 0;
                                                ev << "Current node has " << queue.length()
<< " packets stored in its queue.\n";
                                                ev << "Outgoing node " <<
outgoingNode[l]->getFullName() << " has " << outgoingNode[l]->queue.length() << "
packets stored in its queue.\n";
                                                ev << "Thus the optimal weight for this link
is " << optimalRoute[l] << ".\n";

        routeCriterion[l]=(optimalRoute[l]/power_per_pps[l])-queueThreshold;
                                                ev << "Criterion is " << routeCriterion[l] <<
".\n";
                                        }

                                        else {
                                                queueDifference[l]= this->queue.length();
                                                optimalRoute[l]=queueDifference[l] > 0 ?
queueDifference[l] : 0;
                                                ev << "Current node has " << queue.length()
<< " packets stored in its queue.\n";
                                                ev << "It is connected to the sink also.\n";
                                                ev << "Thus the optimal weight for this link
is " << optimalRoute[l] << ".\n";

        routeCriterion[l]=(optimalRoute[l]/power_per_pps[l])-queueThreshold;
                                                ev << "Criterion is " << routeCriterion[l] <<
".\n";
                                        }
                                }
                                scheduleAt( simTime() + timeInterval , sendPackets );
                        }
                        //simulation
                        //simulation
                        //simulation
                        else if(!strcmp("Send Packets",msg->getName())) {
                                for(int m=0;m<gateSize("out");m++) {
                                        if (routeCriterion[m]>0) {
```

```
                                        double idle_counter=0;
                                        double real_counter=0;
                                        for(int
j=0;j<(max_power/power_per_pps[m]);j++) {
                                                if (this->queue.empty()) {
                                                        ev << "Queue is empty,
sending out blank packet.\n";
                                                        cMessage
*blank_packet=new cMessage("blank_packet");

                packetToBeRouted=blank_packet;
                                                        send(packetToBeRouted ,
"out",m);
                                                        idle_counter++;
                                                        idle_throughput++;
                                                }
                                                else {
                                                        packetToBeRouted=
(cMessage * ) queue.pop();
                                                        send(packetToBeRouted ,
"out",m);
                                                        real_counter++;
                                                        throughput++;
                                                }
                                        }
                                //              total_energy-
=(max_power*(durationOfSubTimeslot.dbl())*(real_counter/(real_counter+idle_counter)));
                                        total_energy-
=(power_per_pps[m]*(timeInterval.dbl())*((real_counter+idle_counter)));

                                        ev << "Remaining energy is : " <<
total_energy << ".\n";
                                        if (total_energy<=0) {
                                                ev << this->getFullName() << " has
just died resulting in the network's end of operation.\n";
                                                endSimulation();
                                        }
                                }
                        }

                        scheduleAt( simTime() + timeInterval ,
packetFromSensorReceived );
                }
        }

        //simulation
        //simulation
        else {
                if(!strcmp("blank packet",msg->getName())) {
                        ev << "Blank packet has just arrived. It will be discarded.\n";
                        delete msg;
                }
```

```
                        else {
                                queue.insert(msg);
                                ev << "Packet from the network layer has just arrived and
has been inserted in the queue.\n";
                        }
                }
        }


}
```

Sink

```
#include "Sink.h"


Define_Module(Sink);

Sink::Sink() {
   // TODO Auto-generated constructor stub

}

Sink::~Sink() {
   // TODO Auto-generated destructor stub
}


void Sink::initialize()
{
        throughput=0;
        WATCH(throughput);
        lastArrival = simTime();
   iaTimeHistogram.setName("interarrival times");
   arrivalsVector.setName("arrivals");
   arrivalsVector.setInterpolationMode(cOutVector::NONE);
}

void Sink::handleMessage(cMessage *msg)
{

   //simtime_t d = simTime() - lastArrival;

   EV << "Received " << msg->getName() << endl;

        if (!strcmp("packet",msg->getName())){

                throughput++;
                iaTimeHistogram.collect(throughput);
                arrivalsVector.record(throughput);
```

```
                lastArrival = simTime();


        }

        delete msg;

}

void Sink::finish()
{

        recordStatistic(&iaTimeHistogram);
   ev << "Total throughput is : " << throughput << " packets." << "\n";
}
```