# A Flooding Approach to Harvesting Agents

by

**Iker Jimenez**

A dissertation submitted to the University of Dublin,

in partial fulfilment of the requirements for the degree of

M.Sc. Computer Science

2006

# Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Iker Jimenez

September 10, 2006

# Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this dissertation upon request.

Iker Jimenez

September 10, 2006

# Acknowledgments

I would like to thank my project supervisor, Stefan Weber, for his direction and advice during the project. To all my family and friends - for their support during this year. Finally, to the NDS people, for making this a most enjoyable, memorable year.

Iker Jimenez

*University of Dublin, Trinity College*
*September 2006*

# A Flooding Approach to Harvesting Agents

Iker Jimenez,
University of Dublin, Trinity College, 2006

Supervisor: Stefan Weber

Mobile Wireless Ad Hoc networks (MANETs) are composed of stations or devices that are connected directly by wireless links. These nodes can form any arbitrary topology, don't make use of any backbone structure and can move freely. They are expected to be widely used in the near future.

Networked applications communication pattern is defined by direct communication between endpoints across a number of hops. Translation of these applications to MANETs introduces challenges, communication across multiple hops is difficult because of the mobility of nodes, varying network topology and changing network population.

Exploiting the inherent characteristics of wireless communication broadcast that every signal is broadcasted to all nodes in the immediate vicinity allows for a more reliable distribution of data throughout a network. A programming paradigm that addresses the incompatibility of current applications and ad hoc networks can then be

defined by employing this mechanism to propagate active code in network.

This research implements a group of five different broadcasting protocols specifically designed for MANETs and runs a set of experiments to analyse them in a wireless network simulator called SWANS. A complete agent framework has been developed to enable nodes to make use of the propagated active code.

This dissertation contrasts the selected broadcasting protocols for mobile ad hoc networks and describes the suitability of each one for different network conditions.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background

Networked applications communication patterns are defined by direct communication between endpoints across a number of hops. Translation of these applications to ad-hoc networks introduces challenges.

An ad-hoc network is a self-configuring network of stations or devices that are connected directly by wireless links and not via an access point, these nodes can form any arbitrary topology [25]. A particular case of ad hoc networks are mobile ad-hoc networks, commonly known as MANETs, where stations can move around freely and change the network topology rapidly and unpredictably. Networks of this kind can be part of a larger wired network or work in a standalone fashion.

MANETs provide mobile communication capability to satisfy a need of a temporary nature and without the existence of any well-defined infrastructure. Most common scenarios for this kind of networks are disaster recovery situations, defence applications, emergency medical situations, academic institutions, corporate conventions or meetings etc. as they just need a minimal configuration and can be quickly deployed.

MANETs are not a new concept, originally they were called "packet radio" or "multi-hop networks" and were part of DARPA [6] projects in the 1970s. It is interesting to note that these early packet radio systems predated the Internet, and indeed were part of the motivation of the original Internet Protocol suite. Later DARPA experiments included the Survivable Radio Network (SURAN) project, which took place

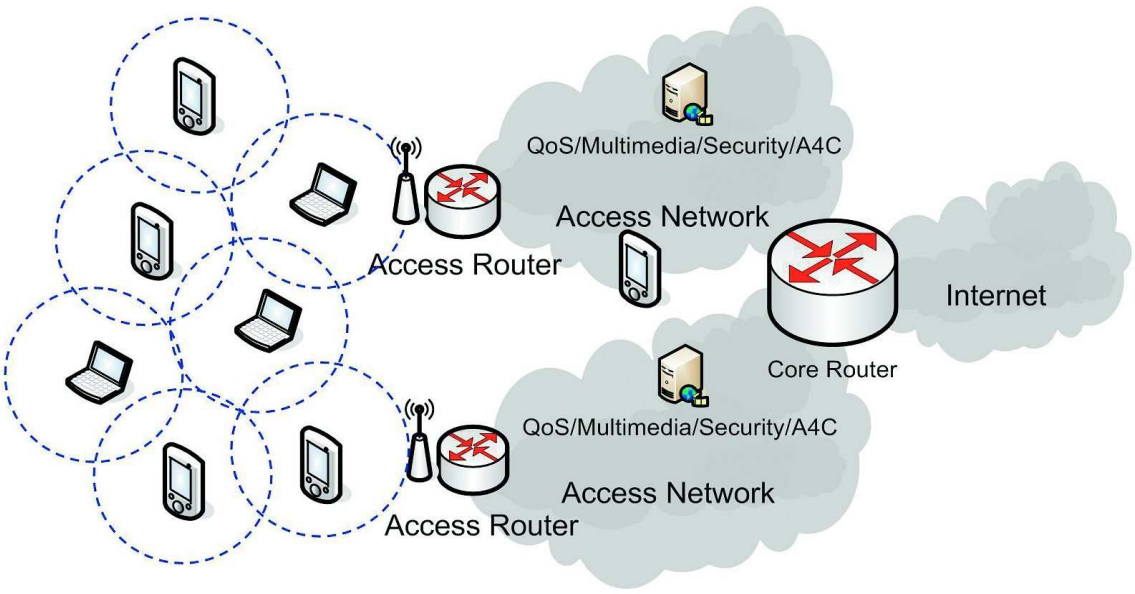


Figure 1.1: Mobile Ad Hoc Network *MANET*.

in the 1980s [19] and significantly improved upon the radios, making them smaller, cheaper, power-thrifty. The scalability of algorithms and resilience to electronic attack were also matter of improvement.

The latest interest of the academic community in MANETs started in the mid 1990s, when 802.11 radio cards for personal computers and handheld devices became popular. Most important MANET projects are focused on military utility; Joint Tactical Radio System (JTRS) for use by the U.S. military in field operations or Near-Term Digital Radio (NTDR) for example. However, there are other projects taking advantage of the MANETs, as the MIT Media Lab $100 laptop program, which has developed a cheap laptop for mass distribution, several million at a time, to developing countries for education. These laptops use a Linux system with ad-hoc wireless networking capabilities enabled in order to develop their own communications network out of the box. A working group called "manet" has been formed by the Internet Engineering Task Force (IETF) [8] to study the related issues and stimulate research in MANETs [9].

Communication across multiple hops is difficult because the mobility of nodes vary-

ing network mobility and changing network population. Due to this mobile nature of the nodes, the devices that are normally used in this kind of networks have some physical limitations that influence greatly the way the communications take place. As they are designed to be constantly moved they trend to be small sized and lightly weighted. These characteristics introduce limitations in the autonomy of the batteries and efforts to save as much energy as possible are made in the design of the communication protocols.

In the other hand, not having a fixed location makes the communications more difficult to take place comparing to a wired network and network topology information messages are normally needed in order to deliver the desired information. Nodes don't have accurate information of the topology around them because it is constantly changing, so they have to ask information to nodes around themselves. Different approaches for this topology information gathering have been followed, each one with its benefits and drawbacks.

Searching in MANETs is not an easy problem. In a searching usual scenario a node is looking for a certain item, i.e. a file. How to ensure a high probability of finding it -if it exists- with the minimum effort? Nodes are joining or leaving the network continuously, constantly moving, network can get partitioned, leaving some nodes isolated from the rest. As most of the nodes are not within radio signal coverage area of the requester node, there is a need for using intermediate nodes.

There are different approaches available to the problem of searching in MANETs. One of the approaches is unicast search, using TCP connections. The request jumps from node to node, using different routing algorithms, searching for the desired item. A Peer to Peer (P2P) approach can be also applied. The requester could search in a central server which has information about what each node is sharing or using a decentralised approach. Either way there is a need for keeping an updated topology information.

In this research the focus is set to network wide broadcasting algorithms, using them to search for a resource in the network. These broadcasting algorithms try to reach all the nodes within range during their transmissions; exploiting the inherent characteristics of wireless communication broadcast that every signal is broadcasted to all nodes in the immediate vicinity allows for a more reliable distribution of data throughout a network. A considerable number of broadcasting schemes have been pro-

posed during the last years but there is no standard broadcasting method for MANETs yet.

Autonomous Intelligent agents are being used as part of the searching process. An intelligent agent is a software program that shows some form of artificial intelligence. In this background "intelligent" means the ability to adapt and learn. An agent is initially created by a node and broadcasted to help the node find the desired resource in the network. As lately is explained agents that have been used for this research are rather primitive and can roughly be called intelligent, but the framework is designed in a way that doesn't make complicated to develop a more sophisticated agent, with a more developed intelligence able to take advantage of the capabilities the framework can offer to the agent.

## 1.2 Projects Goals and Scope

As stated in the previous section, the main scope of the project is to determine the suitability of an agent based flooding approach to communication in wireless ad hoc networks, to test the performance of different searching algorithms that take advantage of the broadcasting capabilities inherent to wireless ad hoc networks.

Wireless ad-hoc networks have singular characteristics as node density or speed that make different broadcasting techniques to act in a very different way when environment characteristics vary. Is one of the goals of this project to determine how each different broadcasting technique reacts to the different network set-ups that have been designed, study how they react to the changing network conditions and check if they are able to accomplish their tasks.

As a preliminary goal, different broadcasting techniques that have been already designed by the research community have been studied [2]. These techniques can be classified in different groups based on their design characteristics. From all these techniques a significant set has been selected, based on the design characteristics of each one, trying to pick up some of the most representative techniques from each group.

The chosen broadcasting techniques have been implemented in order to test them in a network simulator. A large set of simulations has then been defined to test the capabilities and performance of each of the implemented methods. The performance of these techniques has been compared under different network conditions, such as heavily

and poorly populated networks, static or mobile nodes, etc.

As part of the project an agent framework has been developed. This framework is capable of executing incoming agents and provides them with access to a set of resources that can help the agents in their decision making process.

Finally, the suitability of each approach for different network configurations and scenarios has been determined.

# Chapter 2

# State of the Art

In this chapter current situation about Mobile Ad hoc Networks is explained. In the first section papers related to this research are introduced and explained and in the next section a representative set of broadcasting algorithms is detailed and classified.

## 2.1 Related Work

Sze-Yao et al. [21] introduce the concept of broadcasting storm. As radio signals are likely to overlap with others in a geographical area, a straightforward broadcasting by flooding is usually very costly and will result in serious redundancy, contention, and collision, to which is referred as the broadcast storm problem.

Redundancy consists on a node sending a broadcast message to its neighbours, and by the time these neighbours received the message they already had it. Contention is defined as a mobile host trying to broadcast its message at the same time as many of its neighbours are also trying to broadcast theirs. This would result into too many radio signals contending for the transmission channel and therefore none of these messages would reach their destinations. Moreover, due to the absence of collision detection in the radio devices collisions are more likely to occur and cause more damage.

MANETs are composed of mobile nodes that communicate with each other at random periods without the help of a base station or access points. These nodes are usually equipped with carrier sense multiple access with collision avoidance (CSMA/CA) [13] radio devices.

In these mobile networks synchronisation -synchronise between the nodes for not transmitting at the same time, using time slots or different frequencies- is unlikely, and global network topology information is normally unavailable to facilitate the scheduling of a broadcast, as it is continuously changing. This scenario allows a node to communicate with another node directly or through intermediate nodes. In the second case is also known as a multihop scenario.

The broadcasting issue is then defined as sending a message to all the nodes in the network. Some characteristics of the broadcasting storm problem are also detailed:

- Any node can send a broadcasting packet at any time. Due to the node mobility and the lack of synchronisation pretending to create any kind of global topology knowledge is so expensive and useless and would require at least as much effort as the broadcast itself. Normally there is little or none local topology information collected before sending the broadcast.

- The broadcast is unreliable as no acknowledgement mechanism can be used. Nevertheless, best effort must always be attempted to distribute a broadcast message to as many hosts as possible without paying too much effort. One of the reasons why no acknowledge can be used is that a node can lose a broadcast message because it is off-line, it is temporarily disconnected from the network, or it is experiencing repetitive collisions that disallow to receive any packet. Another reason is that the sender could experience medium contention if it receives all the acknowledges in a short period. Anyway, in many applications such as route discovery a 100% reliable broadcast is not necessary [5].

Sze-Yao et al. [21] show that if flooding is used carelessly, many redundant messages will be sent and serious contention or collision will be incurred. It demonstrates, through analyses and simulations, how serious this problem could be. Several schemes families, like probabilistic, counter-based, distance-based, location-based, and cluster-based scheme families, are proposed to solve this issue. Simulation results based on different parameter values are presented to verify and compare the effectiveness of these schemes, that will be introduced in the next section.

Thomas Kunz [22] researches in the broadcasting in MANETs problem as part of multicasting communication protocols, considering its use to allow high delivery ratios.

They selected a broadcast protocol, BCAST, and extended it with a NACK mechanism to increase the packet delivery ratio. They concluded that designing a reliability mechanism to improve the packet delivery ratio is a complex problem. Because of the mobile nature of MANETs, getting a 100% packet delivery is not possible, as the network might get partitioned, leaving nodes unconnected from the request senders. Particularly, in low mobility scenarios, these network partitions can exist for long periods of time, and recovering from them is therefore a complicated issue, and would require important buffering capabilities at all MANET nodes, which are usually limited devices.

The conclusions from Obraczka et al. [11] [12] show that flooding and broadcasting results in higher packet delivery ratios than multicast routing protocols. However, in the scenarios researched in these papers, flooding would only achieve a packet delivery ratio as low as 70%, leading the authors to conclude that "even flooding is insufficient for reliable multicast in ad hoc networks when mobility is very high" [12]. Even though these papers were researching about multicasting, the simulation scenarios were all configured based on the assumption that the totality of the nodes in the MANET was interested in the data packets, which is exactly a broadcasting scenario.

## 2.2 Broadcasting techniques

Williams et al. [2] and Sze-Yao et al. [21] list and explain an extensive set of broadcasting protocols for MANETs. Both classifications are very similar and define four main protocol families as can be seen in table 2.1.

Simple flooding is the simplest family of protocols, and in fact it only includes a single broadcasting protocol, which gives name to the family. In this broadcasting protocol each node rebroadcasts all the packets it receives only once, without any other consideration. In this approach nodes don't usually have any location nor topology information.

Probability based methods try to use some basic understanding of the network topology to assign a probability to a node to rebroadcast. They don't use any location information in their rebroadcasting probability calculating process.

Area based methods are based on the premise that all the nodes have a similar transmission range and after consulting their location information they will rebroadcast

only if they can reach any sufficient extra area.

Neighbour knowledge methods have updated topology information, gathered through a "Hello" packets exchange process and will rebroadcast or not based on this information.

These broadcasting protocol families have been explained in order of increasing algorithm complexity and per node state requirement. The objective of these added expenses is to decrease the amount of unneeded transmissions, in an effort of saving the scarce energy resources of the mobile nodes and avoiding congesting the network.

| Broadcasting Protocol Families | | | |
|---|---|---|---|
| | Rebroadcast Probability | Uses Location | Uses Topology |
| Simple Flooding | 100% | NO | NO |
| Probability Based | Fixed or Variable % | NO | In a very basic way |
| Area Based | Variable % | YES | In a basic way |
| Neighbour Knowledge | Variable % | NO | Yes, extensively |

Table 2.1: Broadcasting Protocol Families

### 2.2.1 Simple Flooding Protocols

This protocol family has only a single broadcasting protocol, Simple Flooding. In this protocol a source node starts a broadcasting process by sending a packet to all the nodes within its coverage area. On receipt of this packet each node will rebroadcast it only once. Different ways of identifying broadcasting packets have been proposed, one way to do so is to associate a tuple -source ID, sequence number- with each broadcasted

message as it is explained by Broth et al. [5] and Perkins et al. [3].

### 2.2.2 Probability Based Methods

These broadcasting methods are based on the fact that in dense MANETs some node's broadcasted packets don't reach any additional receiver node if a node near them has broadcasted just before them, as both share the same set of receiver nodes. Both Probabilistic Scheme and Counter Based Scheme, explained in the following paragraphs, try to adapt to network conditions in order to avoid unnecesary rebroadcasts.

#### Probabilistic Scheme

The Probabilistic scheme, explained by Sze-Yao et al. [21] is very similar to Flooding, the only difference is that in this protocol nodes will only rebroadcast with a predefined probability. When the network is highly populated different nodes will cover a similar area. By this way, randomly choosing some nodes not to rebroadcast will avoid wasting node resources sending packets that are already send or receiving packets that are already received. On the other hand, in poorly populated networks there are not too many nodes with the same coverage area and the risk of not reaching all the nodes is high, particularly those in the edge of the network. In this case, depending on the node density, only a high probability of rebroadcasting could be applied. Consequently, when the probability is 100%, this scheme is exactly the same as Flooding.

In order to avoid the contention and collision problems characteristics of the broadcast storm problem a small random delay needs to be introduced before rebroadcasting any message. This way the timing for rebroadcasting in different nodes would be different.

#### Counter Based Scheme

Counter Based Scheme bases its algorithm on the fact that there is an inverse relationship between the number of times the same broadcasted packet is received and the probability for that node to reach any additional non-covered nodes with its rebroadcasting.

When a node receives a packet for the first time it establishes a random assessment delay (RAD) and starts a counter, which it initialises to a value of one. It won't

decide whether to rebroadcast or not until the end of this random time, based on the number of times it has received that same packet. Previously a threshold value for rebroadcasting must have been established. According to Sze-Yao et al. [21] values greater than six for the threshold provide a little extra coverage area.

One of the main advantages of this protocol is that it is very flexible to network densities, as in areas of the network where there is a high density of nodes only a few of them will rebroadcast while in other less populated areas, like in the edges of the network, almost all nodes will do it. Another advantage of this algorithm is its simplicity, which makes it easy to implement and doesn't require a high amount of resources.

| Probability Based Methods | | | |
|---|---|---|---|
| | Rebroadcast Probability | Uses Location | Uses Topology |
| Probabilistic Scheme | Fixed % | NO | NO |
| Counter-Based Scheme | Variable % | NO | In a basic way |

Table 2.2: Probability Based Methods

### 2.2.3 Area Based Methods

Area based methods try to make use of location information to adjust packet rebroadcasting. In both Distance-Based Scheme and Location-Based Scheme nodes have some sort of information of the physical location of the nodes around them.

#### Distance-Based Scheme

In this broadcasting protocol the distance between the sender and the receiver is used to decide if the receiver will rebroadcast or not. The distance is calculated based on the signal strength perceived by the receiver. This implies that all the nodes are supposed to have a similar radio signal strength. A threshold distance for rebroadcasting is set

up previously. When a packet is received a RAD is established, every time the node receives a copy of that packet it stores its calculated distance from the sender and when the RAD expires if there has not been a packet received from a distance smaller than the established threshold distance it will rebroadcast.

The limitation of this approach is that it requires all the devices to have a similar radio signal strength, which it makes it unusable in an heterogeneous network.

### Location-Based Scheme

This broadcasting protocol requires all the nodes to have a GPS or other mean to calculate their own location. Each time a node broadcasts a packet it will add its own location information to the header of the packet.

When a node receives a packet it will calculate what additional area it would cover based on its location information and the location information of the sender. If the additional area is smaller than an established threshold value the packet would be discarded. However, if the additional area is bigger than the threshold value a RAD time would be calculated. Every time another copy of the packet is received the additional covered area would be recalculated to check if it falls below the threshold value. If the RAD expires and the packet has not been discarded it is rebroadcasted.

| Area Based Methods | | | | |
|---|---|---|---|---|
| | Rebroadcast Probability | Uses Location | Location Method | Uses Topology |
| Distance-Based Scheme | Fixed % | YES | Signal Strength | NO |
| Location-Based Scheme | Variable % | YES | GPS or similar | NO |

Table 2.3: Area Based Methods

### 2.2.4 Neighbour Knowledge Methods

This group of broadcasting methods gather information about the nodes around them prior to broadcast. Each of the following seven protocols make use of this neighbour information in a different way.

#### Flooding with Self Pruning

This is the simplest of the Neighbour Knowledge Methods. Lim et al. [7] name it as Flooding with Self Pruning.

In this protocol nodes exchange information about their neighbours via periodic "Hello" packets. So each node will know which its 1-hop neighbours are. When a node broadcasts a packet it includes information about its neighbours in the header. When the receiver processes the packet it checks its neighbour list against the sender's neighbour list. If it has neighbours that are not covered by the sender it will rebroadcast the packet, otherwise it will be discarded.

#### Scalable Broadcast Algorithm (SBA)

As this is a more advanced protocol it requires each node to know about the neighbours within 2-hop distance. With both this 2-hop neighbour information and the address of the sender node, the receiver node is capable to decide if it would reach any additional node by rebroadcasting.

The 2-hop neighbour information is gathered using the same way as in the previous protocol, the exchange of periodic "Hello" packets. In this case, the hello packet contains the address of the sender node and the list of its neighbours, allowing each node to keep updated information about all the nodes within 2-hop radius.

Once a new packet is received a RAD is established. Each time another copy of this same packet is received the node calculates if any additional node would be covered by its rebroadcast. Basically, as the receiver node knows which nodes it has as neighbours and which ones the sender has, the receiver node compares both lists and if it has any extra node it will schedule the packet for broadcasting. If no new nodes are covered the packet would just be discarded. When the RAD expires the packet is sent.

Peng et al. [24] propose a method for adjusting the RAD to network conditions. The RAD would be calculated by multiplying a random number by a ratio. This ratio

is the result of dividing the number of maximum number of neighbours that any of the node neighbour's has between the number of neighbours that node has. This weighting scheme favours nodes with more neighbours to rebroadcast before the rest.

### Dominant Pruning

Dominant Pruning uses 2-hop neighbour knowledge, obtained via "Hello" packets, for routing decisions. This protocol requires rebroadcasting nodes to choose a set of its 1-hop neighbours as rebroadcasting nodes previous to rebroadcasting .

These predetermined nodes will be the only ones allowed to rebroadcast. On the header of each broadcasted packet a list of rebroadcaster addresses is included. When a node receives a packet and is chosen as a rebroadcaster it will use a Greedy Set Cover algorithm, explained by Lim et al. [7], to decide which of its 1-hop neighbours will be rebroadcasters. It will create a cover set from its list of 2-hop neighbours and discard those that the sender has already covered. It will then choose the 1-hop node that covers most number of 2-hop nodes. This node will be chosen as a rebroadcaster. It will update the cover set, eliminating those 2-hop nodes covered by the chosen rebroadcaster and choose another rebroadcaster in the same way. This algorithm is repeated until all 2-hop nodes are covered.

### Multipoint Relaying

This protocol is described by Qayyum et al. [1] and is similar to Dominant Pruning in that rebroadcasting nodes are chosen from the senders previous to broadcasting. The node originating a broadcast packet has previously selected some or all of its one hop neighbours to rebroadcast all packets they receive from that node.

These nodes are called Multipoint Relays (MPR) and are the only nodes that can rebroadcast any packet received from that sender. Additionally, these MPRs are also required to choose which of its neighbours will be MPRs as well.

Ideally, MPRs would be those one hop nodes that cover most efficiently all of the two hop nodes. The algorithm for choosing these MPRs is the following:

1. Find which 2-hop nodes can only be reached by a single 1-hop node and assign those 1-hop nodes as MPRs.

2. Calculate the resultant cover set; the set of 2 hop nodes that are covered by the broadcasting of the already selected MPRs.

3. Choose a 1-hop node from the remaining 1-hop neighbours, that covers the most number of 2-hop neighbours that are not in the cover set.

4. Repeat from step number 2 until all 2-hop neighbours are covered.

In this protocol the MPRs are designated in the header of the exchanged "Hello" packets, so this state is refreshed or updated each time a new "Hello" packet is received. Depending on the source node of a packet the receiver node can be a MPR or not.

### Ad Hoc Broadcast Protocol

Ad Hoc Broadcast Protocol (AHBP) is very similar to Multipoint Relaying. In this protocol broadcasting neighbours are called Broadcast Relay Gateway (BRG) instead of MPRs, but essentially they are the same thing. They are chosen by the upstream sender previous to broadcasting using exactly the same algorithm described in the previous protocol. However, there are a few differences between them:

1. In AHBP the nodes are informed of being BRGs by a field in the broadcasted packet header, instead of using "Hello" packets for transmitting this information. By this way, the decision of which nodes are going to be BRGs is taken just before sending the packet, being more accurate and effective for the current topology.

2. When a node receives a broadcasted packet and is listed as a BRG it also has to decide which of its 1-hop neighbours will be BRGs. Contrary with what Multipoint Relaying does, AHBP takes in account where the packet came from, which nodes were covered by that same transmission and eliminates them from the neighbour information used to determine the BRGs.

3. AHBP is extended for high mobility networks in extended AHBP (AHBP-EX). If a node receives a packet from another node that has not exchanged "Hello" packets with it, it will assume that that packet is from a node that has just moved near itself. The receiver will act as a BRG even though it is not listed like that and rebroadcast that packet.

### CDS-Based Algorithm

Connected Dominating Set(CDS)-Based Broadcast Algorithm is explained by Peng et al. [23]. This algorithm is a more advanced way of choosing BRGs. Based on AHBP, it reduces the number of nodes to be covered by using the neighbour information of other selected BRGs for the same received packet.

In the broadcasted packet BRGs are listed in order of more covering neighbours. So a BRG can eliminate those nodes that are already covered by those BRG before itself in the BRG list of the received packet.

### LENWB

The Lightweight and Efficient Network-Wide Broadcast (LENWB) protocol, introduced by Sucec et al. [10] also uses 2-hop neighbour knowledge achieved from exchange of "Hello" packets. In this protocol, instead of a node choosing which of its 1-hop nodes will rebroadcast its packets, is the receiver nodes which will decide it.

Each node knows which of its neighbours have received that packet from the same transmission. It also knows which of those nodes that have received the packet have a higher priority than itself. Priority is based on the number of neighbours that each node has, so the more neighbours it has the higher the priority is. Based on this information a node will rebroadcast if there are nodes with a lower priority than itself that wouldn't be covered from nodes with a higher priority.

| Neighbour Knowledge Methods | | | |
|---|---|---|---|
| | Neighbour Knowledge | Algorithm | Broadcasting decision |
| Flooding with Self Pruning | 1-hop with "Hello" packets | Check if it covers more than the sender | In the receiver |
| Scalable Broadcast Algorithm (SBA) | 2-hop with "Hello" packets | Check if it covers additional nodes. | In the receiver |
| Dominant Pruning | 2-hop with "Hello" packets | Uses Greedy Set Cover | In the sender / Broadcast Packet |
| Multipoint Relaying | 2-hop with "Hello" packets | MPR Algorithm | In the sender / "Hello" Packet |
| Ad Hoc Broadcast Protocol | 2-hop with "Hello" packets | BRG(MPR Algorithm) + sender info. | In the sender / Broadcast Packet |
| CDS-Based Broadcast Algorithm | 2-hop with "Hello" packets | AHBP Algorithm + BRG Priorities | In the sender / Broadcast Packet |
| LENWB | 2-hop with "Hello" packets | Node priority + sender info. | In the receiver |

Table 2.4: Neighbour Knowledge Methods

# Chapter 3

# Design

In this chapter the design process of the simulator is explained. In the first section which protocols have been implemented and the reasons for choosing them are described. After it, the used network simulator is introduced. In the following section the application design is detailed. Finally, some relevant design decisions are described.

## 3.1 Implemented Protocols

For this research five protocols have been implemented. Simple Flooding from the Simple Flooding family, Probabilistic Schema and Counter Based Scheme from the Probability Based Methods, Scalable Broadcast Algorithm (SBA) and Ad Hoc Broadcast Protocol (AHBP) from the Neighbour Knowledge Methods. This section explains made choices.

Initially we decided to implement the same protocols as Williams et al. [2] to evaluate their results, but as we found some difficulties we changed a little bit our selection. We chose two of each family, except from the Simple Flooding family where there is only one protocol and from the Area Based Methods, where due to limitations of the used network simulator, none of the protocols has been implemented. Finally, we implemented 4 of the 5 protocols that are evaluated by Williams et al. [2] and another protocol that is not implemented in that same paper.

For the Probability Based methods, as there are only two protocols, we implemented both of them. Both the Probabilistic scheme and Counter-Based scheme were proposed

in the same paper [21]. According to this paper, Counter-Based scheme performs better than Probabilistic scheme. We wanted to check this conclusion so we implemented the Probabilistic scheme too. Moreover, as Simple Flooding is a Probabilistic scheme with a 100% of probability to rebroadcast, we thought that it could be useful and interesting to make comparisons between both of them.

Neighbour Knowledge is the family were a larger number of choices can be made; there are seven different available protocols. As can be seen in table 2.4 neighbour knowledge protocols can be classified based on where the decision to rebroadcast is taken, in the sender node or in the receiver node. So we decided to implement one protocol from each one. There are four protocols in which the receiver is the node that takes the broadcasting decision and three protocols in which the sender is the decider.

As explained by Williams et al. [2], some of the protocols in which the decision was taken locally do not perform correctly in certain network conditions. Both LENWB and Flooding with Self Pruning experience this problem, so the only protocol left which performs efficiently in all network conditions is SBA. We implemented this protocol to represent this subgroup of protocols.

From the subgroup of protocols in which the broadcasting decision is taken in the upstream node we chose to implement AHBP. Dominant Pruning can be discarded as it only takes in account 1-hop neighbour information to take the broadcasting decision. Multipoint Relaying is a less evolved version of AHBP and uses almost the same amount of resources. CDS-Based Broadcast Algorithm could be another good choice, but it requires more extensive calculations and apparently doesn't improve AHBP performance in a significant way.

## 3.2 JiST/SWANS: The Simulator

There are multiple available network simulators: ns2 [20], Opnet [15], GlomoSim [14] and others. The network simulator that we have used for this research JiST/SWANS [17], a network simulator from Cornell University, is completely programmed in Java, with available source code, it needs a low resource consumption, it's easy to scale, allows being distributed in different machines and can easily run Java programs on the simulated nodes.

As can be seen in figure 3.1, there are 4 layers in the application stack:

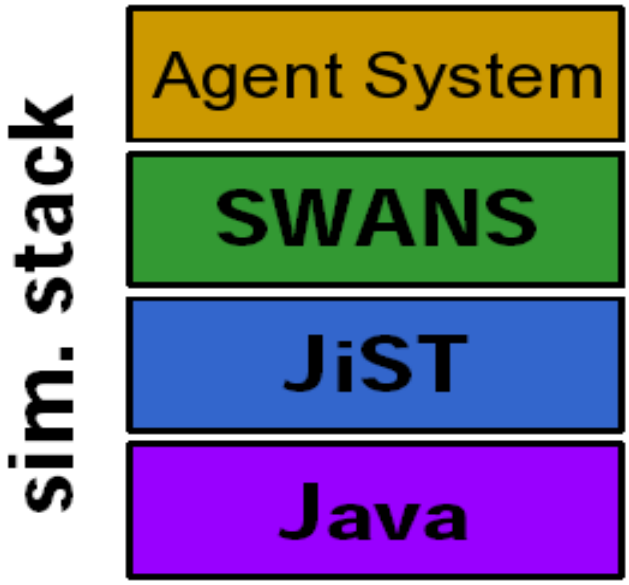


Figure 3.1: JiST/SWANS Stack

**JVM** Is the lower layer in the stack. Java Virtual Machine is the sandbox where the whole simulation takes place. It is a regular Java Virtual Machine.

**JiST** Inside the JVM JiST is run. JiST stands for Java in Simulation Time and is a high-performance discrete event simulation engine that runs over a standard Java virtual machine. It is a prototype of a new general-purpose approach to building discrete event simulators, called virtual machine-based simulation. JiST simulations are written in Java, compiled using a regular Java compiler, and run over a standard, unmodified virtual machine. It outperforms other simulators in both event throughput and memory footprint [17].

**SWANS** SWANS stands for Scalable Wireless Adhoc Network Simulator. Built atop the JiST platform, SWANS is organised as independent software components that can be composed to form complete wireless network or sensor network configurations. Its capabilities [17] are similar to ns2 and GloMoSim, but is able to simulate much larger networks. One of the most attractive advantages of SWANS is its ability of running standard Java network applications over simulated net-

works.

**Agent System** This is the application that has been developed to run on each simulated node. Basically it is just a normal Java UDP sockets network application with some special characteristics that will be explained in the following sections.

## 3.3 Application Design

In this section the Agent System application design is described. In the first subsection a searching agent use of case is detailed. After this a general class diagram with application's most important classes is explained. Finally some sequence diagrams explain the interaction that application classes have during typical execution.

### 3.3.1 Searching Agent Use Case

Figure 3.2 shows the use case of an agent searching for a token in a mobile wireless ad hoc network. As can be seen in the diagram, there are three possible actors interacting within the system:

**Sender Node** This is the node that creates the searching agent, specifies the desired token and starts the broadcast searching process.

**Intermediate Node** These are the nodes that are between the sender node and the token holder node. Their main goal is to rebroadcast the received agents based on the selected broadcasting protocol.

**Token holder Node** This or these nodes hold the searched token and when an agent that is looking for it arrives to them they send the agent back to the sender.

**Basic course of events**

1. A node desires to get a token and selects a broadcasting protocol to search for it.
2. Creates an agent specific for that broadcasting protocol.

3. Broadcasts the agent to all the nodes within its cover area.
4. An intermediate node receives the agent.
5. The intermediate node runs the agent.
6. Agent searches in the node if it has the token.
7. As the node doesn't have the token the agent is rebroadcasted.
8. Agent jumps from intermediate node to intermediate node.
9. Agent arrives to a token holder node.
10. Token holder node runs the agent.
11. The token is found, agent updates its information and is send back to the sender.
12. Agent returns to the sender following back the same route it used to arrive to the token holder.
13. Agent arrives to the sender with the token and the simulation ends.

**Alternative paths**

**Isolated sender** After being broadcasted for the first time the agent cannot reach any node and the sender keeps waiting until the simulation ends.

**Network partition** The network gets partitioned and the sender and the token holder cannot communicate.

**Broken returning route** As nodes can be moving, after the agent has found the token it cannot return to the sender because one of the links it used is now broken.

**Congested network** If the network is very congested some transmissions can be lost and the agent might not be able to reach the token holder or return to the sender.

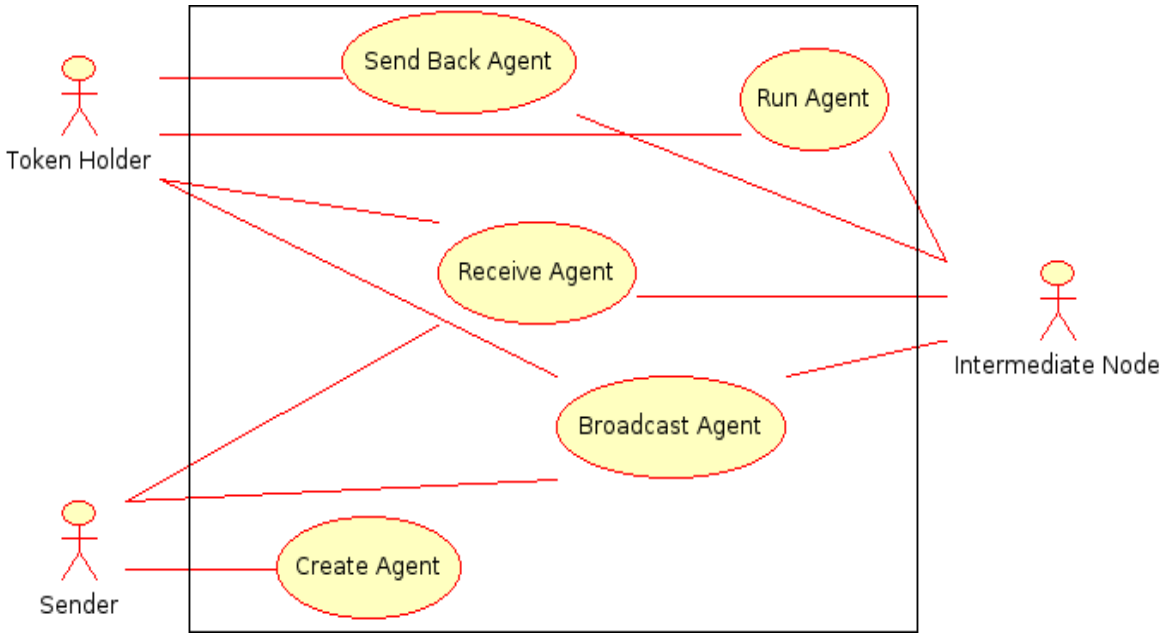


Figure 3.2: Searching Agent Use Case

### 3.3.2 General Class Diagram

Figure 3.3 shows the main classes of the application.

#### Simulation Class

This is the class that makes use of the classes and methods of SWANS API. It configures the simulation, selects the size of the field, the number of nodes and other parameters. These parameters are passed to the class from command line.

It creates and initialises all the nodes of the simulation and is also responsible of taking "snapshots" of the links between the nodes and their location into a log file.

#### AgentBaseSystem Class

A single instance of this class is created on each node. It receives parameters from the Simulation class to know how it has to configure the node, what kind of broadcasting protocol is going to be used, if the node has the token or is a sender, etc.

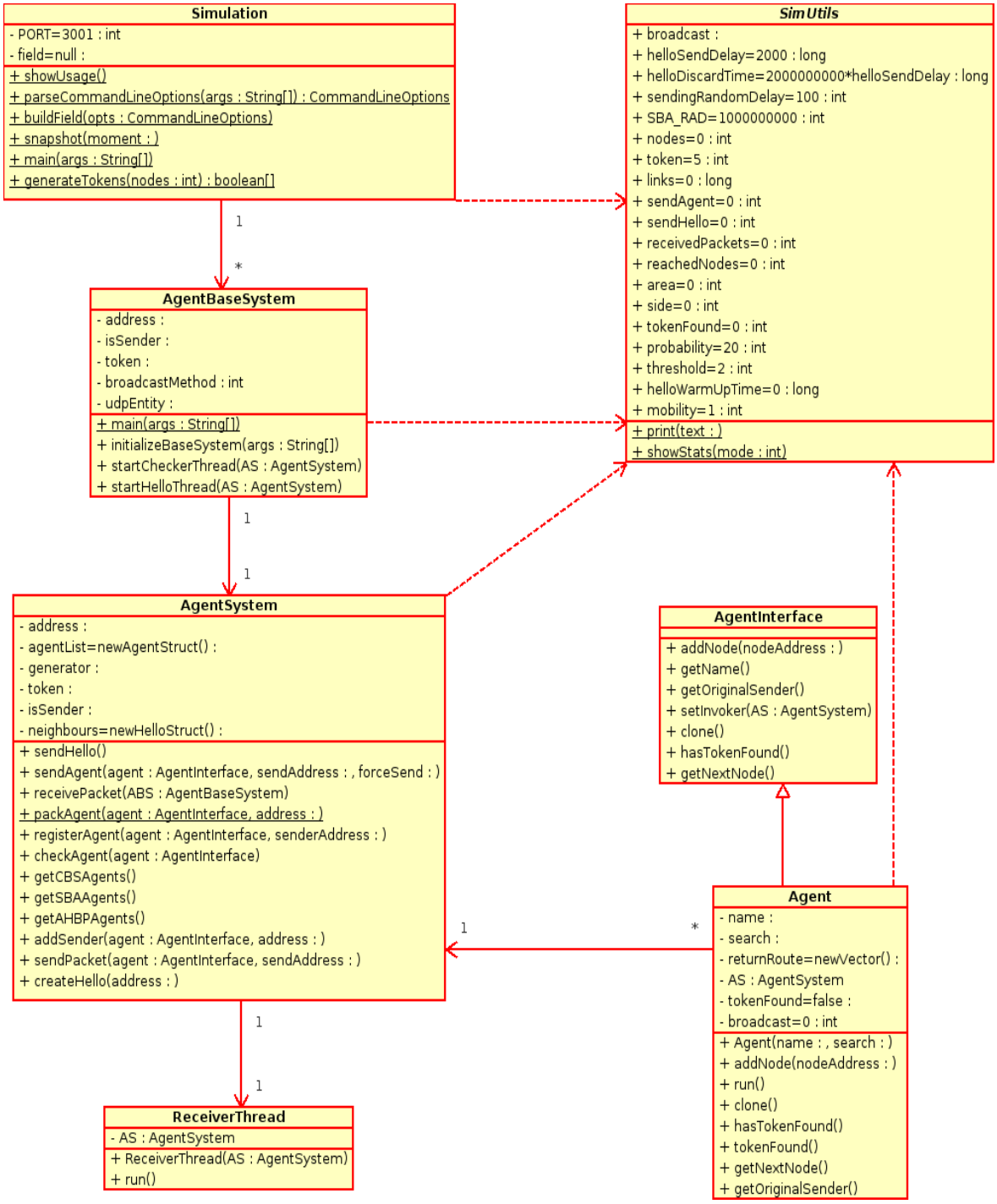


Figure 3.3: General Class Diagram

It starts all the needed threads. In the minimum configuration there is at least a receiver thread running.

### AgentSystem Class

This class has all the main methods for the Agent System. It provides services to the agents and to some threads. Some of the most important services are: sending and receiving packets, packing and unpacking agents, checking if the agents meet the needed preconditions to be rebroadcasted according to the current protocol, etc. It also creates the data structures used to store the needed information.

### ReceiverThread Class

This thread processes all incoming packets. Determines if they are agents or `HelloPacket`s and forwards the control to the appropriate thread.

### Agent Interface

Defines the common methods that all agents have to implement. This is a design decision that allows creating new agents without having to change the rest of the application.

### Agent Class

Implements the Agent Interface and is the agent used for the most simplest protocols. The rest of the agents used in the application extend this class.

### SimUtils Class

A static class used to store simulation configuration parameters and to gather information about running simulations.

### 3.3.3 Sequence Diagrams

#### Initialisation Sequence Diagram

Figure 3.4 explains the flow of execution in the initialisation of the simulation. When the Simulation class is called it instantiates and run as many copies of the AgentBaseSystem class as nodes the simulation has.

When the run method in this class is called it will create an Agent System -there is a single Agent System created per each node- and call the initializeReceiver method on the AgentSystem. This method will create a Receiver Thread and start it. From this moment the node is capable of listening to incoming packets.

The next step in the AgentBaseSystem is to check whether it is a sender node. If this is the case it will create an agent, the type of agent will be determined by the type of broadcasting protocol, and pass it to the AgentSystem to be broadcasted with the sendAgent method.

The AgentSystem will pack this agent and send the resulting packet with the sendPacket method. This packet can be received by several receiver threads from other nodes within cover area of the sender node, these are represented by ReceiverThread N.

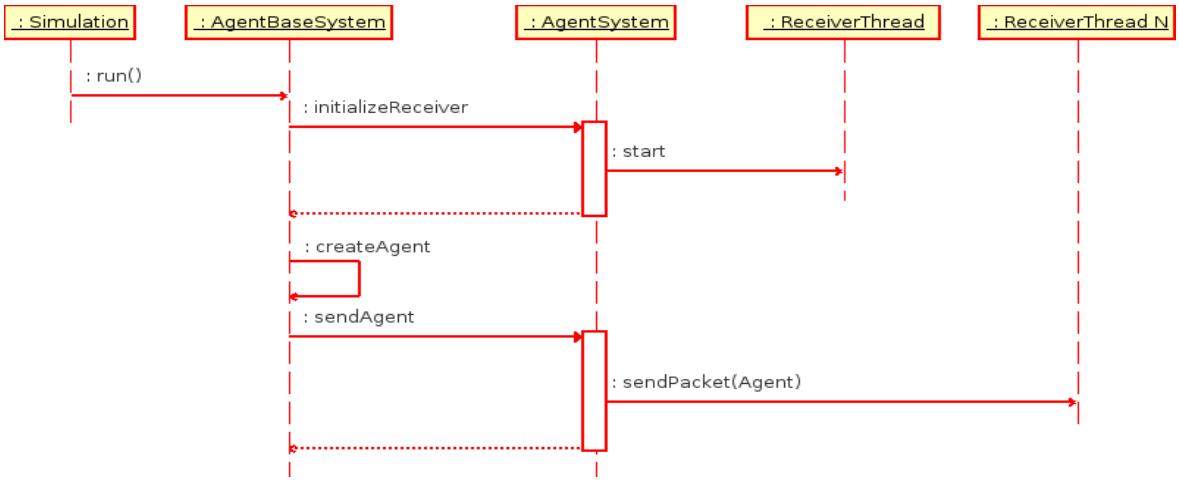


Figure 3.4: Initialisation Sequence Diagram

### Receive an agent Sequence Diagram

Figure 3.5 shows the basic process that is followed when an agent comes for the first time to a node. ReceiverThread class is in a continuous loop listening for incoming packets, once a packet arrives and it determines that the content is an agent it starts an AgentHandlerThread class to handle this agent.

Supposing that this is the first time the agent arrives to the node, the AgentHandlerThread registers the agent in the AgentSystem of the node. Then the agent is run on the node, it will search for the desired token and update itself with the information gathered.

Supposing that this node doesn't have the token, it will just rebroadcast the agent to the nodes around itself. The AgentHandler calls the sendAgent method of the AgentSystem, which will pack the agent and broadcast it with the sendPacket method.

As in the previous diagram, multiple receiver threads are represented by the receiverThread N class.

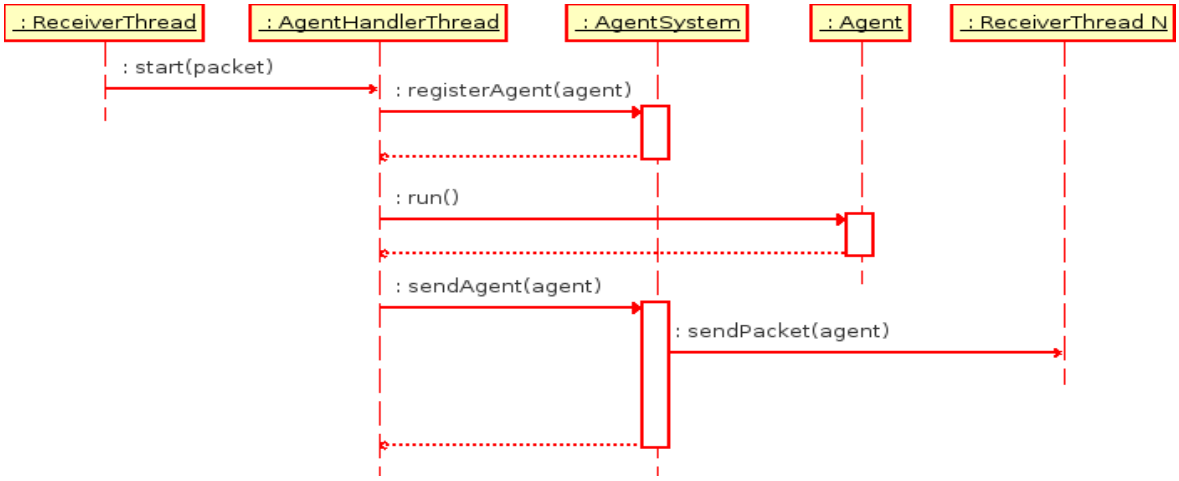


Figure 3.5: Receive an agent Sequence Diagram

## 3.4 Relevant Design Decisions

In this section some relevant design decisions will be explained. While designing and implementing the application some problems and unexpected issues were found and consequently the application design changed to adapt to these new requirements. In the following paragraphs some of these issues will be detailed.

### 3.4.1 UDP sockets

At the beginning of the development of the application tests were done with a small number of nodes but soon was noticed that when two nodes were transmitting at the same time only one of the messages was received by the nodes within cover range of both.

Trying to fix this behaviour several alternatives where considered. The first improvement that was made was to create a listener thread which would be only responsible of receiving a packet from the DatagramSocket and forward it to another thread to handle it. This way the time that the node was listening was maximised and less packets were loss. However, in heavily populated network conditions there was still a high number of packets being discarded.

Next improvement considered was to create a pool of listener threads, trying to reduce the possibility of a node not listening when a packet was being transmitted. Although this approach works fine with TCP transmissions it doesn't make any sense in UDP communications because of its connectionless nature. UDP is unreliable, there are no acknowledges (ACK) nor negative-acknowledge (NACK) and even if this was possible it would be very resource expensive for the simulation to have a high number of threads running per node.

There are basically three things that can be done to partially solve this problem:

1. Send the packets more slowly, introducing a random delay.

2. Have a large receive buffer on the receiving side.

3. Have a large send buffer on the sending side.

None of these solve completely the problem, but they improve it substantially. Due to implementation details of the simulator there aren't any buffers implemented in the

DatagramSocket class so solutions two and three couldn't be applied for the simulation, but should be added for real world testing.

Consequently the adopted solution was to introduce a random delay on each broadcasted packet. So, finally, when a packet is received there is a single thread per node to handle it and when a packet is send a random delay is introduced to avoid a node's transmission to coincide with others, which is very likely to happen when broadcasting.

# Chapter 4

# Implementation

In this chapter implementation details of the application are described. The first section shows how the application is executed and how it can be configured in different ways. In the next section, SWANS network protocol stack is detailed; how to configure a simulation, how to create a node or how to run a custom application on each node are also explained. In the last section the Agent System application's classes are introduced.

## 4.1 Launching the simulation

As explained in 3.2 the simulation is a Java application running in top of SWANS. SWANS is a Java application running atop JiST, and JiST is another Java application running inside a JVM. So the way to call the application is the following:

```
java jist.runtime.Main jist.swans.Main nds.Simulation
```

Each Java application is the parameter for the Java application that has to run it. The nds.simulation class is also prepared to receive simulation configuration parameters from command line. These are the parameters that the application is prepared to receive:

- `-h, --help`

  Prints the help message, showing the available parameters.

- `-e, --endat`

  Simulation ending time, default is set to infinite.

- `-p, --protocol`

  The routing protocol that is going to be used. Can choose between ZRP, AODV and the default one, DSR.

- `-n, --nodes`

  The number of nodes that will be used in the simulation. Default is 100.

- `-b, --broadcast`

  Choose between different available broadcasting protocols. 0, 1, 2, 3, 4.

  **0** Simple Flooding, this is the default one.

  **1** Probabilistic Scheme.

  **2** Counter-Based Scheme.

  **3** Scalable Broadcast Algorithm (SBA).

  **4** Ad Hoc Broadcast Protocol.

- `-%, --probability`

  This parameter is only used in Probabilistic Scheme. It specifies the probability to rebroadcast. Default is 20%.

- `-t, --threshold`

  This parameter is only used in Counter-Based Scheme. Default value is a threshold of two.

- `-c, --tokens`

  It allows to change the percentage of nodes that will have the token. Default is a 5%.

- `-d, --helloDelay`

  This allows to change the periodicity of sending "Hello" packets. Only used in Neighbour Knowledge methods: Scalable Broadcast Algorithm and Ad Hoc

Broadcast Protocol. By default it will send a "Hello" packet every 2 seconds (2000 milliseconds).

- `-f, --field`

  Used to choose the field dimensions in meters. It expects to values in the following way:[x,y]. Default is a field of 50x50.

- `-a, --arrange`

  This parameter changes the way the application uses to place the nodes inside the field. It can be randomly or in a grid. The default value is to do it randomly.

- `-m, --mobility`

  Selects the mobility model. Two are supported: static and waypoint. By default static mobility is used. In static model nodes are not moving and in waypoint model nodes pick a random "waypoint" and walk towards it with some random velocity, then pause and repeat.

The following example is for a simulation of an Ad Hoc Broadcast Protocol, with mobility waypoint. There are 10,000 nodes in an area of 10000x10000 meters with a "Hello" packet every 2 seconds.

```
java jist.runtime.Main jist.swans.Main nds.Simulation -b 4 -m waypoint
-n 10000 -f 10000,10000 -d 2000
```

## 4.2 SWANS: The network protocol stack

SWANS has a whole set of classes already implemented to allow the user to create a complete TCP/IP network stack. The programmer can choose from different implementations for some of the protocol layers to customise the networking behaviour of its application.

Figure 4.1 shows graphically how these classes interact with each other to create the simulation environment.

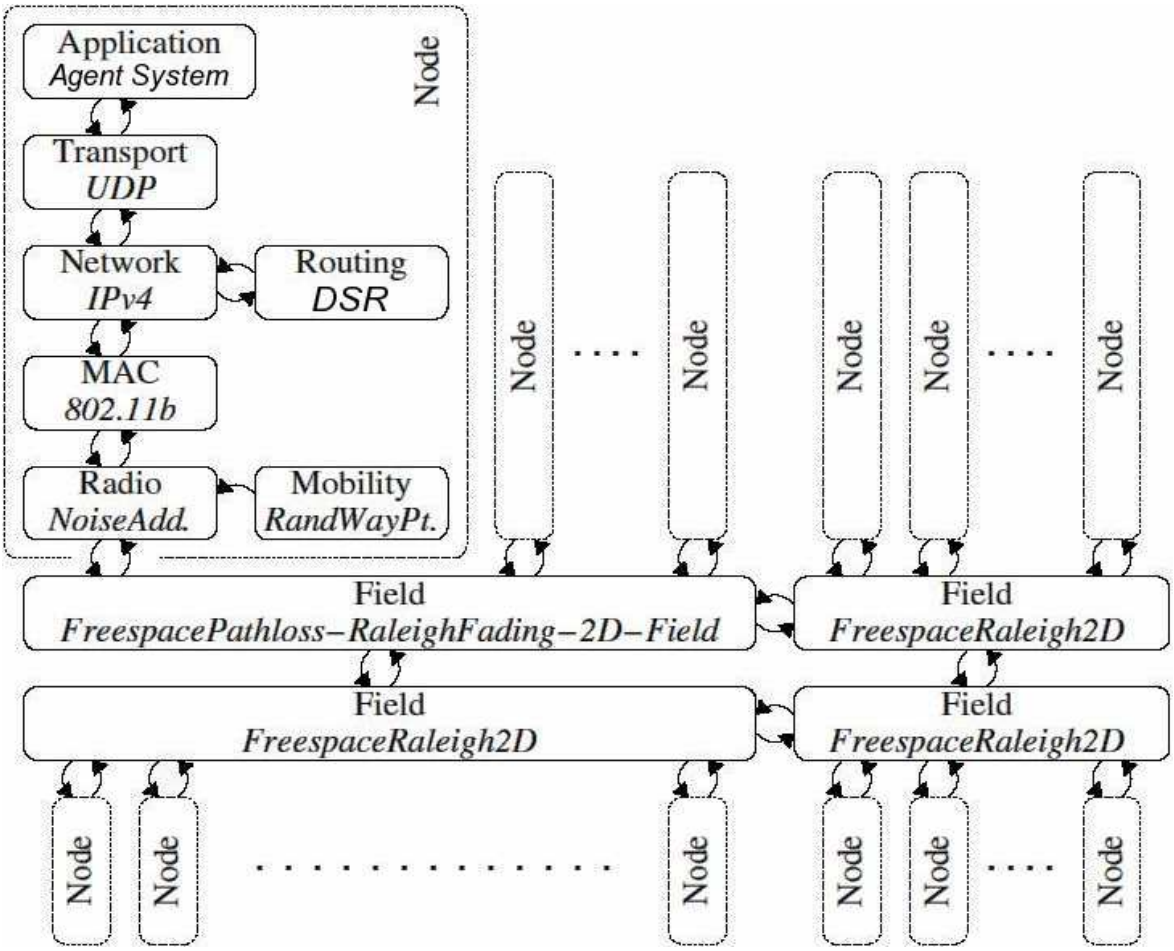


Figure 4.1: SWANS network protocol stack

### 4.2.1 Creating the simulation Field

The simulation is initialised in the Simulation class. The Field class acts as a container for the rest of the classes that are used in the simulation, as can be seen in figure 4.1. After processing all the command line parameters the first thing that needs to be done is to initialise the mobility model.

```
Mobility mobility = null;
switch (opts.mobility) {
        case Constants.MOBILITY_STATIC:
                mobility = new Mobility.Static();
                break;
        case Constants.MOBILITY_WAYPOINT:
                mobility = new Mobility
                .RandomWaypoint(opts.field, PAUSE_TIME,
                 GRANULARITY, MIN_SPEED, MAX_SPEED);
                break;
}
```

The mobility is then used to initialise the Field instance that the Simulation class has as an attribute.

```
if (opts.mobility == 1) {
        field = new Field(opts.field, false);
}
else {
        field = new Field(opts.field, mobility);
}
```

Once the Field is created with the mobility information a RadioInfo instance is created. RadioInfo is used to represent the information possibly shared among numerous Radio instances. It would substitute the physical radio channel fof real world.

```
// initialise shared radio information
```

```
RadioInfo.RadioInfoShared radioInfo = RadioInfo.createShared(
        Constants.FREQUENCY_DEFAULT, Constants.BANDWIDTH_DEFAULT,
        Constants.TRANSMIT_DEFAULT, Constants.GAIN_DEFAULT, Util
        .fromDB(Constants.SENSITIVITY_DEFAULT), Util
        .fromDB(Constants.THRESHOLD_DEFAULT),
        Constants.TEMPERATURE_DEFAULT,
        Constants.TEMPERATURE_FACTOR_DEFAULT,
        Constants.AMBIENT_NOISE_DEFAULT);
```

As the parameters being passed show, the radio channel can be configured in many different ways. The next step is the Placement model. As seen in 4.1 placement can be random or grid. This Placement class will be used later to give its location to each node.

```
// initialise node placement model
Placement place = null;
switch (opts.placement) {
case Constants.PLACEMENT_RANDOM:
        place = new Placement.Random(opts.field);
        break;
case Constants.PLACEMENT_GRID:
        place = new Placement
                .Grid(opts.field, opts.placementOpts);
        break;
default:
        throw new
        RuntimeException("unknown node placement model");
}
```

And finally in which nodes the tokens will be is calculated prior to the creation of the nodes themselves.

```
// generate the tokens randomly
boolean[] tokens = generateTokens(opts.nodes);
```

At this point there is a simulated field created, with a shared radio channel and mobility and location models initialised. This is the moment to start creating the nodes that will be inside this field.

### 4.2.2 Creating a node

The whole network stack has to be created on each node. As can be seen in figure 4.1 the lower level is the physical layer, which in this case is represented by the Radio class. In this application the RadioNoiseIndep model has been used, which is a radio with an independent noise model. The other possibility would be an additive noise model. The independent interference model considers only signals destined for the target radio as interference. The additive model considers all signals as contributing to the interference. Both radios are half-duplex, as in 802.11b.

```
// radio
RadioNoise radio = new RadioNoiseIndep(i, radioInfo);
```

The following layer is the Data Link layer. As this is a wireless simulation it is implemented in a 802.11 entity. SWANS includes IEEE 802.11b and a "dumb" protocol. The 802.11b implementation includes the complete DCF functionality, with retransmission, NAV and backoff functionality. The "dumb" link entity will only transmit a signal if the radio is currently idle. This simulations have been run using the regular 802.11b implementation.

```
// mac
Mac802_11 mac = new Mac802_11(new MacAddress(i),
                              radio.getRadioInfo());
```

After the Data Link layer the Network layer is created and initialised. In this case IPv4 is being used. First a NetAddress is created, based on the node number; node 1 will have address `0.0.0.1`. The network entity is then created using an instance of the class NetIp.

```
// network
final NetAddress address = new NetAddress(i);
NetIp net = new NetIp(address, protMap, loss, loss);
```

At the same level of the network layer SWANS creates a routing entity. The routing entity receives upcalls from the network entity with packets that require next-hop information. It sends downcalls to the network entity with next-hop information when it becomes available. SWANS implements the Zone Routing Protocol (ZRP), Dynamic Source Routing (DSR) and Ad hoc On-demand Distance Vector Routing (AODV). In this simulations DSR has been used as it was the available simplest one and broadcasting doesn't require of any routing, it is only being used by packets returning to the sender.

```
// routing
RouteInterface route = null;
switch (opts.protocol) {
case Constants.NET_PROTOCOL_DSR:
        RouteDsr dsr = new RouteDsr(address);
        route = dsr.getProxy();
        dsr.setNetEntity(net.getProxy());
        break;
case Constants.NET_PROTOCOL_AODV:
        ...
        break;
case Constants.NET_PROTOCOL_ZRP:
        ...
        break;
default:
throw new RuntimeException("invalid routing protocol");
}
```

Finally the transport layer is created. As this application is created to research broadcasting it will only use UDP communications. This is implemented in the TransUdp

class. There is a TransTcp class available for TCP communications. This completes the network stack and leaves the node almost ready to start sending and receiving UDP packets.

```
// transport
TransUdp udp = new TransUdp();
```

There are a few simulator specific tasks to be performed before the Agent System can be run on the node. First the node is provided with a location in the field, based on the placement model selected and initialised previously.

```
// placement
Location location = place.getNextLocation();
```

Then the radio device of the node is added to the field.

```
field.addRadio(radio.getRadioInfo(), radio.getProxy(), location);
```

If there is any mobility it is initialised on the node.

```
if (opts.mobility == Constants.MOBILITY_WAYPOINT) {
        field.startMobility(radio.getRadioInfo()
        .getUnique().getID());
}
```

### 4.2.3 Running the Agent System

At this point the simulation is ready to run any Java application on each node. First thing to do is to check if the node has the token. As explained in 4.2.1 as a result of the execution of generateTokens() method, an array that specifies which nodes have a token is created.

```
boolean token = tokens[i - 1];
```

The following lines show the way this simulator runs embedded applications. It uses a AppJava class which receives the main class of the application that wants to be run as a parameter. Next the UdpEntity is binded to the application to be run. UdpEntity is the class that stores all the needed information for an application to access networking services.

As a normal Java application, parameters can be passed to AppJava. For this simulation application the node address, if the node is the sender, if it has a token and what kind of broadcasting it is going to be used are passed as parameters. Finally the application's main method is called. From this moment the flow of execution follows in the Agent System application.

```
try {
        AppJava app = new AppJava(AgentBaseSystem.class);
        app.setUdpEntity(udp.getProxy());
        String[] parameters = new String[] { address.toString(),
                String.valueOf(isSender), String.valueOf(token),
                Integer.toString(opts.broadcast) };
        app.getProxy().run(parameters);
} catch (NoSuchMethodException e) {
        e.printStackTrace();
}
```

## 4.3 Agent System

In this section all the classes of the Agent System application will be commented. The most relevant methods will also be briefly explained.

**AgentBaseSystem** This static class is responsible of initialising all the needed threads on the node, depending on the broadcasting protocol that is going to be used. It also creates the original agent in the sender and sends the first broadcasting.

| | |
|---|---|
| *Main* | Creates an instance of `AgentSystem` and initialises it. Calls the `initializeReceiver` method on the `AgentSystem` instance. If the node is the sender creates the appropriate agent type and broadcasts it. Finally it runs, if needed, the Checker and/or Hello threads. |
| *initializeBaseSystem* | Reads the received parameters and stores them in class attributes. |
| *startCheckerThread* | Called when the broadcasting method is `Counter Based Scheme`, `Scalable Broadcast Algorithm (SBA)` or `Ad Hoc Broadcast Protocol`. It creates an instance of `CheckerThread` and executes it. |
| *startHelloThread* | Called when the broadcasting method is `Scalable Broadcast Algorithm (SBA)` or `Ad Hoc Broadcast Protocol`. It creates an instance of `HelloThread` and executes it. |

Table 4.1: AgentBaseSystem Class

**AgentSystem** This class is the kernel of the application. It has the data structures that are used on the system, provides with networking services to the rest of the classes and also has some broadcasting protocol specific methods.

| | |
|---|---|
| *sendHello* | This method waits for a random time before creating a `HelloPacket`, opening a UDP `DatagramSocket` and broadcasting it. It is called regularly from the `HelloThread` |
| *sendAgent* | This method receives an agent as a parameter and calls the `sendPacket` method to broadcast it. If the broadcasting method is Probabilistic Scheme a random number is calculated before sending. If it falls below the probability the packet is discarded. |
| *initializeReceiver* | Creates an instance of the `ReceiverThread` class and starts it. |
| *packAgent* | Receives an agent and a `InetAddress`, transforms the agent into a stream of bytes and creates a `DatagramPacket` that will send the agent to the specified `InetAddress`. |
| *registerAgent* | This method registers the agent when it is received for the first time in the `AgentStruct` data structure that every node has. In those protocols where a RAD is used, its sending time is calculated at this moment and stored with the agent in the `AgentStruct`. |
| *getCBSAgents* | Goes through the `AgentStruct` of the node to check what CBS agents are ready to be send and returns them in a `Vector`. |
| *getSBAAgents* | Goes through the `AgentStruct` of the node to check what SBA agents are ready to be send and returns them in a `Vector`. |
| *getAHBPAgents* | Goes through the `AgentStruct` of the node to check what AHBP agents are ready to be send and returns them in a `Vector`. |
| *sendPacket* | Receives an agent, packs it into a `DatagramPacket` and sends it. |
| *createHello* | Creates a "Hello" `DatagramPacket` with nodes current neighbour information |

Table 4.2: AgentSystem Class

**AgentStruct** This class is used to store information about agents in the nodes. For each agent it stores the agent itself, the number of times it has been received, when it is scheduled to be send -depending on the RAD-, if it has been already send and from what nodes it has been received.

**ReceiverThread** This thread is called on the initialisation of each node. It is continuously running, waiting for incoming packets.

| | |
|---|---|
| *run* | This thread is in an infinite loop waiting for incoming datagrams. Once a datagram is received it checks if it is a `HelloPacket` or an agent. If it is a `HelloPacket` it creates a `HelloHandlerThread` to handle it. If it is an agent it creates an `AgentHandlerThread` and forwards it the agent. |

Table 4.3: ReceiverThread Class

**AgentHandlerThread** This thread is responsible of handling incoming agents.

| | |
|---|---|
| *run* | It checks if it is a searching or a returning agent. If it is still searching and it is the first time it has been received it runs it. If the agent is returning to its sender the thread calls the `returnAgent` method to send the agent to the next node. |
| *returnAgent* | Receives an agent, gets from its returning route the next node it has to go to and sends the agent to that node. |

Table 4.4: AgentHandlerThread Class

**CheckerThread** This thread is only used in Counter-Based Scheme, Scalable Broadcast Algorithm (SBA) and Ad Hoc Broadcast Protocol. It is continuously running to check if there is any agent ready to be send.

| | |
|---|---|
| *run* | It executes an infinite loop that checks every ten milliseconds if there is any agent ready to be send. Makes use of the `getCBSAgents`, `getSBAAgents` and `getAHBPAgents` methods of the `AgentSystem` class. |

Table 4.5: CheckerThread Class

**HelloPacket** It is just a `Vector` with all the addresses of the neighbours of a given node.

**HelloThread** Used only in neighbour knowledge methods: Scalable Broadcast Algorithm (SBA) and Ad Hoc Broadcast Protocol. It sends a `HelloPacket` regularly with the information about neighbours that the `AgentSystem` class has in its `HelloStruct`.

**HelloStruct** This class is used to store information about neighbours. For each neighbour it keeps information about its address and when it was the last time a HelloPacket came from it.

| | |
|---|---|
| *getBRG* | This is used in the Ad Hoc Broadcast Protocol. This method receives a `Vector` with the nodes that were reached in the transmission through which the agent was received and calculates which neighbours will be marked as `Broadcast Relay Gateway (BRG)`; which ones will be marked to be rebroadcasters. |

Table 4.6: HelloStruct Class

**HelloHandlerThread** This thread is called from the `ReceiverThread` each time a `HelloPacket` arrives. It updates the information of the `HelloStruct` in the `AgentSystem` and removes outdated information.

**AgentInterface** This interface defines the minimum set of methods that any agent has to implement. The whole Agent System application is programmed in a way that agents are interchangeable. The methods are basically for getting information

about what broadcasting method the agent is using, updating its route, cloning the agent or getting information about which node send it.

**Agent** This is the main agent. The rest of the agents extend this class. It has a `Vector` where it stores information of all the nodes it has gone through. It also has a boolean that represents if it has found the token. It stores in an integer the broadcasting protocol that is willing to use. This agent is used for `Simple Flooding` and `Probabilistic Scheme`.

| | |
|---|---|
| *run* | Checks in the `AgentSystem` in which it is located if it has the token and runs the `tokenFound` method if the `AgentSystem` has the token. |
| *tokenFound* | Saves the followed route to the token holder node. Changes the broadcasting method to `Simple Flooding` for returning to the sender. When an agent is returning to the sender instead of being broadcasted is send only to the next node in its returning route. In this case all the checks and considerations that other broadcasting methods do are not needed. |
| *getNextNode* | Returns the address of the next node to go when an agent is returning to its sender. |

Table 4.7: Agent Class

**AgentCBS** This agent is used for the `Counter-Based Scheme (CBS)`, extends `Agent` class. It adds information about the threshold the agent is configured to use and the maximum time its RAD can be.

**AgentSBA** This agent is used for the `Scalable Broadcast Algorithm (SBA)`. It doesn't add any extra functionality apart from setting the correct broadcasting method.

**AgentAHBP** This agent is used for the `Ad Hoc Broadcast Protocol`, extends the

`Agent` class. Adds a `Vector` for passing information about the `Broadcast Relay Gateway (BRG)` to the receivers.

**SimUtils** This is a static class which is mainly responsible of gathering all the information that the simulation is generating.

| | |
|---|---|
| *showStats* | This method makes some calculations on the statistics and displays them. |

Table 4.8: SimUtils Class

# Chapter 5

# Evaluation

This chapter contains the evaluation of the experiments. In the first section the experiments are described. The following sections analyse the experiment's results. This analysis are focused on congestion, searching speed and number of packets used when broadcasting.

## 5.1 Design of the experiments

As explained in 4.1, this application allows multiple parameters to be defined. Table 5.1 shows the selected parameter set common to all implemented broadcasting methods.

| Set of selected parameters | | | | | |
|---|---|---|---|---|---|
| `Mobility` | | Static | Waypoint | | |
| `Number of nodes` | | 20 | 100 | 1000 | 10000 |
| `Field Size` | | 1000x1000 | 5000x5000 | 10000x10000 | |

Table 5.1: Set of selected parameters

As the number of nodes that have the token is a 5% of the total, the minimum number of nodes chosen was twenty, so there would always be a node with the token. The

next chosen value is a hundred and the following ones were chosen to rise exponentially.

A similar approach was taken with the field size. Three different sizes, from a thousand by a thousand meters field to a ten thousand by ten thousands one.

| Number of experiments | |
|---|---|
| `Simple Flooding` | 72 |
| `Probabilistic Scheme` | 288 |
| `Counter-Based Scheme` | 216 |
| `Scalable Broadcast Algorithm (SBA)` | 144 |
| `Ad Hoc Broadcast Algorithm` | 144 |
| `Total number of experiments` | 864 |

Table 5.2: Number of experiments

These numbers represent that for each broadcasting method at least twenty four experiments have been run. Due to the random placement that the token has each experiment has been repeated three times in order to get average statistics of each scenario. So for the simplest broadcasting protocol seventy two experiments have been produced.

Replicating the experiments done by Sze-Yao et al. in [21], the Probabilistic Scheme has been tested with probabilities of 20, 40, 60 and 80%. So for the Probabilistic Scheme two hundred and eighty eight experiments have been executed.

For the Counter-Based Scheme threshold values of two, four and six have been selected, as values higher than six do not add a significant improvement[21]. This makes a total of two hundred and forty four experiments.

For the Scalable Broadcast Algorithm (SBA) and Ad Hoc Broadcast Protocol a

"Hello" packet delay of 2 and 4 seconds have been used. For each of these two protocols one hundred and forty four experiments have been run.

The overall number of experiments is eight hundred and sixty four, as table 5.2 shows.

The combination of these field sizes and node numbers allow for different node densities, which is useful for studying network congestion. Table 5.3 shows these different node densities per a hundred square meters.

| Node densities | | | | |
|---|---|---|---|---|
| | 20 nodes | 100 nodes | 1,000 nodes | 10,000 nodes |
| 1,000x1,000 | 0.2 | 1.0 | 10.0 | 100.0 |
| 5,000x5,000 | 0.008 | 0.04 | 0.4 | 4.0 |
| 10,000x10,000 | 0.002 | 0.01 | 0.1 | 1.0 |

Table 5.3: Nodes per a hundred square meters

## 5.2 Experiment Results

In the following table 5.4 the number that represents each protocol in the rest of the graphs of this chapter can be seen.

### 5.2.1 Congestion Analysis

The objective of this subsection is to analyse how the packet delivery percentage evolves in the selected scenarios, how the network gets more congested when the number of nodes per a hundred square meters increase. This experiment shows how the network performs under the most aggressive broadcasting protocols: Simple Flooding. Based

| Protocol representation | |
|---|---|
| Protocol number | Protocol type and configuration |
| 0 | Simple Flooding |
| 1 | Probabilistic Broadcast 80% |
| 2 | Probabilistic Broadcast 60% |
| 3 | Probabilistic Broadcast 40% |
| 4 | Probabilistic Broadcast 20% |
| 5 | Counter Based Method - Threshold = 6 |
| 6 | Counter Based Method - Threshold = 4 |
| 7 | Counter Based Method - Threshold = 2 |
| 8 | Scalable Broadcast Algorithm (SBA) - "Hello" send delay = 2 sec. |
| 9 | Scalable Broadcast Algorithm (SBA) - "Hello" send delay = 4 sec. |
| 10 | Ad Hoc Broadcast Protocol - "Hello" send delay = 2 sec. |
| 11 | Ad Hoc Broadcast Protocol - "Hello" send delay = 4 sec. |

Table 5.4: Protocol representation

on the data from table 5.3, graph 5.1 shows how the packet delivery percentage evolves in relation with node density.

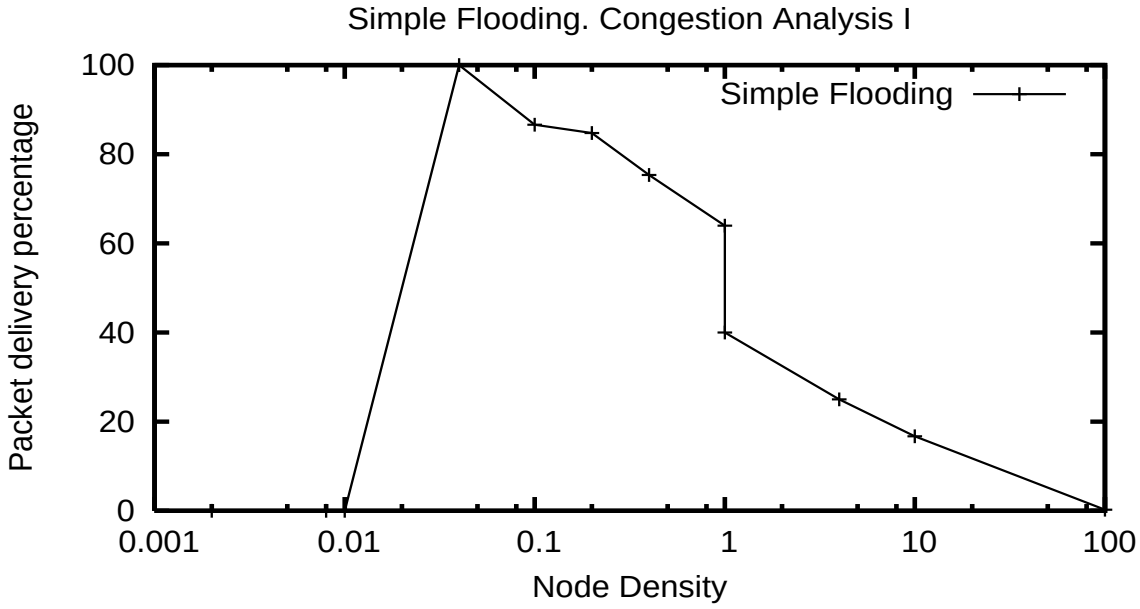


Figure 5.1: Simple Flooding. Congestion Analysis I

Figure 5.1 shows node densities per a hundred square meters. The graph shows that there is not any packet delivery with node densities below 0.01, as nodes are not connected with each other.

As it seems logical, the least congested network is that one that has the minimum number of nodes that can successfully communicate between themselves. This graph indicates that with a density of approximately 0.04 the network works at full performance.

Once the density of nodes increases the performance starts decreasing. When node density is a hundred, which represents ten thousand nodes in a 1,000x1,000 meters area, network performance decreases to almost 0. Congestion of the network is almost maximum.

As can be seen in table 5.3, there are two different scenarios where node density is 1.0; a hundred nodes in a 1,000x1,000 meters area and ten thousand nodes in a 10,000x10,000 meters area. Event though both have the same node density these two

scenarios are very different. This can be observed in graph 5.1, where for node densities 1.0 there are two different values. In fact, node density is not the factor that influences more the evolution of network congestion. For a more exact analysis the number of links per node must be used.

Figure 5.2 shows how the network gets more congested when the number of links per node increases. In this graph the effect that appeared with node density 1.0 disappears to show a more linear graph.

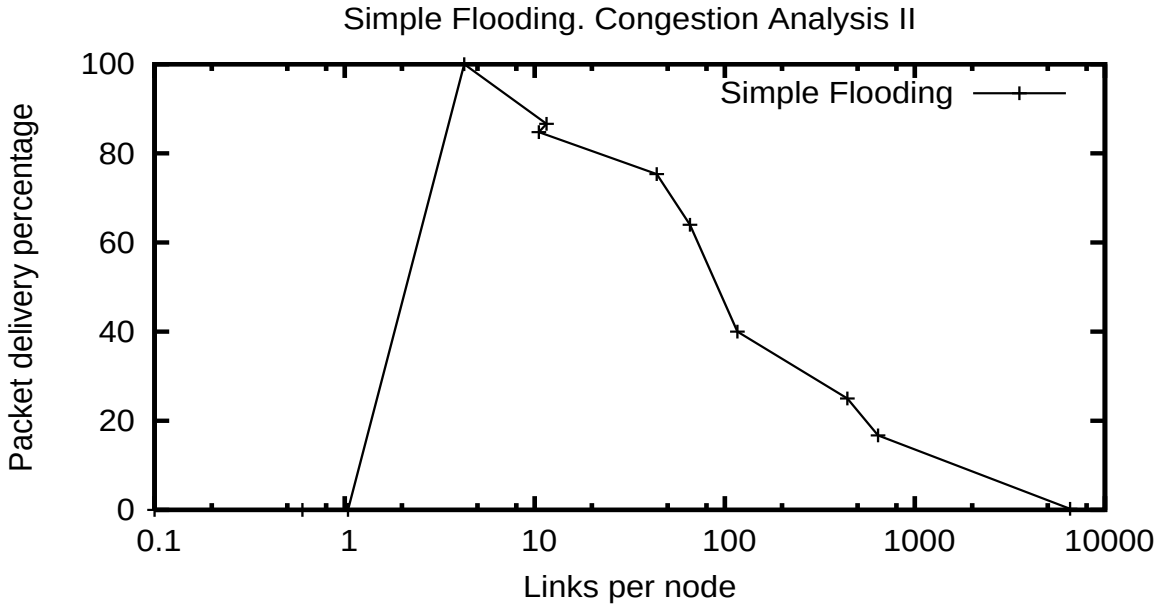


Figure 5.2: Simple Flooding. Congestion Analysis II

Finally, to end with congestion analysis, figure 5.3 shows how different protocols perform in a typical scenario; one thousand nodes in a 10,000x10,000 meters area. Protocols are numerated based on the information from table 5.4.

Graph 5.3 shows how the packet delivery percentage increases from around a 50% with Simple Flooding -protocol 0- to an 80% with Probabilistic Method at a 20% -protocol 4- of rebroadcasting probability. The counter based scheme -protocols 5,6,7- shows a similar performance in all its different configurations. Neighbour Knowledge methods -protocols 8 to 11- vary from 90% to 70% of performance.

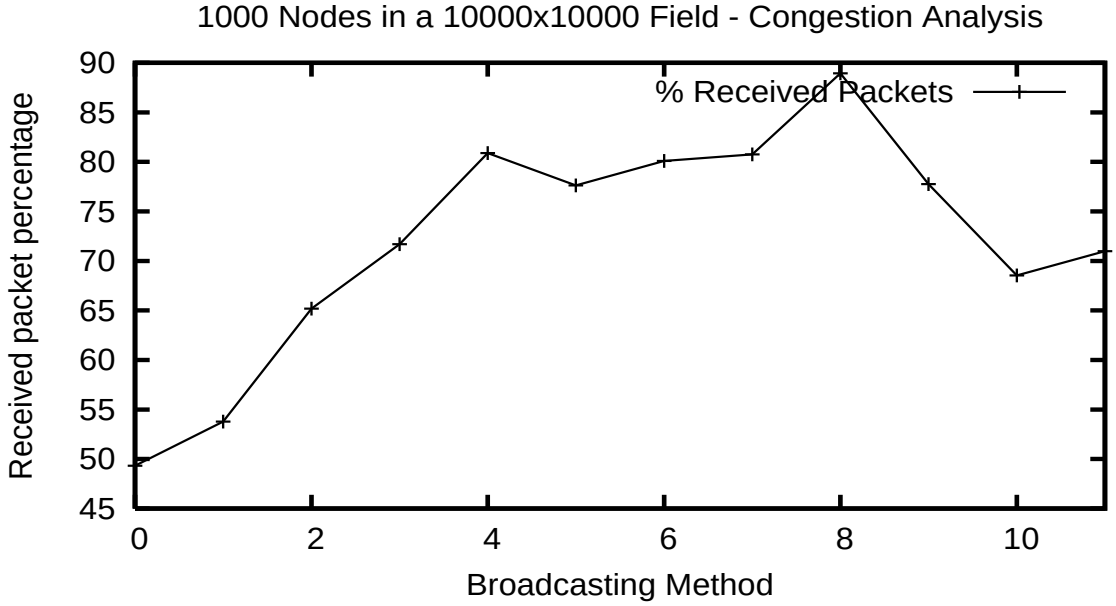


Figure 5.3: Simple Flooding. Congestion Analysis III

### 5.2.2 Search speed analysis

This subsection shows how much time each protocol needs to find a node that holds the token. This time starts when the agent is created and ends when a copy of the agent reaches a node with the token.

As explained before there are a 5% of nodes with a token. In this analysis network congestion considerations are not taken in account. For Neighbour Knowledge methods there is a network "warm-up" time during which "Hello" packets are exchanged several times until network topology around each node is completely known by each of them. In this case the agent is not created and broadcasted for the first time until this time has passed. This "warm-up" time has been subtracted from this analysis as in real world it would also take place before the creation of the agent and its first broadcast.

For this analysis two different area sizes have been studied, a 1000x1000 meters area, described in figure 5.4 and a 10,000x10,000 meters area, described in figure 5.5.

Graph from figure 5.4 shows how fast each protocol performs with different network populations. Populations of one hundred nodes and above in an area like this will always find the token in the first broadcast, as each node is linked with around a 60%

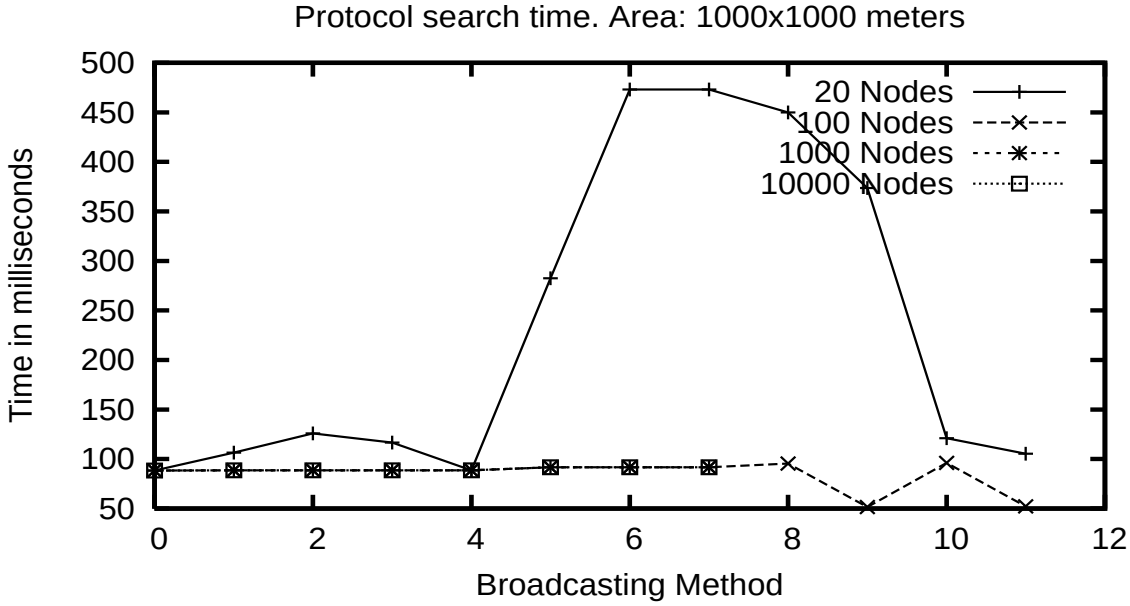


Figure 5.4: Speed Analysis. 1000x1000

of the rest of the nodes, as will be explained in 5.2.3. The most valuable data from this graph is the twenty nodes one. In this case protocol performance can be divided in two main groups, those protocols that find the node in around one hundred milliseconds and those that need three hundred or more milliseconds. Protocols 0,1,2,3,4,10 and 11 are in the first group and 5,6,7,8 and 9 in the second one. Protocols from the first group are those that don't use a RAD in their algorithm, from the Simple Flooding -protocol 0- to the advanced Ad Hoc Broadcast Protocol -protocol 11- none of them uses a RAD. All protocols in the second group use a RAD.

Figure 5.5 shows how fast the same protocols perform in a much bigger area, of a 10,000x10,000 meters. In this case only one thousand and ten thousand node populations are shown, as smaller populations are not linked due to the low node density. There are not values for ten thousand nodes in the most advanced protocols due to the testing machine hardware resources. However, one thousand nodes line is quite relevant.

As it was explained in the previous graph, there are two different groups of protocols. In this example groups are divided in the same way if the one thousand nodes line is

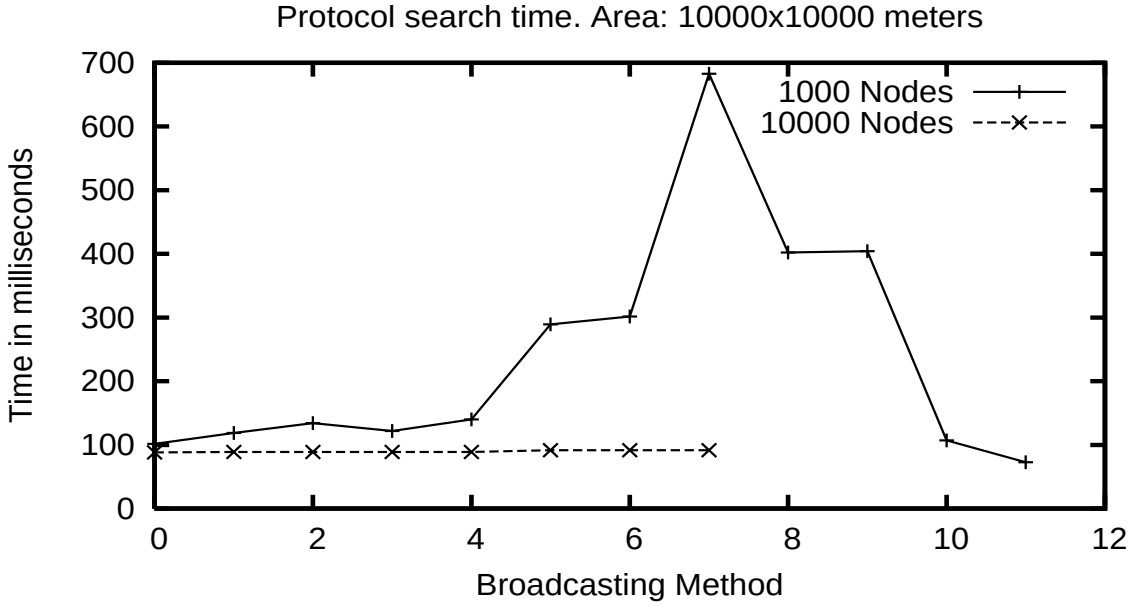


Figure 5.5: Speed Analysis. 10000x10000

observed. Protocols that don't use a RAD find the token in around one hundred milliseconds and the rest of the protocols need three hundred milliseconds or more.

The fact that a neighbour knowledge protocol like Ad Hoc Broadcast Protocol -protocol 11- performs as fast as Simple Flooding -protocol 0- while congesting the network much less is a very interesting improvement over other advanced protocols.

### 5.2.3 Broadcasting analysis

The goal of this subsection is to analyse how many packets each protocol needs to broadcast until it finds a node that holds the token. For this analysis "Hello" packets from neighbour knowledge methods have not been taken in account, as usually any search finishes before a new set of "Hello" packets has been send, which is every two or four seconds.

Figure 5.6 shows the number of packets each protocol needs in an area of 1,000x1,000 meters. Results vary from one to three packets and it doesn't seem to be any identifiable pattern. It appears that Simple Flooding and Probabilistic methods are more variable while more advanced protocols trend to use less number of packets.

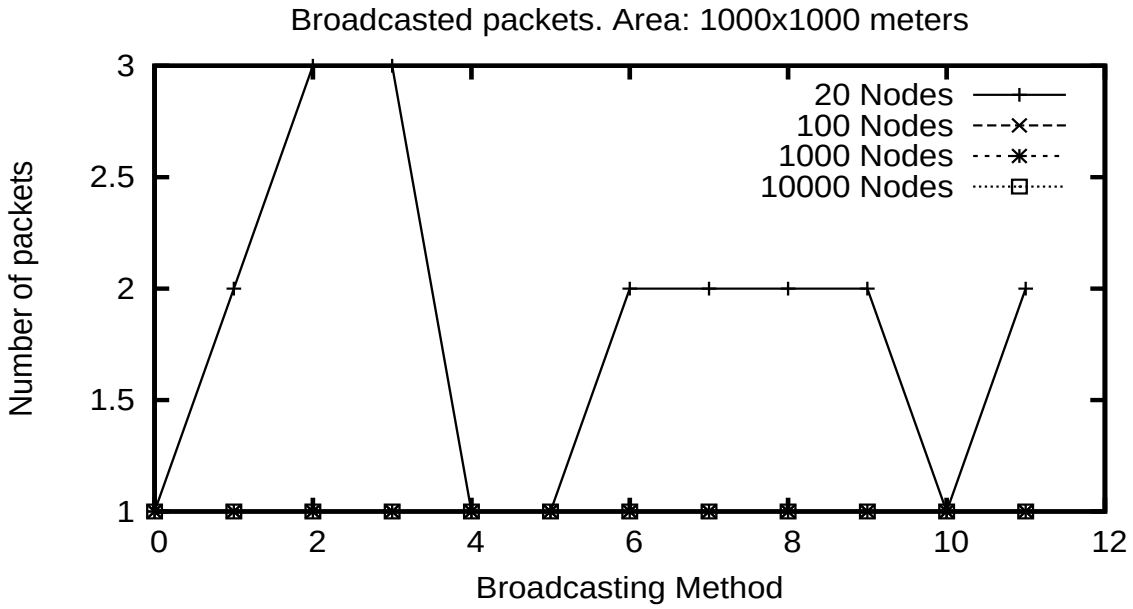


Figure 5.6: Broadcasting Analysis I. 1000x1000

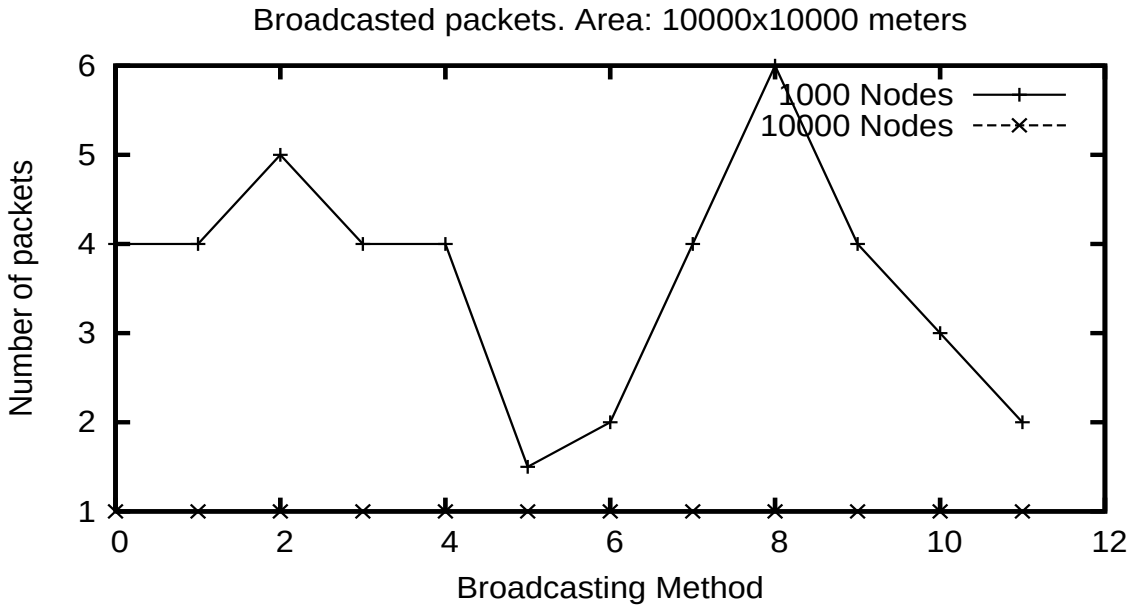


Figure 5.7: Broadcasting Analysis I. 10000x10000

Figure 5.7 is from an area of 10000x10000 meters. Again there is not a very clear pattern, although more basic protocols like Simple Flooding and Probabilistic methods don't broadcast less than four packets. More advanced methods usually perform better, using a lower number of packets, with the exception of Scalable Broadcast Algorithm -protocols 8 and 9- which uses between 6 and 4 packets.

The problem with this approach to broadcasting analysis is that with a 5% of nodes holding the token, and being its distribution random, it will never take more than a few hops for any agent to find a token holder. Moreover, most of the chosen scenarios have a very high density of nodes, which is very usefull for congestion analysis but not very helpfull for studying broadcasting performance.

For a better understanding on how each protocol performs a different set of experiments has been run. In these experiments the simulation runs until all the nodes have had time to receive and rebroadcast the agent, allowing for the protocol to reach all possible nodes in the network.

Graph from figure 5.8 shows the number of packets each protocol used for flooding the whole network in an area of 1000x1000 meters. Unfortunately there are no data for neighbour knowledge methods with node densities of above one hundred nodes, but the pattern seems to be repeating with all shown node densities. In the scenario represented by this graph the whole network was reached with all protocol and configurations.

This graph shows how with lower node densities Simple Flooding -protocol 0- and Counter-Based Scheme with a threshold of six -protocol 5- use the same amount of packets. With a higher node density, like a thousand nodes in this area, all Counter-Based Scheme configurations perform better than Probabilistic Scheme with a 40% of rebroadcasting probability. In all the cases Ad Hoc Broadcast Protocol is the broadcasting protocol that requires less number of packets.

Figure 5.9 shows the performance of the same protocols in a 10000x10000 area, with one thousand nodes. However, protocol numbers three and four do not reach the whole network, so their data cannot be compared against the rest of the protocols. This is an example of the risks of protocols that don't take in account network topology in poorly populated networks, nodes within range can be skipped when they shouldn't.

Apart from that, similar conclusions to those in the previous graph can be extracted. In this scenario too, neighbour knowledge methods are clearly the ones that use the least number of packets while reaching all the available nodes.

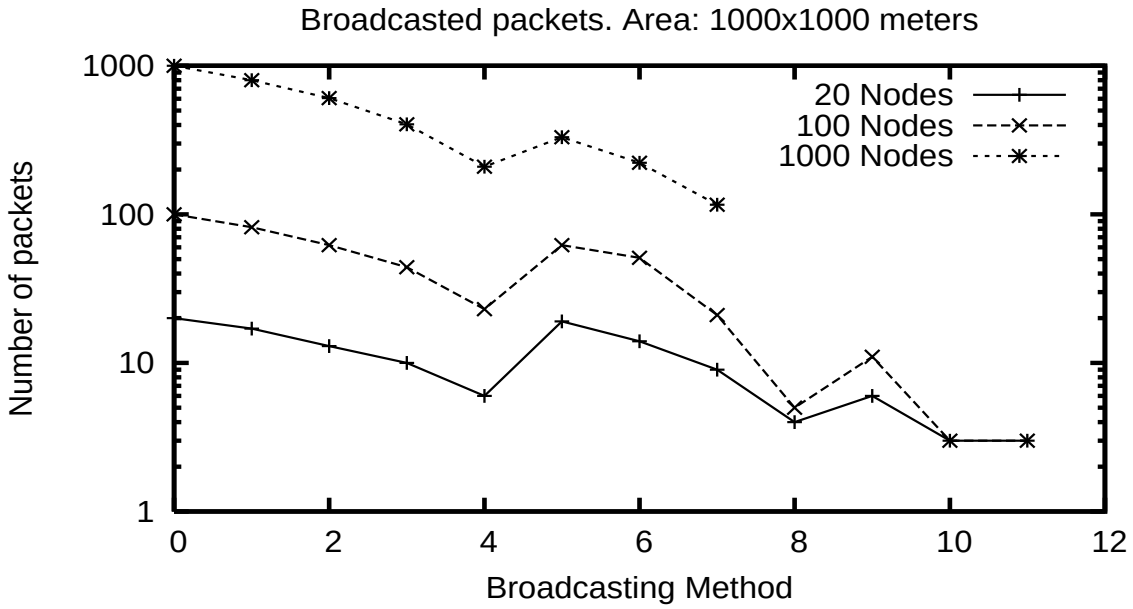


Figure 5.8: Broadcasting Analysis II. 1000x1000

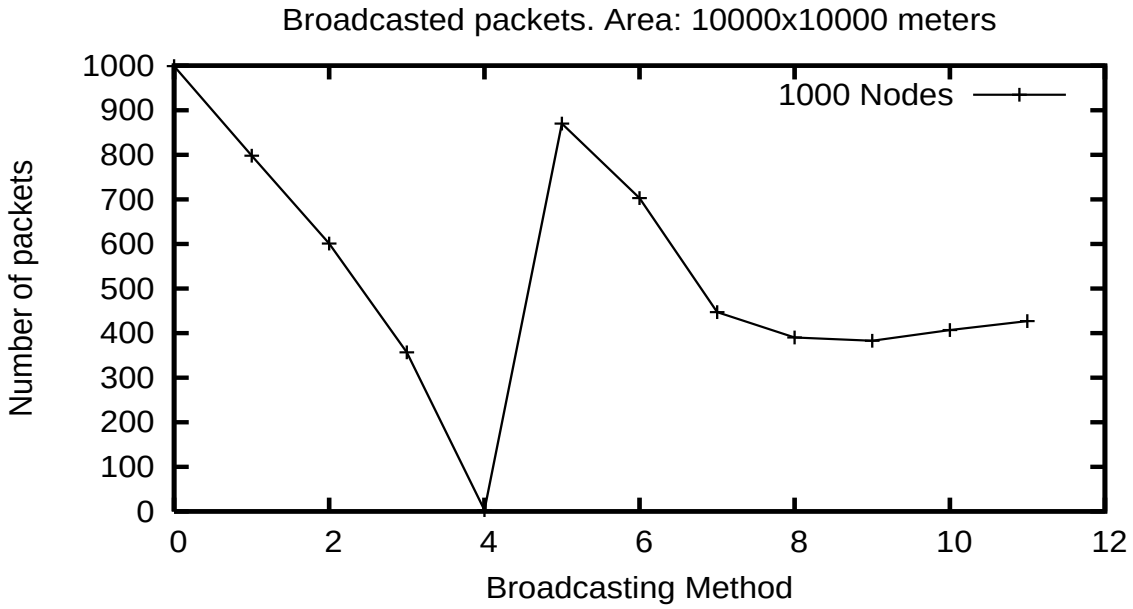


Figure 5.9: Broadcasting Analysis II. 10000x10000

# Chapter 6

# Conclusions

## 6.1 Protocol suitability

Based on the results from the experiments several conclusions can be extracted. The most relevant factor that influences the suitability of a protocol for any given network is the network density or even better, the average number of links each node has. For low density networks probabilistic methods are not recommended. Graph from figure 5.9 showed how these protocols can lead to nodes not being reached when they were within cover range.

In these networks neighbour knowledge protocols or Simple Flooding would be recommended. Depending on the characteristics of the devices Simple Flooding could be a better option that neighbour knowledge methods. If communications don't take place very often the cost of keeping an updated topology information that neighbour knowledge demands could be unnecessary. If the devices have better capabilities or there is a high network traffic then neighbour knowledge, and specifically ad hoc broadcast protocol is the most suitable protocol.

In densely populated networks Simple Flooding is completely discouraged. If nodes don't have enough capabilities to run neighbour knowledge protocols and the average density of nodes is known a well adjusted Scalable Broadcast Algorithm(SBA) would be recommended. This protocol doesn't require much more resources than Probabilistic Scheme and adapts much better to network topology, reducing the risk of leaving isolated nodes.

Ad hoc broadcast protocol has probed to perform very well under all network conditions, it creates low congestion, is as fast as more aggressive protocols like Simple Flooding and uses less broadcast packets than any other protocol, if "Hello" packets are not taken in account. If the network supports high traffic "Hello" packets would not represent a big overweight and would reduce congestion dramatically.

Comparing the two neighbour knowledge protocols studied, the performance of both is very similar under every situation but on searching speed. The use of a RAD in the Scalable Broadcast Algorithm (SBA) makes it much slower than Ad Hoc Broadcast Protocol. This fact indicates that choosing which nodes are going to rebroadcast in the sender node based on its neighbour knowledge performs as well as choosing it in the receiver after a RAD time and saves the node from spending this extra time.

## 6.2 Future Work

Apart from the obtained results the created simulating environment allows for several improvements and new researches.

There is a tool called Javis that is able to represent graphically the topology information written by the simulator into log files. Currently SWANS is not able to generate log files that can be used to reproduce graphically how the simulation executed, although the tool does. An extension for SWANS to write packet movement information into these log files would improve significantly the analysis of protocols.

Currently the agents that have been used in these simulations are quite simple and do not exhibit a very developed intelligence. However, the execution of more complex agents wouldn't be very difficult, as the framework is already there and can support any kind of agent.

Some of the simulations couldn't be run due to hardware requirements. Particularly when there is a high number of nodes in a small area and the broadcast method uses "Hello" packets the memory consumption is very high. Probably a refactoring of the application could improve these requirements.

Contrasting the results of this experiments with a real world experiment shouldn't be very difficult as 90% of the code is completely usable in a real world experiment.

Another interesting extension for this research would be to compare it against other searching methods like Unicast, using TCP communications, or peer to peer

approaches.

Finally, adding security to the agent system and measuring its impact would also be valuable. As the agent can come from anywhere and joining the network is very easy due to the used transmission channel, identifying the source of the agent as a valid source would be an important concern. Before executing any unknown code into a node the system must try to ensure its safety.

# Bibliography


[1] A. Qayyum, L. Viennot and A. Laouiti. Multipoint relaying: An efficient technique for flooding in mobile wireless networks. In *Technical Report 3898, INRIA-Rapport de recherche*. February 2000.

[2] B. Williams and T. Camp. Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc '02)*, pages 194–205, June 2002.

[3] C. E. Perkins and E. M. Royer. Ad hoc on demand distance vector (AODV) routing. In *Internet Draft*. November 1998.

[4] C. Siva Ram Murthy and B.S. Manoj. *Ad Hoc Wireless Networks: Architectures and Protocols*. Prentice Hall PTR, May 2004.

[5] D. B. Johnson, D. A. Maltz and Yih-Chun Hu. The dynamic source routing protocol for mobile ad hoc networks. In *Internet Draft*. July 2004.

[6] Defense Advanced Research Projects Agency (DARPA). http://www.darpa.mil/, August 2006.

[7] H. Lim and C. Kim. Multicast tree construction and flooding in wireless ad hoc networks. In *ACM International Workshop on Modeling,Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*, August 2000.

[8] IETF Mobile Ad-hoc Networks working group. http://www.ietf.org/html.charters/manet-charter.html, March 2006.

[9] J. P. Macker and M. S. Corson. Mobile ad hoc networking and the IETF. In *ACM Mobile Computing and Communications Review*, January 1998.

[10] J. Sucec and I. Marsic. An efficient distributed network-wide broadcast algorithm for mobile ad hoc networks. In *CAIP Technical Report 248 - Rutgers University*. May 2000.

[11] K. Obraczka, G. Tsudik and K. Viswanath. Pushing the Limits of Multicast in Ad Hoc Networks. In *21st Int. Conference on Distributed Computing Systems*, pages 719–722, April 2001.

[12] K. Obraczka, K. Viswanat and G. Tsudik. Flooding for Reliable Multicast in Multi-Hop Ad Hoc Networks. In *Wireless Networks, Vol. 7, No. 6*, pages 627–634, November 2001.

[13] LAN MAN Standards Committee of the IEEE Computer Society. Wireless LAN Medium Access Control MAC and Physical Layer. In *IEEE Standard 802.11*. February 1997.

[14] M. Takai, L. Bajaj, R. Ahuja and R. Bagrodia. GloMoSim: A Scalable Network Simulation Environment. *Technical report 990027, UCLA, Computer Science Department*, May 1999.

[15] OPNET Technologies. OPNET Modeler. http://www.opnet.com/products/modeler/, September 2006.

[16] R. Barr and Z. J. Haas. SWANS User Guide. http://jist.ece.cornell.edu/docs/040319-swans-user.pdf, March 2004.

[17] R. Barr and Z. J. Haas. JiST/SWANS. http://jist.ece.cornell.edu/, March 2005.

[18] R. Barr and Z. J. Haas. SWANS Java Documentation. http://jist.ece.cornell.edu/javadoc/index.html, September 2006.

[19] R. Ramanathan and J. Redi. A brief overview of Ad Hoc Networks: Challenges and direction. *IEEE Communications Magazine - 50th Anniversary Commemorative Issue*, April 2002.

[20] S. McCanne and S. Floyd. ns2-Network Simulator. http://www.isi.edu/nsnam/ns/, November 2005.

[21] Sze-Yao Ni, Yu-Chee Tseng, Yuh-Shyan Chen and Jang-Ping Sheu. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *Mobicom '99*, August 1999.

[22] T. Kunz. Multicasting in Mobile Ad-Hoc Networks: Achieving High Packet Delivery Ratios. In *2003 conference of the Centre for Advanced Studies on Collaborative research*, pages 156–170, October 2003.

[23] W. Peng and X. Lu. Efficient broadcast in mobile ad hoc networks using connected dominating sets. *Journal of Software - Beijing, China*, 1999.

[24] W. Peng and X. Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *MOBIHOC*, August 2000.

[25] Wikipedia - Mobile Ad Hoc Network. http://en.wikipedia.org/wiki/Mobile_ad-hoc_network, August 2006.