UNIVERSITÀ DEGLI STUDI DI CATANIA

Facoltà di Scienze Matematiche Fisiche e Naturali

Dottorato di ricerca in Informatica

ENFORCING TRUST, COLLABORATION AND POWER-SAVING IN MANETS

GIANPIERO COSTANTINO

A dissertation submitted to the Department of Mathematics and Computer Science and the committee on graduate studies of University of Catania, in fulfillment of the requirements for the degree of doctorate in computer science.

ADVISOR

Prof. Salvatore Riccobene

COORDINATOR

Prof. Cantone Domenico

XXIII Ciclo

Printed December 2010

Abstract

Nowadays mobile data communications are becoming more and more popular and, at same time, people routinely carry along the last generation of mobile devices, called *Smartphones*. In particular, the ability to connect directly to other similar devices makes smartphones very powerful.

The field that we are going to explore is that of Mobile Ad hoc Networks *(MANETs)*. In a network without infrastructure, devices are connected directly or by means of others and this makes collaboration fundamental even though the behaviour, that is, the collaborative attitude of each participant.

To provide a more transparent system, we present a mechanism in which a device, through observation of a neighbour, is able to establish the reputation of that neighbour. Afterwards, that value for each node is scattered across the network by means of special packets called *Recommendations*.

Values of reputation obtained with the previous mechanism are parameters of judgement for a Quality of Service *(QoS)* technique used in communications. Indeed, devices with a bad reputation are strongly penalised when they require a service to others. In particular, the penalisation that a node gets depends on its contribution of collaboration: the less is the collaboration the more is the penalisation.

Finally, we focus our research on power-saving. By introducing a novel protocol to cover a weakness of the Dynamic Source Routing *(DSR)*, each device becomes capable of preserving its own energy especially during long and repetitive communications.

All prototypes designed in this thesis were implemented and simulated through the Network Simulator 2 *(NS2)*.

Acknowledgements

I would like to thank my two advisors: the official one, Salvatore Riccobene, and the unofficial one, Giampaolo Bella. They taught me a lot with their experience throughout these three years of Ph.D. In particular, many thanks to my Professor because he gave me important suggestions regarding the hard life of a Ph.D. student, as a father makes with his son. On the other hand, Giampaolo has been for me a person often strict and meticulous, but fundamental for my career.

I would like to thank my external advisor at the University of Cambridge, Jon Crowcroft. I was a visitor for four months at the most important European University and Jon gave me the possibility to have an amazing experience there, both of research and human life.

I would like to thank my friends, my colleagues, Teresa and all the people close to me for spending with me important moments during my Ph.D.

Finally, but with the same importance, I would like to thank my family and in particular, my parents who sustained me throughout my eight years of University.

Contents

Chapter 1

Introduction

Back at the beginning of this century, when people spoke about mobile phones, what they had in mind were devices with standard features such as calling or sending text messages in mobility. At that time, purchase preferences were mainly driven by the exterior design of products. Then, companies introduced new features such as bluetooth communication [1] and a camera to take and share pictures directly with a mobile phone. Feature integration was the trend of the market of mobile devices until companies presented the *Smartphone*. According to the webpage of Wikipedia: [2] *a* ***smartphone*** *is a mobile phone that offers more advanced computing ability and connectivity than a basic "feature phone". While some feature phones are able to run simple applications based on generic platforms...*

This definition explains clearly that this new generation of devices is an evolution of the traditional ones. This is provided by the possibility to get connectivity with the Internet or other smartphones and to install applications. In particular, those may be very complicated.

The fully-fledged Operating System *(OS)* makes a smartphone very powerful. The *OS* of recent smartphones are particularly similar to the *OS* of a computer. Their functionalities are very similar even though they have different constraints. While the first generation of mobile-*OSs* were very limited, today, instead, they allow their users to perform mobile browsing, emailing and so on. In addition, it is possible to download new applications from an endless list of softwares to add new functionalities to the original *OS*. That is the reason why in the last years people are getting used to buying and using smartphones instead of traditional phones.

It is also true that a smartphone is typically less powerful than current personal computers, hence carrying out all tasks that are normally done in a desktop or laptop may not be possible to do. Although this is potentially feasible, smartphones are constrained by computational power and battery life. This latter is a severe limitation of smartphones. Indeed, applications require a huge use of the processor with consequent significant battery consumption. The more intensively a processor runs, the faster it consumes energy. Therefore, companies such as Apple [3] for a few years limited the number of applications that are running in multitasking in order to save energy. Moreover, depletion of energy does not depend only on processor activity but also on communication workload. For this reason a user must prevent an excessive consumption of the battery due to his usage.

Nowadays, people are tempted to buy a smartphone rather than a traditional mobile phone at least because the latter offers more functionalities. People can easily create a network with their smartphones in order to exchange data. The size of the network can be formed depends upon the communication technology used. For example, a bluetooth device cannot reach the same distances as a Wi-Fi one. This is due to hardware limitations. Either technology allows creation of an Ad-Hoc network. In particular, an Ad-Hoc network is characterised by the lack of a central infrastructure, so that the users are connected directly. In case of devices, called also nodes, that are not reachable because they are too far away, other mobile phones are used to forward messages to a destination. This network is called Mobile Ad hoc Network *(MANET)*, and its participants can only be mobile devices. Mobility is the peculiarity of MANETs with respect to other types of networks.

In order to make sure that a MANET runs properly, those users that are in that network, must take a fair behaviour. Fairness here means that each device should collaborate to the networks survival. The reason why this is very important is the limitations of the devices, which were outlined above. As already noted a user cannot reach all other devices directly or in one hop but others have to help forward the message. Since nothing enforces that collaboration, a device could decide to avoid collaboration and as a consequence the network may not work in the right way.

With the motivation of thwarting the fragility of MANETs just outlined, we decided to study these networks both macroscopically, that is holistically, and microscopically, that is terms of behaviour of each participating devices. Moreover, we want to highlight that the research fields considered in this thesis are split into three different branches:

- Reputation;

- Collaboration;
- Power-saving.

All those research fields represent our fully-fledged work seen as different "bricks" and by merging all points, we provide a sound product ables to prevent and reduce selfishness, combined to energy saving for MANETs.

We are particularly interested in observing the reputation of the participating nodes. This parameter indicates how much a node is willing to collaborate in keeping the network. Although the willingness to collaborate of each node cannot be precisely known to its neighbours, this may obtain a good estimate by calculating the reputation of the node. In this vein, we will introduce a metrics for the reputation of all participants of an Ad-Hoc network.

However, the goals of this thesis are yet more complex and ambitious. It was already remarked that collaboration is crucial to MANETs. Therefore we aim at increasing the collaboration of the participating nodes as much as possible. In a rational prospective, a node collaborates only if it earns something. In particular, using an approach similar to that used in P2P networks — fully described later — we show a protocol that, through a Quality of Service *(QoS)* technique pushes nodes to collaborate.

Intuitively, a node that behaves selfishly will be somehow penalised. The obvious way to penalise a node in a network is to delay the traffic that the node generates. By contracts, a node can prevent such a penalisation by being collaborative, in such a way that its generated traffic will not undergo any delay.

So, by introducing *QoS* during communications, we want to show that collaboration can be increased. The intuition is that a node tends to collaborate in order to avoid a low quality service. In Chapter (6), we show our idea of *QoS* adapted for MANETs. Also, we provide some simulations in which *QoS* is used by all participants of the network.

Finally, we focus our work on power-saving. As mentioned above, the battery constrained of mobile nodes is crucial. It is impossible to build an infinite battery, so our aim is a mechanism to reduce the wasting of energy. In particular, our approach takes advantage of a weakness of the routing protocol. As we describe later, although nodes keep the ability to reduce their collaboration to preserve energy, they will be guaranties that the network remains alive. This strategy can be implemented together with the intelligent routing strategy mentioned earlier. We have tested them through simulations and we will describe our findings. For example, we shall see that a node with role of forwarder can preserve a lot of energy.

This thesis is organised in the following way: Chapter (2) introduces in a deep way MANETs and, also we give a survey of the most important routing protocols. Then (Ch.3) we provide to the reader an overview of the key features of the Dynamic Source Routing protocol. After that, we present the Network Simulator 2 (Ch.4), its way of working and its wireless section. In Chapter (5) the topic regarding reputation is introduced and argued. Afterward, we present our mechanism to push collaboration (Ch.6) through a Quality of Service technique. Finally, we tackle the power-saving issue (Ch.7) and we conclude this thesis with Chapter(8).

Chapter 2

MANETs

A Mobile Ad hoc network is characterised by the presence of wireless devices that are connected each other without a prefixed-infrastructure. The main feature of this network is the lack of particular structural devices and this makes MANETs and P2Ps belonged to under the same branch. On the other hand devices, such as routers, switches and so on, make wireless ad hoc network very different from the wired one. In the latter case, a participant can feel free to send a packet to the network because when a packet reaches a router, this is able to route it in the right way.

Since in a MANET devices have to perform different tasks, such as routing, collaboration between participants is fundamental. However, a device not alway gives its availability to collaborate and this could be dangerous for a network in terms of reachable nodes. One of the main causes of this selfishness is to save energy's node since it is a limited resource.

In the past a MANET was mainly used for military aims. The possibility to realise a wireless network without external supports, like radio-bridge, made this technology very suitable for the previous field. Indeed, each soldier, which carries his own mobile device, could be able to reach another teammate in a easy way.

Nowadays, the presence of mobile devices is becoming more and more popular. This allows people, which are using their devices, like a smartphone or PDA *(Personal Digital Assistant)*, to create a network instantly. All these handsets are equipped with one or more wireless resources. So, people who are in a public place are able to realise a network without any additional charge.

2.1 Main features of a MANET

As we said before, MANETs and P2Ps are very similar and a fair behaviour represents a fundamental aspect of both network. In particular, sharing of resources by nodes is needed to provide an efficient net.

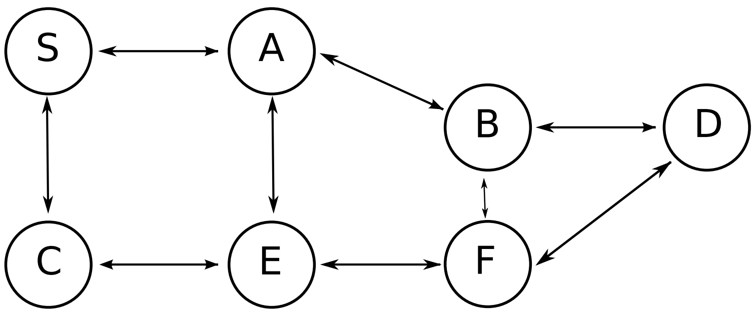

Figure 2.1: Example of a wireless network

In fig.2.1 is shown a MANET composed by some nodes. They are able to communicate each other and this can be done directly or through other devices. Due to its transmission range, the sender *"S"* cannot reach in one-hop the receipt *"D"*, so it has to ask its neighbours to forward the message. The request of collaboration can be performed only if others share their resources, but on the other point of view, a forwarder pays something in terms of resources during the sharing. According to this, not always a node wants to forward data for others and this way to behave as selfish could be very dangerous for a MANET.

Another feature mentioned before is the lack of a prefixed infrastructure. The latter can be classified as the biggest difference between traditional cabled networks and ad hoc wireless ones. In the first one, a node performs only the role either of sender or of receiver. In particular, when it wants send a message, it uses just a gateway node, such as a router. Instead, since in a MANET devices run the same role, they have to forward data for others.

Mobility is an important characteristic of a wireless network and it makes the topology of a network very dynamic. In particular, this feature requires that the routing protocol used in MANETs must be as reliable as possible. According to this protocols that build paths on-demand are proposed.

2.2 Main features of a device

A node of a MANET is characterised by its light weight. That is why smartphones nowadays are perfect candidates to take part in this network, even though they have a limited lifetime due to their battery. Sending and receiving information are activities in which a node can waste a lot of energy. Especially, when a node is running out of energy it could decide to stop collaborating in order to save the remain lifetime.

Short-coverage range is the reason why a node needs collaboration of others. Today a traditional smartphone can cover a transmission range for one-two hundred meters and only close devices can be reached. Also, a communication range depends on the position of a node. For example, an indoor communication is more limited because of walls that can reduce drastically the transmission. Instead, when a network is built in a free space, this can allow participants to cover a larger area.

From the security point of view, a device should adopt a cryptographic layer during communications since wireless traffic can be easily "observed". In spite of this, a cryptographic layer to add to a MANET cannot be high because low computation processor embedded in each device could not be able to perform heavy operations. That is why in the majority of situations a message is sent unencrypted.

2.3 Taxonomy of Routing Protocols

As we said before a MANET is composed only by nodes that accomplish a same role. Routing of messages is not done by routers but devices have to discover paths. In literature are presents different routing protocols that are classified in: *Flat Routing*, *Hierarchical Routing* and *Geographic Position Assisted Routing*.

Following in fig.2.2 we show the complete classification of all routing protocols:

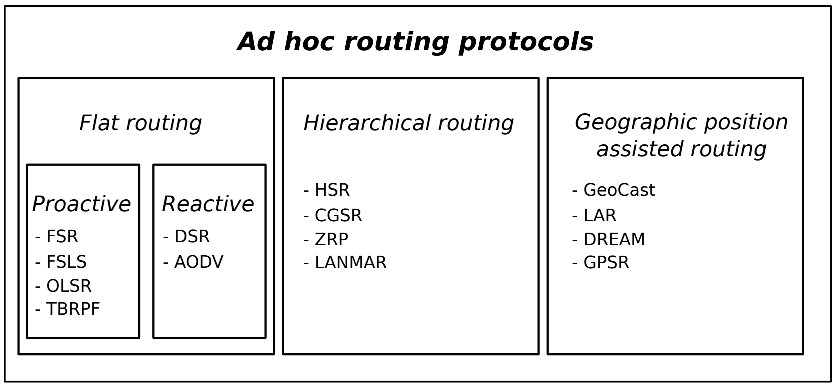

Figure 2.2: Classification of Ad hoc routing protocols

2.3.1 Flat Routing

In this set of protocols all nodes are part of the same hierarchical level. Indeed, a message can be broadcasted towards others without following a particular pattern.

Proactive

As part of flat routing protocols the proactive ones are characterised by the presence of routing tables inside each node. Each node of a network contains a table with the majority of paths already known. When a device wants to send a message in a quick fashion, it searches the path for a receipt inside its own local table. As disadvantage, each node must update regularly its own table in oder to obtain reliable paths. This happens through tables exchange between nodes and, when a participant receives a new update it must replace its own older data with the newer ones.

Reactive

On the contrary these protocols adopt a different approach. A node does not know *a priori* the route to follow to reach another device. When this happens it must discover the right path in real time. Using a special routine, called *Route Discovery*, a node is able to discover the most reliable path. According to this a device knows only paths that it needs.

Benefits of these protocols is the possibility to get the most updated path because mobility is very frequent. That is the reason why recently this branch of protocols are privileged respect to the proactive ones.

2.3.2 Hierarchical Routing

Although a MANET is composed by a limited number of devices, scalability of the participants must be guarantee. To prevent problems of overloaded links a routing protocol must be able to overcome this threat. Due to this factor, hierarchical routing is more suitable. The main idea of these protocols is to split all set of nodes into clusters and, each group keeps a leader, called *clusterhead*. When a node wants to communicate with another device out of the cluster, it must pass through the clusterhead.

Using this approach a node can contain a reduced number of items inside its own routing table. Indeed, it has to know only links towards its clusterhead. Moreover, this idea decrease the amount of maintenance data that travels across the net.

2.3.3 Geographic Position Assisted Routing

Nowadays, majority of smartphone are equipped with a Global Positioning System *(GPS)*, that is very cheap and small. Also, today precision to locate a device is becoming more and more accurate.

Since, a *GPS* is used to find the location of a node, geographic position assisted protocols manage this information to discover paths in a quick manner. By knowing the position of receipt, a sender is able to select *a priori* which nodes must forward the packet. Moreover, due to high mobility of devices, positions of them must be often updated. However, this strategy could overload a network since participants must exchange their new positions.

On the other hand a node can keep its table very light. This because they don't need to save routes in their tables. According to this *GPAR* protocols are suitable for sensor network. Indeed, within this environment devices are characterised by the presence of weak resources.

2.4 Survey of MANET' Routing Protocols

During these years researchers developed different routing protocols for MANETs. In this section we are going to present the most important routing protocols that can be used throughout communications. In particular, we decided to argue only one for each family in a deep way and for a detailed description of others, we refer the reader to the original papers.

2.4.1 Flat Routing

Following the list of protocols that are part of proactive family.

Proactive Routing Protocols

- Fisheye State Routing *(FSR)* [4]
- Optimized Link State Routing Protocol *(OSLR)* [5]
- Topology Broadcast Based on Reverse Path Forwarding *(TBRPF)* [6]

Optimized Link State Routing Protocol

Since a path is not obtained on-demand each node must discover all interested paths before starting a communication. This is done in the following way: first of all through a *HELLO* message a node explores its neighbourhood and it is sent in a broadcast way towards all neighbours. In particular, an *HELLO* message contains a list of paths already known. When a node receives the message, it just stores the information contained without forwarding the packet. All data obtained from an *HELLO* message are kept only for a limited period beacuse mobility of nodes pushes them to refresh their neighbourhood frequently.

Messages to find neighbours are sent in a flooding manner. This technique could be counterproductive but in order to limit it, *OLSR* uses a mechanisms, which is explained in [7], that reduce the problem. Moreover, when nodes receive data regarding the topology of a network, they are able to keep this information in order to store available routes. After that, a node uses the Shortest Path First *SPF* strategy to select a path.

Reactive Routing Protocols

Following the most important reactive protocols are listed:

- Dynamic Source Routing *(DSR)* [8]
- Ad hoc On-Demand Distance Vector Routing *(AODV)* [9]

Dynamic Source Routing

This protocol is deeply used for the aim of this thesis and for this reason why we preferred to dedicate Chapter (3) for the whole description.

Ad hoc On-Demand Distance Vector Routing

AODV was realised in order to obtain a routing as light as possible. According to this a node is used to keeping a table with just the most used links. In particular, a device disowns a route until it does not run the procedure to discover a path for a destination node. The latter procedure is done by means a Route Request *(RREQ)*. A *RREQ* packet contains the following fields:

- *Source*: address of a sender;
- *Source-Sequence*: list of nodes that forward the RREQ;
- *Destination*: address of a destination;
- *Hop-Counter*: counter of hops;

When a node receives a *RREQ*, it forwards the request to its neighbours adding the reference of the previous forwarder into the packet. This process is called *Revers Path Setup* and it is represented in fig.2.3a. However, this information is kept just for a shot period but enough to create a reply message to the source.

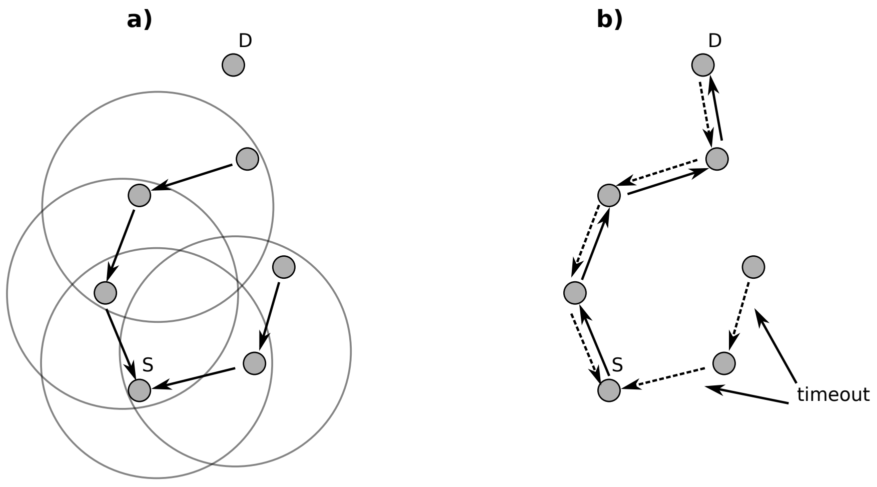

Figure 2.3: Reverse *(fig.a)* and forward path *(fig.b)* in AODV

If a forwarder, during a *RREQ* forwarding, knows the path to reach a destination, it uses this information to build the whole route. After that, it creates a Route Replay *(RREP)* and sends back this packet with the path discovered. In particular, the *RREP* travels through the route found by Revers Path Setup process, fig.2.3b.

When the sender receives a *RREP*, it extrapolates the path from the packet. According to this the route obtained is used by the sender to reach the receipt. However, the new path got by the source node is valid only for a period. Indeed, if its *lifetime* is not updated the path will be removed from the cache table of the node.

As we said before, *AODV* is a reactive protocol, this allows nodes to keep light routing table storing only minimal information regarding to paths.

Another characteristics of the routing protocol is the lack of periodical messages. Adopting this feature the protocol avoid to overload a network with maintenance data.

2.4.2 Hierarchical Routing

The following protocols adopt the hierarchical approach to route information in a MANET:

- Clusterhead Gateway Switch Routing *(CGSR)*
- Hierarchical State Routing *(HSR)* [10]
- Landmark Ad-hoc Routing Protocol *(LANMAR)* [11]

Clusterhead Gateway Switch Routing

This protocol uses the clustering algorithm called Least Clusterhead Change *(LCC)*. The latter one is used to split a network into different clusters. Inside each cluster a *clusterhead* is elected. If a node can be reached by a clusterhead, it belongs to clusterhead's cluster. In particular, whether a node is covered by two clusterheads it assumes the role of *gateway*.

When a clusterhead changes its cluster, it could lose its role. Indeed, it is impossible that two clusterheads "live" inside the same cluster. According to this one of them must lost its role. So, the clusterhead with the bigger numbers of active links belonged to it can become the only clusterhead

$CGWR$ adopts the distant vector routing DV algorithm. For routing a packet each node has two tables. The first one contains the clusterhead of each destination node. Instead, the other one reveals to a node the right path to reach a cluster. When a node wants to send a packet to a destination one, it looks for the clusterhead of a destination node. After that, it sends the packet to its clusterhead. Then, the latter one forwards the packet to the gateway that know how to reach the destination's clusterhead. However, it is possible that the packet must pass through other clusters before reaching the destination. In fig.2.4 is shown an example of packet forwarding until a destination node.

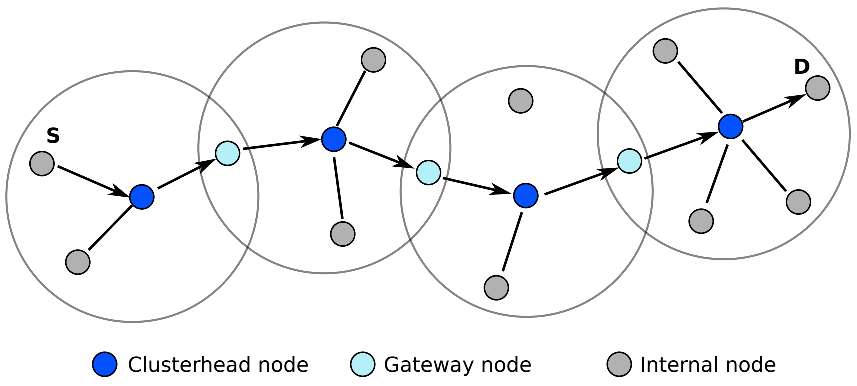

Figure 2.4: Forwarding of a packet using CGSR

2.4.3 Geographic Position Assisted Routing

Below routing protocols that use a *GPS* device to discover node's position are listed:

- Distance Routing Effect Algorithm for Mobility *(DREAM)* [12]
- Geographic Addressing and Routing *(GeoCast)* [13]
- Location-Aided Routing *(LAR)* [14]

- Greedy Perimeter Stateless Routing *(GPRS)* [15]

Distance Routing Effect Algorithm for Mobility

DREAM is a proactive protocol and it uses a node's position to route information. Each node contains a table in order to store the position of others. So that information about position of nodes are accurate, a participant must send its update position periodically. In particular, the bigger is mobility, the more frequent is sent an update message of a node's position.

This protocol adopts special features to refresh the localisation of a device. Indeed, a participant prefers to send an accurate position to close nodes because a device is less interested in a accurate information of far away nodes. This idea was implemented to reduce the cost due to the information exchange across the network.

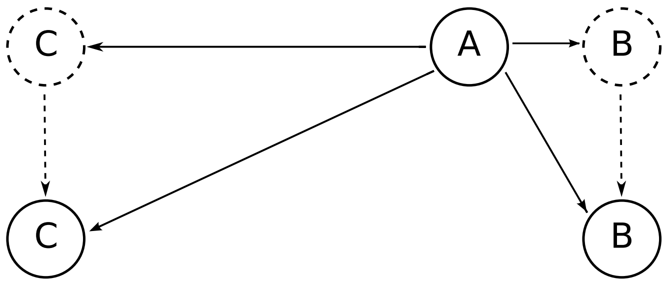

Figure 2.5: Inserire caption con il nome dell'effetto

To explain better the concept, in fig.2.5 are shown three nodes. *Node B* and *Node C* are moving in the same way. In particular, *Node C* is farer to *Node A* respect to *Node B*. It is possible to observe that the movement of *Node C* looks like "less important" of the other one. Indeed, from the point of view of *Node A* the degree of movement of *Node C* seems lower of *Node B* one. Due to this phenomenon *Node A* can recognise the position change of *Node C* even though the latter one provides an information less accurate.

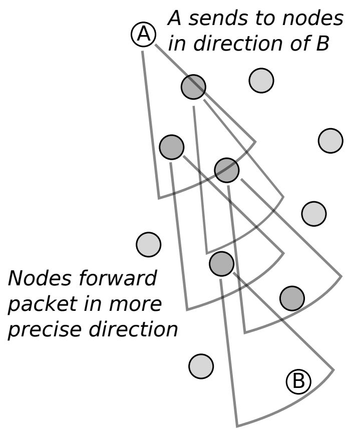
A
A sends to nodes in direction of B
Nodes forward packet in more precise direction
B

Figure 2.6: Partial flooding in DREAM

When a node collected all localisation positions of others, it is able to start to send a message using a partial flooding, fig.2.6.

A sender selects a number of neighbours that are in the same direction of the destination node. Nodes selected by a sender are added in the header of the message. In this way, when one-hop neighbour receives the message, it will forward the latter one only if it is present in the header, otherwise the device will discard it. Each forwarder will repeat the same procedure done by the sender. Following this idea the message is delivered to the destination. In case that a forwarder does not find any neighbours in the same direction of the receipt, it will forward the message using a flooding.

2.5 MANETs like P2Ps?

P2P networks and MANETs can be considered very close each other because they need collaboration for working. In particular, a P2P is a distributed network composed only by participants that share resources. Respect to the client-server paradigm, where only a server provides information, in P2P networks all devices can provide something in terms of resources; these usually are, memories, CPUs or just documents. Moreover, the P2P topology respect to the client-server one avoids that nodes overload the provider, i.e. a single server. This feature makes P2P networks more scalable respect to the older paradigm.

Nowadays, P2P networks are used especially for sharing of documents. The first P2P software developed was Napster [16], it was developed to share music files, like mp3 [17] but its life was not easy since sharing of mp3 was illegal. Indeed, people was able to get music without paying and this phenomenon is, clearly, not allowed.

Working of Napster was very easy: a user could share its files to the community or it could search other music files using different keywords. Doing so a list of downloaded files was available and a peer was able to download the desired file using the P2P protocol.

It is clear that this mechanism cannot be legal and as consequence the life of Napster was short. In fact, the application was closed but at the same time other P2P applications were developed. After some years, Napster was re-open but as a online music store, and today people are able to buy music legally using the same software.

Nevertheless, after Napster a lot of other P2P protocols were born. The most famous are Kazaa [18], Gnutella [19], FastTrack [20], FreeNet [21], Skype [22, 23], Bittorrent [24], Emule [25] and so on. All these listed protocols are different each other although they belong all within the P2P family. Respect to Napster, the others are more complete softwares. These allow peers to share a huge quantity of documents, such as movies, musics, books etc, but as it happened for Napster all of them are often blamed to allow illegal download of files.

On the other hand, these applications can be seen as a very easy way to share resources. Indeed, shared files can be legal and companies often use P2P protocols to provide a faster download of free software. This is a perfect alternative to the client-server paradigm that could make a company's server overloaded easily denying to get the files to others.

Main strength of P2P networks is the collaboration and peers sharing their files allows the network to work properly. In order to reduce the situation in which users would avoid collaboration, P2P protocols adopts mechanisms to push peers to collaborate. For example, Emule introduced a system that works using credits. Indeed each user keeps an amount of credits that increase when a peer uploads any type of file. Doing so, the user is able to get a faster download spending its own credit acquired previously.

When peers discovered that just sharing files their obtained credit, dishonest users decided to share documents with false names. Using this approach a peer who downloads, got a fake document, whereas a peer who uploads, got credit wrongly. This is a strong weaknesses of that protocol and to tackle that problem, Bittorrent decided to use a different approach. The main difference is that when a peer earns credits from another one, it can spend the credits acquired only with that peer. In this way, a user has less interest to share fake files since it cannot re-spend the credit acquired with others. The idea used by Bittorrent got very success and nowadays the latter protocol is becoming more and more used.

In the last years, P2P protocols are not used just for sharing of documents. Skype, which is the application that allows people to call each other using Voice Over IP *(VOIP)* [26], uses a P2P strategy to link peers. Today, this application is often used and peers are able to call both landline and mobile numbers in a very cheap price. Since Skype is a P2P network, it does not use a server to keep peers' information. Moreover, all data exchanged between peers are encrypted and the algorithm used to encrypt is the Advanced Encryption Standard *(AES)* [27].

Although MANETs and P2Ps share some aspects, like collaboration and lack of a fixed infrastructure, they have been developed for different aims. MANETs were thought for small-medium networks, where devices create links of communication and these last for a short time. Then, participants are mobile users constrained by their own battery.

On the other hand, P2Ps manage a large number of peers who are connected each other to exchange an huge amount of data. Here, users can use different machines to connect to a network, such as desktops, laptops and so on. Moreover, they do not have the battery limitation and this allows a P2P network to persist for long time.

Chapter 3

Dynamic Source Routing

Given the frequency of node mobility, it is important to keep routing tables updated and Flat Routing protocols of MANETs fall into two families. The first are *proactive* protocols, which use a table-driven approach to establish paths and the reliability of routing paths depends on how frequently tables are refreshed. If this is done too often, control traffic can overload connection links. To overcome this problem *reactive* protocols are often used to manage the routing. These protocols build paths on-demand and *Dynamic Source Routing* (DSR) is one example. It employs a lighter weight route discovery scheme, consisting of two sub-procedures: *Route Request* (RREQ) and *Route Replay* (RREP), which are used to build a path. To manage the case when links on current routes are broken, DSR implements a separate *Route Maintenance* procedure.

3.1 Route Discovery

Here we review the main features of DSR. For a complete description, we refer the reader to the original paper [8].

3.1.1 Route Request

At the start of any activity, a node does not know paths to other devices. To find out new routes it uses a special packet called *Route Request* (RREQ). The main structure of the packet is as follows:

- A destination node.
- An *identification number* (ID) of the RREQ.

When the packet is received by a neighbour, it can proceed in two different ways:

1. If it knows how to reach the destination, it will take the path from its route cache table and sends back, using a *Route Replay* (RREP) the whole path.
2. Otherwise, it forwards to its neighbours the packet generated by the sender appending its own address to the route record.

This two last points are repeated by each node. If this packet arrives to the receipt, the latter is able to read in the RREQ which nodes have been forwarding the request. At this point the path is established and is sent back to the sender using a RREP.

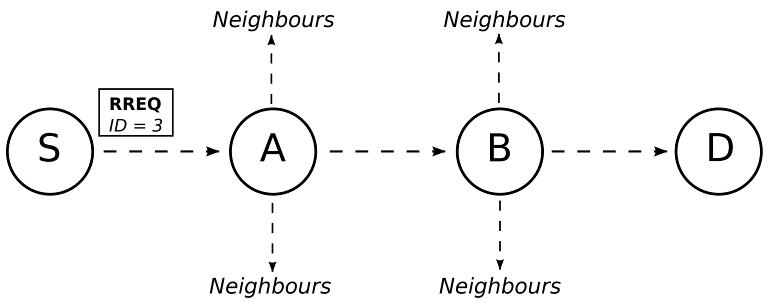

Figure 3.1: Path Discovering with RREQ

The fig.3.1 shows how the RREQ travels through the intermediate nodes until it reaches the destination. Moreover, the "ID" number avoid that a node can forward a RREQ for multiple times, generating an infinite loop.

3.1.2 Route Replay

The RREP contains the path built from *"S"* to *"D"* and it is sent back using those nodes that forwarded the previous RREQ.

In case of unidirectional link, the destination must find an alternative path to send its RREP back. First of all, it checks other ways from its route cache table. If no alternatives are available, it will create a new route discovery for *"S"*. But just in this situation it is able to piggyback the discovered path in the new RREQ.

3.2 Route Maintenance

Mobility is one of the strongest aspects of the MANETs. As consequence, links between nodes change continuously and according to this paths stored in a node's cache table must be reliable. To guarantee this point DSR uses the *Route Maintenance* procedure.

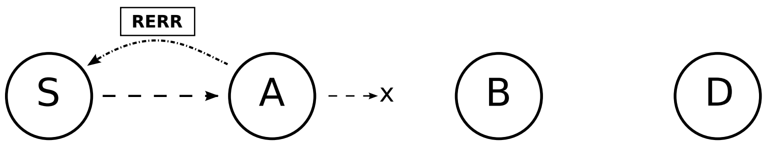

Figure 3.2: Route Maintenance with RERR

Every node is responsible to check if the link between itself and the next one is still active, e.g. in fig.3.2 Node *"A"* checks for the link {A - B} and Node *"B"* for {B - D}. If *"A"* recognizes that there is a link problem in its link, it will wait for a maximum number of times the acknowledgement message from *"B"*. If the message does not arrive, *"A"* removes from its cache route table the link {A - B}. Moreover, it will spread this information using a *Route Error* (RERR) to the other nodes that use the same link too.

Chapter 4

Network Simulator 2

NS2 is a discrete event simulator targeted at networking research. It is often used in university environments especially by researchers, students and particularity of this simulator is its development. Periodically, NS2 undergoes updates from different sources, it can be downloaded with a GNU [28] license and according to this a skilled user could realise modifications to the simulator in order to improve the product itself. Nevertheless, every change to the simulator must be approved and validated by the *NS2* Committee.

Using *NS2* is possible to run the majority of networking features for wired and wireless environment. *NS2* has implemented all stack of network layers, from the physical to the application one and also, and to set up a simulation a user can choose different routing protocols, numbers of nodes, the type of device and other important features.

NS2 was born in the 1989 as part of the project *"REAL Network Simulator"* [29] and in the last years it is undergoing important modifications. Today, it is developed by different organisations: Defense Advanced Research

Projects Agency *(DARPA)* [30] with the project Simulation Augmented by Measurement and Analysis of Networks *(SAMAN)*, NFS with Collaborative Simulation for Education and Research *(CONSER)*. Also a contribution has been given by the UCB Daedelus, CMU Monarch projects and Sun Microsystems for the wireless module of the simulator.

In order to perform a graphical simulation, a research can use Network Animator *(NAM)*. It is an application that reads the output of *NS2* and parsing its data traces, *NAM* generates graphical simulations. This allows a user to observe all devices involved in the network, their movement and when nodes send packets to others. Moreover, other features can be enabled or disabled to get an accurate observation.

4.1 How NS2 works

By far the most important feature to know about *NS2* is that the simulator can be programmed using two different languages: *tcl/TK* and *C++*. The reason way developers decided to follow this mechanism is to provide more flexibility during the developing phase. A script done only with *tcl/TK* uses the standards modules, written in *C++*, already ready for *NS2*. Instead, when it is needed to add or realise new functionalities to the simulator a developer has to work inside the kernel, and it can be done through the knowledge of *C++*.

When a simulation written in *OTcl* is run, the script is interpreted during its execution and modifications can be done easily without recompiling all code. On the other hand, a reader should know that *C++* must be compiled each time that a programmer makes a change and this process requires more time.

Although we said that *NS2* is composed by two main languages, these two are strongly linked each other. In order to give an easy understanding how the linkage works, we introduce this example of *C++* class:

```
class MobileNode : public Node {
[...]
}
```

In this case the *MobileNode* class inherits all the functions of the *Node* class and, also it implements new methods. Going through the hierarchy, the class *ParentNode* appears:

```
class ParentNode : public TclObject {
[...]
}
```

This class, i.e. *TclObject*, is used to link the *C++* and *Tcl* language and it is defined in the following way:

```
class TclObject : public TclObject {
[...]
virtual int command(int argc, const char*const*argv);
void bind(const char* var, TracedInt* val);
[...]
}
```

We consider only two functions in this part of code and they are inherited by all the classes of the hierarchy. Each time that a developer uses a *OTcl* method the function *command* is invoked.

The command to move an agent using the *OTcl* code is:

```
set x 10;
set y 2;
set speed 5
$node(1) setdest $x $y $speed
}
```

The method *setdest* is not implemented in *OTcl* but in *C++* and according to this the system invokes the following function with *argc=5* and *argv={_o32, setdest, 10, 2, 3}*.

```
MobileNode::command(int argc, const char*const*argv);
[...]
}
```

Inside the class *MobileNode* is present the method *setdest* that contains the implementation to move a node from a place to another within the network space. It is implemented using *C++* and can be modified changing its source code.

4.2 The structure of NS2

NS2 is composed by three main elements:

- Event Scheduler;
- Network Component Object;
- Trace.

The ***Event Scheduler*** is the component that manages all activity of the network's objects. Every time that an object performs a new action, this is put in a data structure, such as a list or heap. An action can be a new packet generated by an application or the movement of a device. When the scheduler extracts the event from the list, the action for that event is run by the simulator. Moreover, it is possible that an event can create a new event as consequence of that. For example when an application generates a packet this travels through different layers until it goes down to the network layer.

NS2 is not able to run parallel events and if more events are scheduled in the same time, they are run in a sequential way. This is a strong weakness of the simulator since a simulation can be speeded up using multithreading technique. During our work we recognised that if NS2 had been multithreading we would have saved a lot of time in phase of testing. Especially, if a user implements a large network with a huge amount of devices and connections simulations could be long.

In order to give this feature to *NS2*, throughout my Ph.D. with other students and my supervisor, we tried to move *NS2* into a multithreading architecture. After spending months on this work we realised that *NS2* cannot be adapted to a similar architeture since all system has been created for a serial scheduling.

The ***Network Component Object*** is an entities that can be used as part of a network. This is either hardware or a software components, the first ones are like routers or switches, instead the latter ones can be application protocols, such as *HTTP, FTP*.

All components are part of a main class, then using the object paradigm they created an hierarchy of classes. Doing so, each object has different parameters that can be changed, for example the queue of a router, the speed of a mobile device or the bit rate of a communication channel.

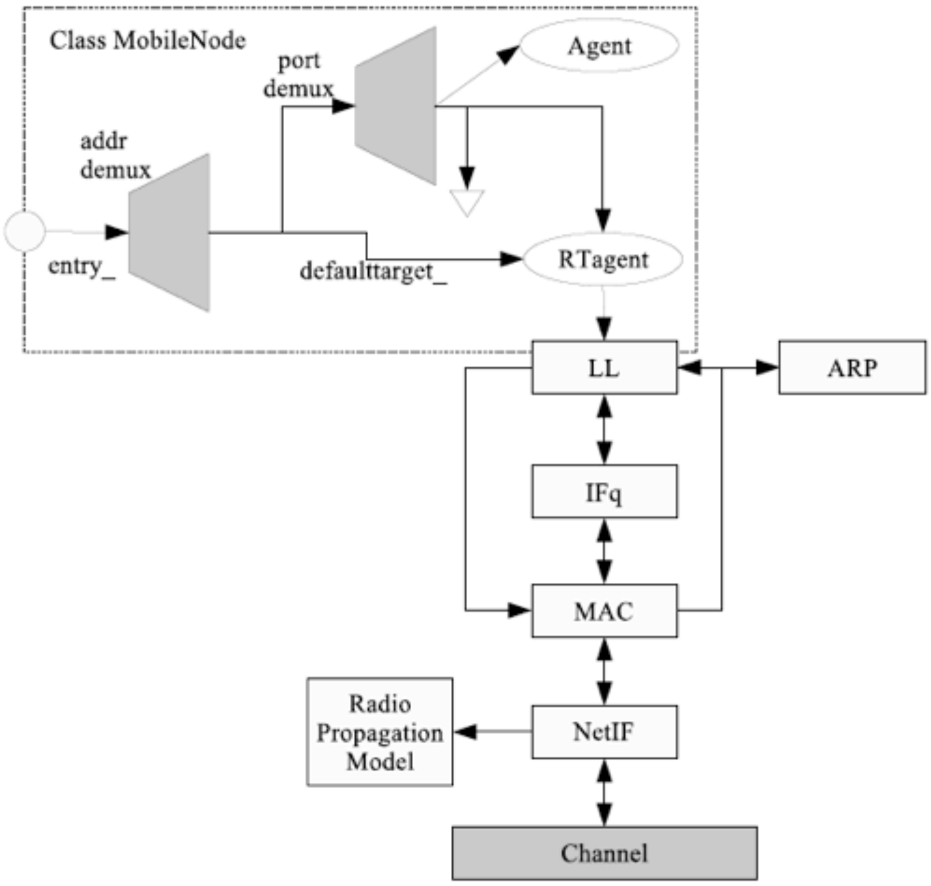

Figure 4.1: Structure of a MobileNode

The objects are split in the following way:

- Nodes (Node, MobileNode, SRNode, etc...)
- Links (SimpleLink, DelayLink, FQLink, CBQLink, etc...)
- Agents (UDP, TCP, SCTP, SRM, etc...)
- Applications (FTP, HTTP, CBR, etc...)

The most important network object is the class *Node*. It represents a device that could be a PDA, smartphone or a mobile generic object. The *Node* object has implemented all layers that the stack protocol needs: from the network layer to the application one. Every time that a device receives packets from its own network layer, it moves the packets towards its above layers. If the packet is not intended for it, the receiver will put the packet again in the network. This procedure is repeated until the packet does not arrive to the destination.

The last component is ***Trace***; it is the output interface of the network and allows a user to know in a detailed way how the simulator has worked. Every time that an object creates a new event this is written in the trace file. Analysing the file a user can observe all the activity of the simulator, for example, if a *FTP* packet has been sent or whether a node changed its position. In particular *NS2* provides three different trace files, that are used for different goals. These are:

1. *NS2 trace*: the simulator stores all the information concerning the network simulated;

2. *NAM trace*: it is used by NAM. This is an application that shows in a graphic way the behaviour of the network's objects. For example, it is possible to observe the movement of a device or a packet that travels from a node to another one.

3. *CMU trace*: this file is used to trace only information regarding to wireless networks.

4.3 Wireless networking in NS2

The wireless module has been introduced in 1992 as part of the *CMU Monarch Project*. Developers of this project decided to use the standard *NS2* object adding the wireless module and this has been done implementing the wireless interface, the radio propagation module and routing protocols. Subsequently, they added features like mobility and energy consumption.

During our research work, we used only the wireless part of *NS2* and according to this we chose *DSR* as routing protocol. In order to use that, *NS2* provides a set of objects that implement the protocol.

The following code, in a tcl script, enables *NS2* to use *DSR*:

```
set val(rp)    DSR
...
$ns node-config -adhocRouting $val(rp) \
...
```

The main features of *DSR* are implemented in the class *(dsragent.c)*, which can be found under the subdirectory ./ns2/dsr/ . The method that we explored and used in a deep way is *recv* and it manages each packet received by a node. In particular, its signature is:

```
void DSRAgent::recv(Packet* packet, Handler*)
```

When a node receives a packet from the physical layer, this goes up following the protocol stacks until it is managed by the *dsragent.* The packet received is processed by the receiving function. In particular, it controls the packet's content in order to decide how to handle it. For example, if a node receives a packet that is not for itself, probably it should be forwarded to another node. On the contrary if it is for the node, the packet is managed in another way. Anyway, in both cases the *dsragent* has been implemented to lead both situations.

Since *dsragent* is an open source code a researcher can modify or add some features. For example adding the following code into the *dsragent::recv* method a node will check the topology of the packet, if it results *CBR-like*, then *dsragent* performs a particular action.

```
if (cmh->ptype() == PT_CBR)
{
    ...
}
```

Always within the receiving method, a node is able to manage a Route Error *(RERR)* generated by another node as it is explained in the chapter (3.2). A *RERR* is sent by a node as a particular packet when it wants to inform neighbours about one or more link broken. When the packet is made, the sender adds into the header all information required by a *(RERR)* packet.

On the other side, when a node receive that packet, the latter one is processed inside the receiving function. In particular, the condition that triggers the function for managing that packet is the following:

```
if (... && srh->route_error())
{
   processBrokenRouteError(p);
}
```

The method called *processBrokenRouteError(p)* handles the *RERR* and its main task is to remove from the connection's table of the receiver the link or links that now are not available anymore.

Finally, going towards the end of the method *recv* after all the controls listed before, an *agentDSR* manages the forwarding of a packet done during a Route Request *(RREQ)* or a normal packet forwarding.

```
...
handleForwarding(p);
...
```

All methods listed before have been deeply analysed during our research. Moreover, new code has been added in order to insert some relevant features for our goals.

Another model of wireless networking that we considered is the energy one. Respect to a wired device a wireless one must take into account its own energy and to manage this *NS2* provides an energetic model. For example each time that a node sends or receives a packet its own energy decrease. In order to make that as real as possible, developers of *NS2* created the object *energy-model.cc*.

To adopt it in a simulation a user must add the following lines of code into the *TCL* script:

```
...
set val(energymodel)     EnergyModel
...
$ns node-config
 - energyModel $val(energymodel) \
```

To give to the energy model more accuracy it is possible to specify other parameters, such as the initial value of energy, the energy used for sending or receiving a packet. These values can be always inserted using the *TCL* script.

```
...
set val(rxPower)         1.0
set val(txPower)         2.0
set val(initialenergy) 200
...
$ns node-config
...
 - initialEnergy $val(initialenergy) \
 - rxPower $val(rxPower) \
 - txPower $val(txPower) \
...
```

The majority of those parameters are optionals and if a user does not how to set them, then *NS2* will apply the default values. However, the *initialEnergy* parameter is used to initialise the amount of energy *(in Joule)* that each node has at the begin of the simulation. Instead, the other two parameters *rxPower* and *txPower*, which are measured *in Watt*, express the amount of energy used for sending or receiving a packet.

Nevertheless, we want to point out that the energy model used in NS2 is quite simple. Indeed, the model misses in some aspects respect to the real way of handle the energy in a device. For example, in the simulator, when a node send or receive a packet its current value of energy is decreased only of a constant value. Instead, considering in the real case the model considers the distance between nodes too.

Going to the C++ implementation of the energy model, part of code is shown below:

```
void EnergyModel::DecrTxEnergy(double txtime,
                                    double P_tx)
{
        double dEng = P_tx * txtime;
        if (energy_ <= dEng)
                energy_ = 0.0;
        else
                energy_ = energy_ - dEng;
        if (energy_ <= 0.0)
                God::instance()->ComputeRoute();
```

```

void EnergyModel::DecrRcvEnergy(double rcvtime,
                                double P_rcv)
{
        double dEng = P_rcv * rcvtime;
        if (energy_ <= dEng)
                energy_ = 0.0;
        else
                energy_ = energy_ - dEng;
        if (energy_ <= 0.0)
                God::instance()->ComputeRoute();
        // This variable keeps track of total energy
        // consumption in RECV mode..
        er_=er_+dEng;
}
```

As it is possible to observe in both situations the current level of energy it is decreased of the amount of energy set before in the *TCL* script. It is also true that when the value of energy of a node is less than zero, it cannot send or receive any other packet.

We tested for long time this class and during the simulations we found that the following row of code is not run as it should be.

```
God::instance()->ComputeRoute();
```

In particular, *God* is an object of the wireless part of NS2, it has been implemented to manage wireless devices in order to keep trace of them in a easy way. In the frame of code above case the energy model calls *God* for establishing a new route that avoids devices without energy. Anyway, we recognised that this is not really done since the object *God* is not properly created by the simulator and as consequence that task is performed in a different way.

Using always the energy model, a node has the ability to switch its state during its activity: for example a device in order to save energy can switch its own state from active to sleep. The reason why a node changes its state is to save part of its energy because a node looks apparently unreachable and it cannot receive or send any packet.

The energy spent for each change can be set using the following code:

```
...
$ns node−config
...
 − sleepPower          0.100  \
 − transitionPower     0.200  \
 − transitionTime      0.100  \
 − idlePower           0.050
...
```

The *C++* class that implements the changing of a state it is shown in the next frame.

```
void EnergyModel::set_node_state(int state)
{
         switch (state_) {
         case POWERSAVING:
         case WAITING:
```

```
                        state_ = state;
                        state_start_time_ =
                                Scheduler::instance().clock();
                        break;
                case INROUTE:
                        if (state == POWERSAVING) {
                                state_ = state;
                        } else if (state == INROUTE) {
                                // a data packet is forwarded,
                                // needs to reset
                                // state_start_time_
                                state_start_time_=
                                Scheduler::instance().clock();
                        }
                        break;
                default:
                        printf("Wrong state, quit...\n");
                        abort();
                }
}
```

In this section we presented some of the main features that we used for long time throughout our research. Although in the last years the simulator has been updated and, especially, the wireless part improved a lot, we have noticed that the simulator misses in some aspects. In particular, we think that the structure, in terms of classes, is not well detailed. In fact, every time that a developer has a new module ready to add into NS2, it is checked by the *NS2* committee in order to check if there are important bugs or lacks.

Event though the new module is validated it is not always accurate as it should be and because *NS2*, nowadays, has become a large software it is not easy to keep trace of all modules, and as consequence sometimes it happens that the real validation of a new module is done by users.

Using *NS2* we found different weird problems, above we presented the problem of the object *GOD* and although this one should be used during the simulations, in reality for unknown reason it was not initialised at all.

Then, when we started to use the energy model we noticed another strange behaviour of the simulator. As we shown before each node that receives or sends a packet decreases its amount of energy of a particular value. When the current amount of energy is zero a node should be disconnected from the network. That behaviour works properly if the energy is decreased "naturally" and as consequence of this disconnection the routing protocol acts in order to create a new path.

In our research work we needed to change manually the current value of energy of a selected node and according to this we decided to add into the energy model of *NS2* a banal method. That was able to change the current energy in a different value expressed as parameter. Although that one is a negligible modification of the model, we observed that the system did not work in the right way. In fact, the routing protocol did not change the path and it forced to go towards the same node, even if it ran out of energy.

We investigated for a long time this problem but we did not find the way to fix it. Finally, in order to follow our research project we had to undertake another way and this strategy is described in the chapter (7).

Chapter 5

How to establish the reputation of nodes

With the proliferation of mobile lightweight terminals, like smartphones or PDAs, demand of communication using wireless short/middle-range channels is grown. As we said before, one of the main features of a MANET is the collaboration between its participants. Collaboration is necessary to allow multi-step communication. In fact, the physical layer of a wireless system imposes strict limitations to a direct connection. So that pairs of nodes, which are too far, can exchange information only through the cooperation of other intermediate nodes. According to this the packets flow through the network via hop-by-hop routing. This clearly requires that a transit node uses its own energy to provide the connectivity used by other nodes. The transit node does not receive any direct advantage in such a collaboration: if it does not forward packets, it saves its energy, but the network falls down. Because energy, in mobile devices, is often provided by a small battery, nodes are attracted to assume a selfish behaviour, using energy only for personal convenience.

In such a scenario, it is important to focus our attention on the conduct of the nodes, analysing their behaviour in order to define policies that encourage collaboration. A way to summarise the behaviour of a node is to establish its reputation, observing its involvement in other communications. The literature features various definitions of reputation. For example, in [31] J. Liu and V. Issarny say that the reputation toward an agent can be seen as a prediction on that agent's future actions. It is also true that the meaning of reputation and trusting in the literature has changed repeatedly and there is no a global consensus [32]. It depends on the context in which is used. March [33] was the first to advance a formal concept of trusting and a model that does not include reputation.

In [34] Bistarelli and Santini give a definition of reputation "*based on recommendations received also from other nodes*" advancing the concept of multitrust. The trust that a node puts in another node will vary when the latter collaborates with other nodes.

S. Buchegger and J. Le Boudec in [35] introduce a model to help the isolation of misbehaving nodes and to avoid that good nodes do not collaborate with malicious ones.

To enforce collaboration, Michiardi and Molva [36] suggest a protocol to prevent and to individuate selfish nodes. The method CORE calculates the reputation of all nodes and detects selfish behaviour. It also helps the nodes to share such information.

The concept of reputation finds a big interest in the grid field too. In fact, E. Papalilo and B. Freisleben in [37] introduce a mechanism to analyse the behaviour of the participants within Grid Computing. They consider nodes that can be *good* or *bad*, defining the first one as node that after an interaction with another one, it gives a normal expectations from the request of collaboration. Otherwise, the node is considered *bad*. The analysis of the behaviour of a participant is called by the authors *direct behaviour trust* and it is calculated under different trust elements, such as *availability*, *accuracy*, *throughput* and others.

In our work the concept of "node reputation" is essentially a measure of the collaboration provided to maintain the network connectivity, forwarding messages of different senders. According to these requirements we defined the following definition of reputation:

Reputation of a node is the degree of collaboration (in terms of packets forwarding) that a node provides to the network.

With this definition, the value of the reputation can be used as a starting point to generally incentive collaboration, or to define a power-save routing protocol. Quite simply, if a node with a low reputation wants to initiate a communication, its request obtains a low priority and therefore low resources. This idea is not far away from the strategy used in P2P protocols, like [24, 25]. In the latter field, if a node takes a low reputation, because it does not share its resource, then it will suffer a low Quality of Service *(QoS)*. Within the MANET the QoS plays a fundamental role since a node can obtain a good service only if the other ones give their availability. The relationship between helpfulness of collaboration and reputation is strong and the improvement of one depends on the other.

A way to analyse how much a node is collaborative is through its observation. Especially, the neighbourhood of a node can represent a good starting point to observe a device. Moreover, the result of this observation must be as accurate as possible and it must be similar in order to realise a sound mechanism.

An evaluation of the neighbour reputation can be quite simple using direct observation. The main problem in using reputation in a distributed environment without central coordination such as a MANET is the circulation of the reputation information over the network. A network node must have values respectively indicating the node's perceived reputation of each another node. A viable reputation protocol should keep the reputation values of all nodes about a specific node as similar as possible.

A node can take a selfish behaviour in a position, and then move into another region of the network taking a collaborative one. The circulation of its reputation towards all network nodes can disincentive such a double-face conduct.

Generally speaking, in a distributed environment an agent (a node for us) can take one out of three different behaviours, according to the BUG threat model [38]. By recasting that model when the goal is reputation, if follows that each node can be:

- good: when it uses its resources, as energy, to provide services to other nodes without any direct interest;
- bad: when it is essentially selfish, and damages the network intention-

ally by neglecting or limiting particular communications.

- ugly: when it essentially assumes a good or bad behavior according to its cost/benefit analysis of the context.

It is clear that only when acting as good, does a node provide support to the network, forwarding packets of other nodes. When a node takes a bad role, it can cause loss of data and the network can be split, isolating a number of nodes. This type of behaviour strongly depends on the context where ugly nodes operate, as they may decide to collaborate or not.

5.1 The Proposed Protocol

In order to assess reputation, we introduce a protocol to evaluate the behaviour that a node keeps. The protocol initially relies on the local perception of a node, and then refines it by the recommendations provided by other nodes.

5.1.1 Reputation by Direct Observation

The basic idea of our protocol is that a node reputation grows only if it forwards packets of another sender, calling "sender" the node which the communication starts from. The node under analysis has no interest in that connection: on the contrary, it uses its energy only to forward others packets.

Let us consider a simple scenario: a node with only two neighbours (node B in fig.5.1) into the route of a multistep communication ($A \rightarrow C$). It forwards packets received from node A to node C (because communications are often bidirectional, the roles of A and C are inverted in the response phase).

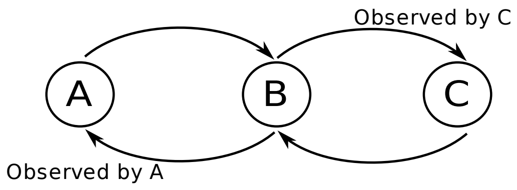

Figure 5.1: A simple model of comunication

Node A can compute B's reputation observing the packets it receives from B. A cannot perform a direct measure of the packet it sends to B that must be forwarded to C; it can obtain an indirect response of this step observing the return traffic, which is composed by packets that B forwards to A. With this method, a unidirectional traffic in a multistep communication, involves a growth in the reputation of the j-th node known by the (j-1)-th. Clearly, a bidirectional traffic provides more information. It is obvious that a node considers its incoming traffic for reputation computation only if the origin is at least two steps far. This means that a node discards, for reputation purposes, any traffic starting from its direct neighbours because these have direct interest in those communications. So this traffic is normally forwarded, but it does not contribute to the reputation measurement.

The information acquired by direct observation are stored in a table, called Neighbour Reputation Table *(NRT)*, which keeps track of the data amount received from closest nodes. The data that are stored in that table are separated into two different categories: *forwarded* and *generated* (fig.5.2 can help to understand better the difference).

In the picture, an application is presented inside a node j and it is able

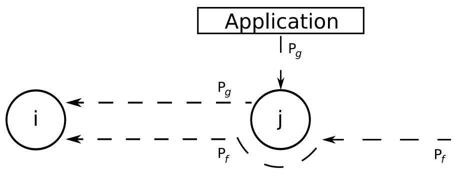

Figure 5.2: Different packet handling

to send message to other devices. When a node i receive a message from a node j, it checks the content of the packet. If i sees that the sender of the message is j, then it stores the size of the packet as generated, otherwise as forwarded.

Due to the fact that packets in a bidirectional communication can be constituted also by a simple *ack*, the values stored in the NRT must consider the number of packets received and their size.

```
void ReputationTable::incrForwarded(int id, long int B)
{
          Tcl& tcl = Tcl::instance();
          int existing_entry = find(id);
          if (existing_entry >= size)
                    existing_entry = getEntry(id);
          table[existing_entry].DF += B;
          table[existing_entry].PF++;
          return;
}

void ReputationTable::incrGenerated(ID id, long int B) {
          Tcl& tcl = Tcl::instance();
          int existing_entry = find(id);
          if (existing_entry >= size)
```

```
                existing_entry = getEntry(id);
        table[existing_entry].DG += B;
        table[existing_entry].PG++;
        return;
}
```

The two methods above implemented in *C++* for *NS2* allow a node to distinguish between packets generated and forwarded during a direct observation. The amount of a data packet is expressed in *Byte* and when one of two methods is called the observer node increments the counter for that observed node, which is defined by an *ID*. As we are going to show later, the table of each node contains two different entries for counting the data generated or forwarded. The first one stores the packet size, instead the last one only counts each packet received by that node, e.g. a simple *ack*.

The following figure shows an example NRT:

Node	**Forwarded**		**Generated**		R_{loc}
B	47	10	60	20	3,74
D	160	109	40	8	30,10
E	15	8	20	5	5,70
P	30	9	203	157	0,34

Table 5.1: NRT of a generic node

The *C++* implementation of the previous table is the following:

```
struct RepTable
{
        ID net_id;
        long int DF;
        long int DG;
        long int PF;
        long int PG;
        float rep_loc;
};
```

Starting from this information, we derive the *Local Reputation* using the following definition:

$$R^i_{loc}(j) = \frac{aP^i_f(j) + bD^i_f(j)}{aP^i_g(j) + bD^i_g(j)} \tag{5.1}$$

where $P^i_x(j)$ represents the packets that node j sent towards node i, e.g ack-packet, and $D^i_x(j)$ represents the amount of data that they carry. The sub-index g identifies data that are generated by node j; instead, sub-index f identifies data that node j forwards (i.e. the source is different from j). See fig.5.2.

The parameters a and b let us give different weights to number of packets and amount of data received.

The formula expresses the local reputation of node i about node j. It shows the ratio of the amount of information forwarded by j and the amount of information that it generated. From the point of view of node i this ratio represents the degree of unselfishness in the use of the link j-i. The higher the amount of traffic that j generates, the lower its reputation. On the contrary, the higher the amount of data forwarded by j, the higher its reputation.

The above expression has been implemented in each node in order to establish the *Local Reputation.* At the begin of the method a node checks whether the observed device is already present inside its own table. If it does not appear then *Local Reputation* is initialised, otherwise its local reputation is updated using the 5.1 formula.

```
void ReputationTable::update_local_rep(ID id)
{
  int index = find(id);
  if (index != size)
  {
    float div, rep;
    if (table[index].PF == 0 && table[index].PG == 0)
    {
      table[index].rep_loc = 0.0;
      return;
    }
    div = par_a * table[index].PG +
          par_b * table[index].DG;
    rep = (par_a * (float) table[index].PF + par_b
                 * (float) table[index].DF) / div;
    table[index].rep_loc = rep;
  }
  else
     printf("Error in updating local reputation");
}
```

We must point out that this definition for the reputation implies a bounded view of the node under observation; clearly, every neighbour of that node could calculate a different value for its reputation. So, it is necessary to introduce a protocol to mix these values, to obtain a uniform one. Also, it is necessary to scatter this information towards far nodes, that are not directly connected to it. This issue is addressed below.

5.1.2 The Global Reputation Table

The *NRT* contains only information on one-hop-far nodes. Every node maintains also the so called Global Reputation Table *(GRT)*, that stores information about all nodes in the network.

The aim of this section is to present a methodology to maintain the values stored in these tables almost uniform all over the network, so that they can be used to implement policies that avoid (or limit) selfish behaviour.

The structure of the *GRT* (see table 5.2) is very simple: it contains an entry for every known node in order to store its reputation. There are two different ways to determine these values:

- if the node is not a neighbour, the value is calculated from the information scattered by other nodes;
- if the node is directly connected (and can be observed), the value is calculated using the personal *NRT* and remote information. This procedure assures that local reputation is tuned using remote observation.

The exact expressions to be used will be presented in the following.

5.1.3 Reputation Table Exchange

As we described above, every node of the network constructs an *NRT*, tracing one-hop-neighbours nodes, and a *GRT*, for neighbours and far nodes. Periodically, the information in the *GRT* is shared with the others, exchanging such tables. This protocol allows to scatter the value of a reputation throughout the net, in order to populate the *GRTs* with information on all participants to the MANET.

According to [31], we call *Recommendation* about x from y the reputation established by y regarding x. So, we will indicate with *Recommendation* a reputation value acquired from the *GRTs* of the neighbours.

Table information is not broadcasted all over the net with a flooding procedure, but precisely. Every node that receives a table from one neighbour, calculates new values, and then, with a prefixed schedule, it sends the new *GRT* to its neighbours.

This kind of sharing limits the traffic in the net, avoiding the overload and limiting the use of energy. However, it can involve a non-uniform distribution of the same value. This issued can be addressed by choosing the correct time interval between sending the *GRT*, and launching the reputation update procedure described below.

The function that allows a node to spread the reputation table to all neighbours is realised in *C++* in the following way:

```
void DSRAgent::sendNeighbor()
{
  int num_entries = rp->get_ptr();
  if (num_entries <= 0)
    return;
  RPPacket rrp;
  rrp.dest = net_id;
  rrp.src = net_id;
  rrp.pkt = allocpkt();
  hdr_rp *rph = hdr_rp::access(rrp.pkt);
  hdr_ip *iph = hdr_ip::access(rrp.pkt);
  hdr_cmn *cmnh = hdr_cmn::access(rrp.pkt);
  iph->daddr() = Address::instance().
        create_ipaddr(IP_BROADCAST, RT_PORT);
  iph->dport() = RT_PORT;
  iph->saddr() = Address::instance().
        create_ipaddr(net_id.getNSAddr_t(), RT_PORT);
  iph->sport() = RT_PORT;
  cmnh->ptype() = PT_RP;
  cmnh->size() = size_ + IP_HDR_LEN; // add in IP header
  cmnh->num_forwards() = 0;
  cmnh->direction() = hdr_cmn::DOWN;
  char* strTmp = (char*) malloc(2048);
  strTmp[0] = 0;
  for (int c = 0; c < num_entries; c++)
  {
    RepTable rpt = rp->table[c];
    rp->update_local_rep(rpt.net_id);
    if (rpt.rep_est > 0.0)
    {
      sprintf(strTmp, "%s\n_%ld:%f", strTmp,
               rpt.net_id.addr, rpt.rep_est);
    }
  }
    sprintf(strTmp, "%s\n", strTmp);
    PacketData* data = new PacketData(2048);
    strncpy((char*) data->data(), strTmp, 2040);
    rrp.pkt->setdata(data);
```

```
    Scheduler::instance().schedule(ll, rrp.pkt, 1);
}
```

In this case we added the method inside the *DSRAgent*, in particular a new structure, called *RPPacket*, has been done in order to contain the table with all *Recommendations* about the observed devices.

Afterwards, when the packet is ready, it is sent in a broadcasting way to all neighbours one-hop far.

5.1.4 How to estimate the Global Reputation

Expression 5.1 introduced the local reputation $R^i_{loc}(j)$ estimated by a node i for one of its neighbour nodes j. Let $R^i_{est}(j)$ be the value stored in the *GRT* of i regarding j, and let $R_{rec}(j)$ be a recommendation received from a neighbour different from j.

The new value to store in the *GRT* will be:

$$R^i_{new}(j) = w_l R^i_{loc}(j) + w_r(w_2 R^i_{est}(j) + w_3 R_{rec}(j)) \tag{5.2}$$

where w_l, w_r, w_2, w_3 are adequate weights, being $w_l + w_r = 1$ and $w_2 + w_3 = 1$.

In particular:

- w_l: is the weight applied to the local reputation.
- w_r: is the weight applied to the indirect observation.

- w_2: is the weight applied to the reputation calculated by an observer.
- w_3: is the weight applied to the recommendation values get by others.

For far nodes, for which $R^i_{loc}(j)$ does not exist, the above expression simplifies as follows:

$$R^i_{new}(j) = w_2 R^i_{est}(j) + w_3 R_{rec}(j) \tag{5.3}$$

The accuracy of the result of the expressions 5.2) and 5.3) depends on the nearness to the device to observe and from the parameters that are used to establish the reputation. In the first case, the possibility to examine the behaviour of a node directly helps the device to give more soundness to the formula's result. In addition, a good tuning of the parameters w_l, w_r, w_2 and w_3, represents a reliable way for scattering the table' values upon the net. As consequence, the local reputation cannot be underestimate because it symbolises the phase wherein an observer node analyses directly the behaviour of another one. On the other hand, the value of recommendation is considered as relevant factor since it merges the node's observation with those of the other devices. In order to obtain thorough values of reputation, both parameters must be balanced, even if, it is convenient to give a little bit higher weight to the local reputation respect to the recommendation. This choose represents the suitable tradeoff between accuracy and spreading of reputation and this is pointed out through further simulations.

The code to update the reputation of an observed device in function of its *Recommendation* is implemented in the *ReputationTable::update_rep_est* method. For each node, which is represented by its *ID*, is received a *Recommendation* and it is used to update its value of reputation. In particular, if an observer node has the local reputation of an observed one it uses the 5.2 formula. Instead, whether an observed is a far device the expression 5.3 is applied.

```
void ReputationTable::update_rep_est(ID id, float new_racc)
{
  int index = find(id);
  if (index != size)
  {
    float remote, new_est;
    remote = par_w2 * table[index].rep_est + par_w3 * new_racc;
    if (table[index].rep_loc > 0)
        new_est = par_wl * table[index].rep_loc + par_wr * remote;
    else
        new_est = remote;
    table[index].rep_est = new_est;
    } else
    {
        table[ptr].net_id = id;
        table[ptr].DF = 0;
        table[ptr].DG = 0;
        table[ptr].PF = 0;
        table[ptr].PG = 0;
        table[ptr].rep_est = par_w3 * new_racc;
        table[ptr].rep_loc = 0.0;
        table[ptr].racc = 0.0;
```

```
            ptr = ptr + 1;
        }
        return;
}
```

In addition to expression 5.2), when a node realises that the reputation of another node has not changed from the previous step, because the observed node is idle or it rejects collaborations, the observer applies the *Old-Age Function*, to decrement the reputation in the *GRT*. In this way if a node stops collaborating, its reputation decreases over time. In consequence, a node with a good level of reputation is encouraged to maintain a high level of involvement in the packet routing.

Node	$\boldsymbol{R_{loc}}$	$\boldsymbol{R_{est}}$	$\boldsymbol{R_{rec}}$
S	47,50	45,12	41,12
F	15,21	14,65	15,01
G	unknown	8,09	9,57
S	70,09	65,12	78,44
H	unknown	40,09	45,8
U	unknown	1,02	1,57
Y	unknown	0,78	0,56

Table 5.2: GRT of a generic node

In the table 5.2 is shown the *GRT's* content of a generic node. The fields that are used in the expressions 5.2) and 5.3) are those that belong to the *GRT* of a node. Some information is missing in the table, like R_{loc}, because for nodes *G*, *H*, *U* and *Y*, the direct observation is not available. This latter happens when the nodes are far away and, therefore, the only way to receive information about them is only through recommendation. More specifically, the field used for establishing the reputation in 5.2) and 5.3) by a node that receive a GRT is the R_{est}.

A first set of simulations was conducted to find suitable values for the parameters of expressions 5.2) and 5.3). The correct choice for them is when the distribution for the reputations stored in the GTRs is sufficiently uniform.

5.2 Simulation and Performance Evaluation

The entire reputation protocol described above was tested through simulations in order to make the optimal choices for the relevant parameters and ultimately to validate the protocol itself. We used the *Network Simulator 2* (NS2) [39], which is a de-facto standard to analyse network protocols. According to this in a study done by Camp at all. claim in [40], *NS2* is the most widely used simulator in the research field.

NS2 is an event-driven simulator that implements packets, nodes, links, transport and application agents. It provides a reliable structure that allows a researcher to model large wired and wireless systems. Also, it implements modules for wireless stations, which can move themselves in a 2D environment. For these stations, NS2 defines some routing protocols, such as DSR, AODV and TORA [41].

The power of NS2 is its flexibility. A user can simply add his own ad-hoc modules to test new protocols. This feature allows us to define some modification to the original DSR agent which turn out to be necessary.

As we showed above, the expression 5.2 contains some parameters that allows a node to calculate the reputation giving more relevance to the local reputation instead of the recommendation or vice versa. The only way to know which are the correct values to assign to the parameters is through simulations. In order to establish the suitable value a lot of tests were done. Moreover, the majority of the charts are obtained as a result of 100 runs and the findings are characterised by a 95% confidence interval.

The reference scenario is constituted by a network with 25 nodes, distributed in a space of 1000 x 800 m^2. Each node has a cover range of 200m and is connected with other 6 nodes (see fig.5.3).

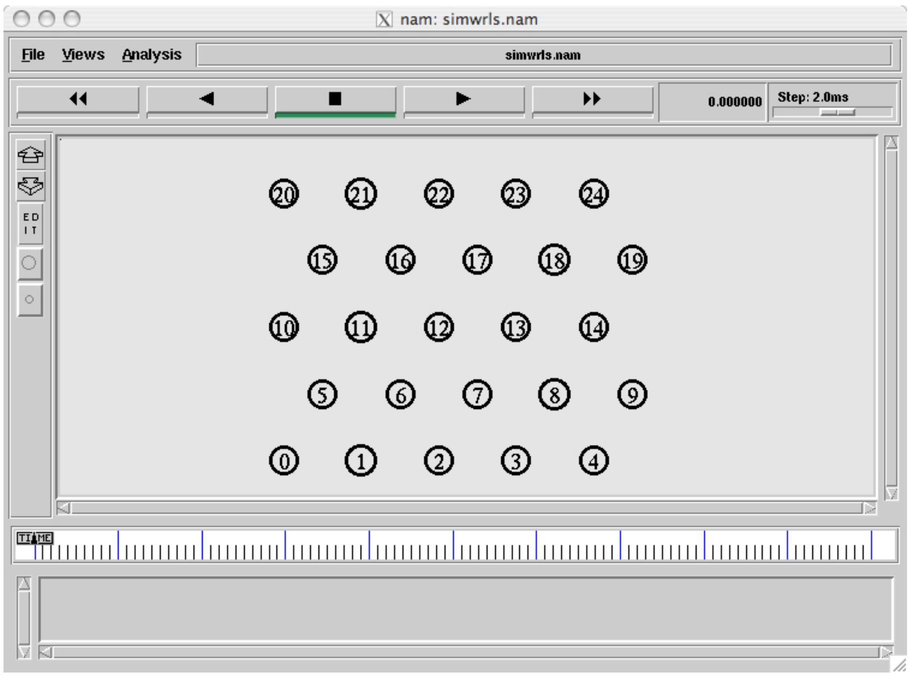

Figure 5.3: Reference Network Topology

We used the same scenario also to evaluate the distribution of the reputation for a particular node. Following we show as the behaviour of the observed node is established by other devices.

5.2.1 An observed good behaviour

The goodness of a node is provided by its own collaboration during the forwarding of a message. We consider that the higher is its availability to collaborate the higher is the reputation acquired by a node.

Fig.5.4 shows the trends of the reputation for an observed node, analysed by the others.

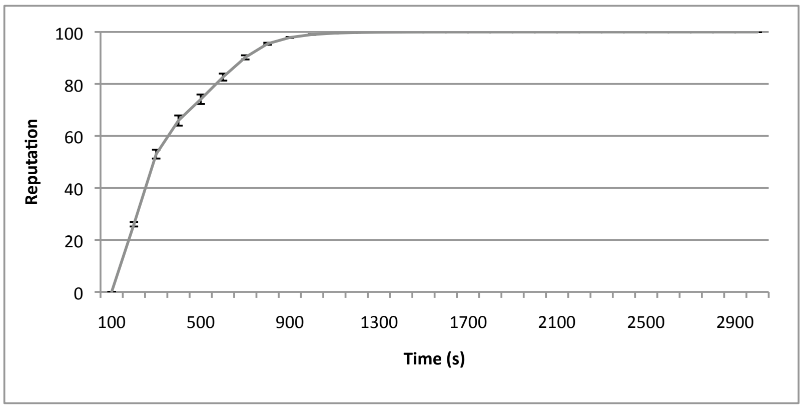

Figure 5.4: Reputation distribution for a good behaviour

In the image is highlighted that the node has taken a good behaviour during the simulation. In fact, the life of the node is fully dedicated to the collaboration and its reputation runs up until to reach the maximum value. For the sake of presentation, it is convenient to observe the node during its two different states:

- *transient state* has the node start exchanging the GRTs;
- *steady state* has the distribution of the GRTs values almost uniform and the node behaving as good;

Fig.5.4 refers to a values of reputation achieved by the observed node wherein the simulator considers the recommendation value is similar but non equal to the local reputation. This allows an observer to analyse carefully the behaviour of the node and, also, the observations obtained can be spread quickly.

5.2.2 An observed bad behaviour

When a device uses its resources only for its own goals does not give any contribute to the net. Collaboration for a mobile ad hoc network is one of the most important features. If some nodes decide to take a non fair behaviour the net will not work properly.

Findings of these simulations show as a device that is taking a bad behaviour is recognised by the community.

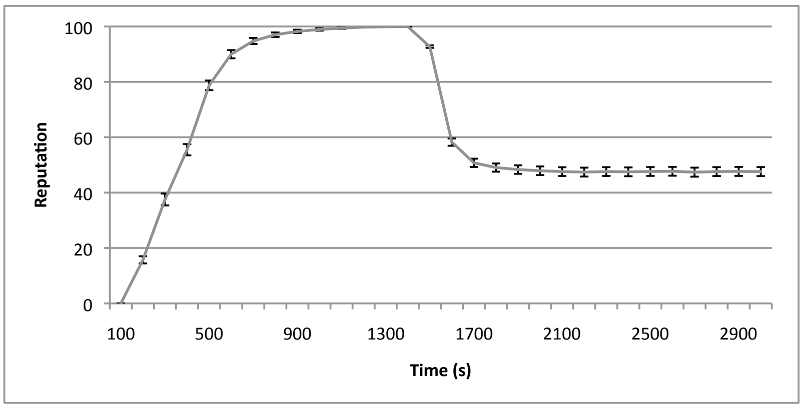

Figure 5.5: Reputation distribution for a bad behaviour

Fig.5.5 depicts the behaviour of the observed node during its activity in a set of simulations long 3000 seconds. From the image it is possible to point out that the device has taken three different behaviours. Firstly, it has been collaborative in order to increase its reputation till a maximum obtainable value. Next, its reputation has gone down; this because it stopped collaborating and then it started to generate data for its own goals. Finally, the reputation of the node decreases very slowly since it is idle.

In this case too, the presence of a good weight for the recommendation allows the protocol to spread the correct reputation of the node earlier.

5.2.3 Joining the network

A MANET is a net characterised by frequent changes of the network topology. Several nodes join the net and other ones leave it and the possibility to recognise quickly the reputation of a new joined node is an actual challenge. The aim of these simulations is to localise the attention on the behaviour of a node that appears in the network after the initial phase. When the device joins the net its behaviour is completely unknown and its oncoming reputation depends on itself. Otherwise, if the device was identified in a previous moment and then it rejoins the net, its degree of reputation is in function of its past behaviours.

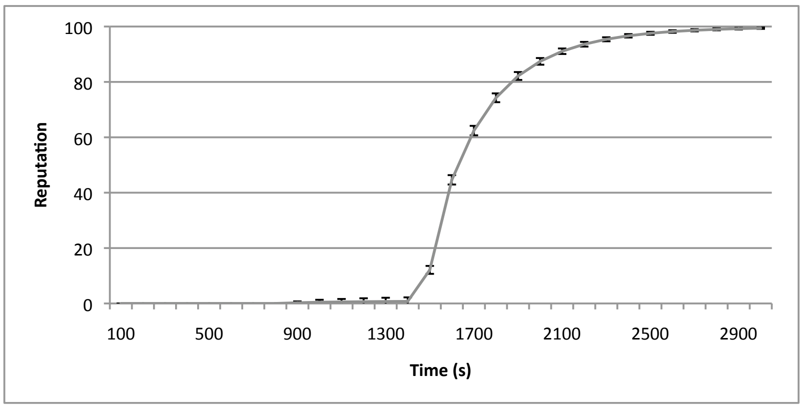

Figure 5.6: Reputation distribution for a good joined node

Fig.5.6 shows a node joined the net after 1400 seconds of simulation. Its reputation raises up till the top of the chart and it never goes down because the node has taken a good behaviour. Moreover, it is important to highlight that in all test of simulations the observer nodes established the new node's behaviour in a correct manner and, in particular, this result is supported by the small confidence interval.

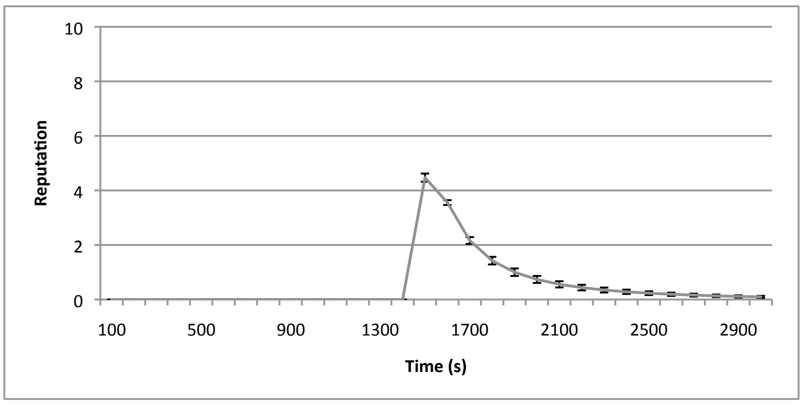

Figure 5.7: Reputation distribution for a bad joined node

The opposite behaviour of a new joined node is shown in fig.5.7. When the device takes part in the MANET, it obtains a value of reputation that depends on the mean of reputation values of all nodes belonged to the net. This choice was taken to avoid of penalising new nodes assigning to them a initial reputation value too low. In the image, the value assigned to the joined node is shown by the peak presented in the chart. Nevertheless, the node lived taking a bad behaviour and this is depicted by its own reputation function.

5.2.4 Uniformity of reputation value

It is also significant to check the spatial distribution of the reputation, which is drawn in fig.5.8. The floor of this graph is constituted by the network topology, like in fig.5.3. Here we highlight only the observed node (*node 12*) and its one-hop neighbours. As the reader can see, close nodes have the same value of reputation. The difference for the zones are due to the path followed by the communications.

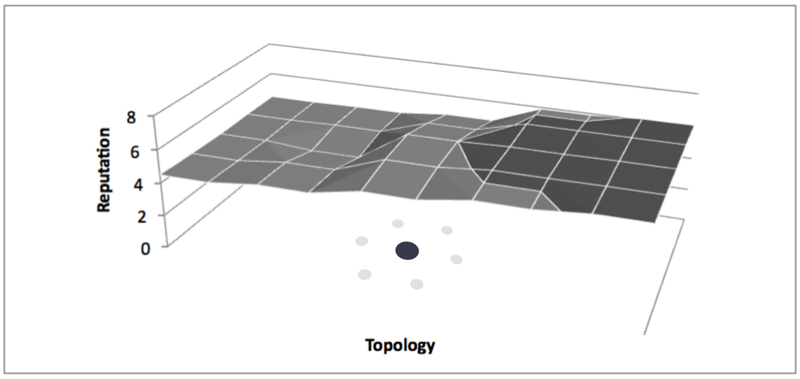

Figure 5.8: Reputation distribution for a selfish behaviour

Moreover, the result obtained in this image is like a screenshot taken upon the reputation of the observed node throughout the steady state. So, even far away nodes can obtain accurate information, through recommendation, about the observed node.

5.2.5 Working of the Old-Age Function

We have briefly mentioned before that our reputation mechanism uses the *Old-Age Function* to prevent nodes to hold its own good reputation without collaborating anymore. This situation could verified when a node in the past has developed a collaborative role and then it decides to quit its collaboration. When at least one of the one-hop neighbours realised that the observed node does not have changed its amount of forwarded information for a long time, then the observer node runs the *Old-Age Function.* The elapsed time to execute the function must be bigger than a threshold called *t*. When it happens, that function uses a fixed parameter to decrease the value of reputation stored inside the observer node. The effect of the parameter depends on its value and, in addition, it is equal for each node and cannot be changed by devices.

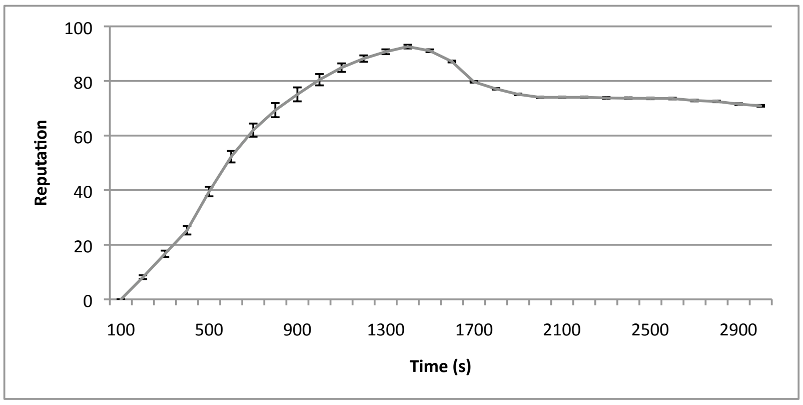

Figure 5.9: The *Old-Age Function* effect in a reputation distribution

Fig.5.9 shows the result of the *Old-Age Function* applies to a reputation distribution of a node. The trend present in the chart after the instant 1700 is due to the function. In fact, the node observed in the simulation was set as idle, so neither generation nor forwarding of information can be done by the node. Moreover, the degree of the inclination line is strongly influenced by the value of the parameter used in the simulation. In fact, with a different value it is possible to increase or decrease the effect of the *Old-Age Function.*

Chapter 6

Enforcing collaboration with a QoS technique

As we said in the previous chapter, every mobile device can ”live” until it gets energy from its battery. Even if in the last years batteries have grown in capacity, they remain the main bound for mobility. Every device must implement policies for power-saving in order to extend its *lifetime*. The author of an important book [42] asserts that the lifetime of a path P is determined by its weakest intermediate node and more precisely by the residual battery power of that node.

Reputation of a node has been the first step of this thesis and helped us to analyse the behaviour of a node. With this tool participants are to able ”to discover” every citizen of a MANET. Now we are going to focus our study on collaboration. The main goal of this is to extend the previous work, adding a new feature to the protocol.

To obtain it we define a modification for a generic MANETs routing protocol to incentive collaboration by promoting nodes that preserve the net connected. To determine the behaviour of a device the reference parameter is its *reputation*. We mainly refer to the definition given (§5) The latter concept is used in our work to analyse the behaviour of devices. In fact, as consequence a device with a bad reputation is an entity that in past has chosen to use its resources only for personal goals. As it happens in Peer-2-Peer *(P2P)* protocols, it is fundamental to push devices to take a fair play. In *P2P*, collaboration means sharing of resources. A user who is sharing its resource gets a "better" treatment from the community in terms of quality of the services it receives. We borrow this idea from the *P2P* area, and introduce in the area of MANETs a policy that penalises selfish nodes. Moreover, our modification to the routing layer introduce a method to extend the lifetime of the whole network.

An effort to introduce collaboration is presented by Yan and Lowenthal in *"Towards Cooperation Fairness in Mobile Ad Hoc Networks"* [43]. The authors show how to achieve fair bandwidth on the basis of nodes' collaboration. In fact, the lower is the degree of collaboration of a node, the lower is the forwarding bandwidth rate that it receives. Moreover, they suggest to use their *"coefficient of collaboration"* to find paths with a high value of collaborative nodes. The latter feature is perilous for the devices of the selected path because the energy of the nodes could be quickly consumed.

Although we are going to show our idea of collaboration in a MANET, we want to point out that the previous work is not directly related to ours. Indeed, the work presented by Yan and Lowenthal considers the bandwidth shared by a device as its degree of collaboration. In our work, we use the reputation to judge the behaviour of the participants taken in the net. Moreover, we take in exam the problem of power saving and provide our solution to extend the lifetime of the devices.

To explain why we decided to explore this sub-field, we must highlight another time a strong difference between Wireless Ad hoc Networks and wired ones. A distinctive feature of the first one is the lack of hard-wired devices. Moreover, the absence of a router and a switch dumps the homologous functionalities upon the participating nodes. In a wired net the positions of the nodes are fixed and until the device is alive it is certainly reachable. In such environment, the routing of messages is simple.

On the contrary, in a MANET, each device can move as it likes, thus constructing a highly dynamic structure such that the neighbourhood of a node can change very frequently. In this scenario the identification of a node could be troublesome because a device can assume either a collaborative or a non collaborative behaviour.

In the first behaviour, collaboration makes a net very useable but a node consumes quickly its own energy. On the contrary, the opposite behaviour leads to a longer lifetime but the functionalities of the MANET are severely compromised.

From a macroscopic standpoint, the energy that is used to keep the network connected must not be considered wasted. If we take a detailed view and look at each node in particular, a device can decide that the energy used for goals other than its owns is a bad investment and in consequence assume a selfish conduct.

We can draft different reasons why a node may want to avoid collaboration:

- *power saving*: to extend its lifetime a device uses energy only for its own communication;
- *throughput increase*: a bigger bandwidth is available to a node if it does not forward any packet;
- *denial of service*: a node could generate a fake link-broken not to participate, as forwarder, in a communication.

Starting from this analysis, the possibility that a device does not have interest to collaborate in the routing phase is an issue that we must nullify.

6.1 Optimising the nodes' use of energy

Energy is a critical resource for mobile nodes. The transmission phase is the most consuming one: the farther is the destination, the higher is the energy required.

6.1.1 Analysis on consuming during forwarding

In a generic forwarding of information, a node receives a new message from its input queue and elaborates it. In the latter phase, the node takes the message and checks if it knows a real path to the final node, either directly or through forwarding. We define ΔT as the time required to catch the message from the input queue, to elaborate it and to send it to a neighbour node.

The number of messages that a node elaborates in a second can be defined as follows:

$$N_{fps} = \frac{1}{\Delta T} \tag{6.1}$$

The correct value of N_{fps} *(Number of messages forwarded per second)* depends on how fast the device processes each message.

As we mentioned above, when a node turns on its transmitter to send a message, it consumes an amount of energy depending on the position of the receiver node. On average, we estimate that the energy consumed for each packet is ΔE. In addition, we define $E_{max}(n)$ as the maximum energy power that the node n can rely on its battery.

This analysis shows that the lifetime $L(n)$ of a generic node n is can be defined as:

$$L(n) = \frac{E_{max}(n)}{\Delta E * N_{fps}} \tag{6.2}$$

Expression 6.2 shows that the value of the lifetime depends on the quantity of available energy and on the amount of data that a node processes in a second.

Our goal is to increase the collaboration within the MANET. Moreover, our contribute provide a solution to extend the lifetime of all nodes. The latter is fundamental because it allows to give a helpful network and avoids potential link broken.

6.1.2 Consideration about lifetime

In [44] Kyoung-Ja Kim and Hyun-Woo Koo provide a method to save energy in DSR adding a delay due to multi-hops forwarding. In our work, we use a delay too because we consider it a fundamental feature to increase the lifetime of the net and to penalize non-collaborative nodes.

A new parameter, called δ, is introduced to represent the *delay* in expression 6.1. The following formula shows the updated expression:

$$N^d_{fps} = \frac{1}{(\Delta T + \delta)} \tag{6.3}$$

The definition of lifetime is refined to consider the delay.

$$L(n) = \frac{E_{max}(n)}{\Delta E * N^d_{fps}} \tag{6.4}$$

$$L(n) = \frac{E_{max}(n)}{\Delta E} * (\Delta T + \delta) \tag{6.5}$$

From 6.5, ΔE and $E_{max}(n)$ are fixed values and it is possible to deduct that the lifetime of a node depends on the values in the brackets. Because ΔT is considered constant during communication for simplicity, the delay δ plays the main role.

6.1.3 Our approach

Expression 6.5 shows that transmission delay and lifetime are two strictly correlated features. The parameter δ is used as a deterrent to enforce collaboration within MANETs. It works as adhesive between *Reputation* and Quality of Service *(QoS)*. Its main goal is to analyse the behaviour through the reputation and according to this latter it gets the suitable *QoS* to the device. A similar strategy is used in some p2p protocols [24, 25], where a bad user obtains a low *QoS*. In these protocols the *QoS* is installed in the application layer, instead we apply it routing layer.

The next section describes our modifications to the *DSR* protocol. We update the main phase of the protocol, which is Route Discovery, by our notion of collaboration awareness. Our updated protocol is perfectly backward compatible, i.e., devices that use our modification to the *DSR* protocol can coexist with devices that use the standard *DSR* on the same environment.

Route Discovery with Delay

The *DSR* protocol allows to create paths from a source to a destination node by means special packets called Route Request *(RREQ)*. Following we summarise briefly how a path is built. Nevertheless, this phase has been deeply explained before in the chapter (3)

When a node starts a communication, it sends a *RREQ* to its neighbours and every node that receives the *RREQ* searches locally if it already knows a route to reach the destination. If not, it forwards the *RREQ* in broadcast to all its neighbours, and so on until the destination is reached. At this point, the destination node sends back to the sender a packet called Route Replay *(RREP)*, which contains the route established. While the *RREQ* was distributed in a broadcast fashion, the *RREP* travels back to the initiator by simply following the path built with the help of the *RREQ*.

Starting from the expression 6.5, we define the sub-factors that compose the delay δ.

$$\delta = \delta_1 + \delta_2 \tag{6.6}$$

The first half δ_1 depends on the reputation of the source node and is defined as follows:

$$\delta_1 = \frac{rep_{max} * \Delta c}{rep_{src}} \tag{6.7}$$

Here, rep_{max} is the highest value of reputation that a node can obtain; rep_{src} is the value of reputation of the sender; the factor (Δc) allows the delay to scale inside a suitable range.

The expression above *(6.7)* is implemented through the following frame of code. In particular, if a node is not listed in the table, it will obtain a value of rep_{src} that is represented by the average of all reputation values collected until that instant. This has been introduced to cover the situation in which a node joins the network later and it could be unknown by the majority of the devices.

```
float ReputationTable::delayRep(ID id) {
        int idx;
        float delay = 0;
        float rep = 0;
        idx = find(id);
        if (idx >= 0 && idx < size)
                rep = table[idx].rep_est;
        else
                rep = repEstMeanValue();
        isDeltaInRange();
        delay = ((100 * deltac) / rep);
        return delay;
}

float ReputationTable::repEstMeanValue() {
        double mean = 0;
        if (ptr > 0)
        {
                for (int c = 0; c < ptr; c++)
                {
                        mean += table[c].rep_est;
                }
                mean = mean/ptr;
        }
        else
                mean = 1.0;
        return mean;
}
```

The variables δ_2 depends on the remaining energy of a node that must forward a transit packet. It is defined as:

$$\delta_2 = \frac{E_{max}(n) * \Delta c}{E_{bat}(n)} \tag{6.8}$$

Here, $E_{max}(n)$ is the maximum energy power of a node n, while $E_{bat}(n)$ denotes the current energy level of the device. Both δ_1 and δ_2 are expressed in seconds.

The method *(6.8)* is easier respect to the *(6.7)*. Indeed, in this case a node always knows its own value of energy and as consequence it can calculate the delay in one step.

```
float ReputationTable::delayErg(ID id, double curr_energy,
                                              double max_energy)
{
    float delay = 0;
    isDeltaInRange();
    delay = ((((float)max_energy) * deltac) / (float)curr_energy);
    return delay;
}
```

From 6.7 the value of δ_1, calculated by forwarding nodes, is similar during a communication because the reputation of the source is coherent in all devices. So, we consider δ_2 to avoid that the sender node uses in its communication the same path. In fact, in this way the nodes that are forwarding the message apply a bigger delay but them energy runs flat quickly. In order to avoid this problem we uses δ_2 as feedback information to warn the sender to change the current used path. The main idea of this approach is to allow to spread the battery power drainage not only upon the same path but into different routes.

Finally, we implemented the total delay (δ) as it follows:

```
float ReputationTable::delay(ID id, double e, double e_max)
{
        float delay = delayRep(id) + delayErg(id, e, e_max);
        return delay;
}
```

When a node receives a *RREQ*, before putting *RREQ* in the output queue, it computes the delay following definition 6.6. Next, it forwards the *RREQ* to its neighbours. A close node that receives the *RREQ* runs the same procedure and adds a new delay and so on. In consequence, the resulting path takes into account the energy of the traversed nodes and the reputation of the source known by the transit nodes. Such a delay enforcement is also done during the communication phase.

In the following we show our findings obtained through a set of simulations. Our main aim was to confirm that a non-collaborative node obtains a low QoS communication. Other tests indicate how effectively and quickly the protocol reacts to nodes' state changes.

6.2 Simulations

Our protocol depicted above was tested using *NS2*. Even in this situation we decided to run simulations through the aforementioned tool. Each simulation relies upon a network of 25 nodes placed in a fixed position in order to avoid bottlenecks or mobility. The lack of the latter feature does not hinder the soundness of the simulation results. In fact, the worst case is constituted by a node that decides to change its position and its identity because previously it has taken a non collaborative behaviour. In this scenario the device does not get any advantage because it is considered as new joined node and its reputation will be low. Otherwise, if it chooses to change only its position inside the net it keeps its low reputation. A scenario relating to a new joined device is deeply examined in the next section.

We decided to adopt a topology in which devices are *200m* far and the transmission range is sufficient to reach only one-hop neighbours. This layout implements our main aim of testing collaboration of the intermediate nodes. Following we are going to show different scenarios. in these testes the observed node assumed different behavours. Finally, we show how much a node pays in terms of *QoS* when it behaves like selfish or collaborative. We got the results using the full range of reputation available.

The topology of the net is the same used in fig.5.3.

Good Behaviour

In these simulations we consider an observed node that becomes active after 1500 seconds of simulation. This means that the node is active for only half of the simulation time, because each simulation lasts exactly 3000 seconds. We want to test if the observed node obtains a value of reputation that correctly reflects its behaviour.

In the results reported, each session between a source and a recipient is constituted by a Constant Bit Rate (CBR) traffic. The connections were realised to diagonally cross the topology of the net, so as to ensure that the most fundamental nodes were involved in the communications.

Our findings about a 100 simulations are depicted in fig.6.1. They are shown in a chart characterised by a 95% confidence interval. Moreover, it can be seen how the reputation of the observed devices grows up to the maximum value. In fact, the collaborative behaviour is quickly recognised by the others.

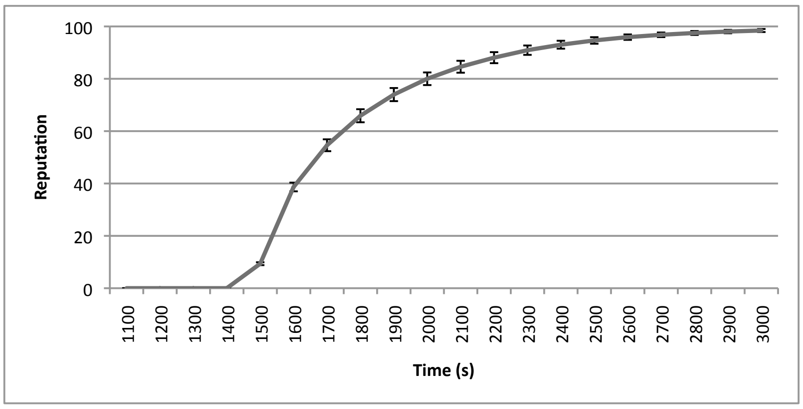

Figure 6.1: Reputation of a good node

Bad Behaviour

Also in this test, we show if the observed behaviour is established correctly by the others. The total simulation time is kept of 3000 seconds and the joining instance remains second 1500, but in this case the device takes a non collaborative behaviour.

The chart in fig.6.2 shows the reputation of the observed node decreases steadily to zero as soon as it is first set.

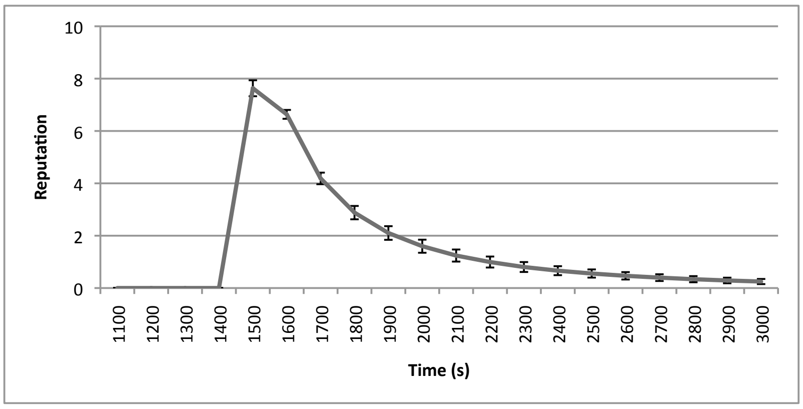

Figure 6.2: Reputation of a bad node

Note that fig.6.1 and 6.2 are represented with a different scale of the y-axis.

In addition, the simulations showed that one-hop-neighbours are the first devices to realise the uncooperative behaviour of the observed node.

6.2.1 QoS: collaboration or not?

Collaboration between nodes is the main feature to allow a good functioning of the net. Devices that did not provide a fair behaviour are penalised by our protocol.

Fig.6.3 shows the amount of delay that is added to a communication. The results only depend on the reputation of the sender.

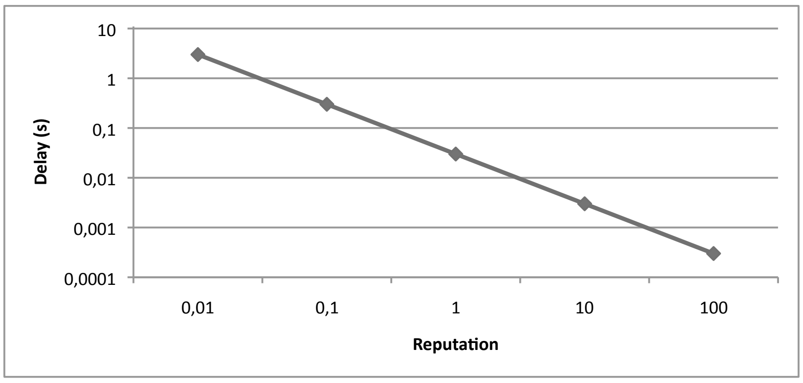

Figure 6.3: QoS depending on reputation of a generic node

In Figure 6.3 it is important to note that both axes follow the logarithm scale. The function is a straight line showing how the delay introduced to a node steadily decreases down to zero when its reputation rises towards rep_{max}.

A value of reputation between 1 and 10 considers a node that is taking a fair play but its degree of collaboration provided to the net is very soft. A reputation over 10 gets a full collaborative device and according its behaviour the node is recompensed with a low penalisation.

The range of delay proposed in the chart were fixed through simulations to avoid that nodes feel an inadequate delay. We observed that to send a message with three intermediate nodes, a communication needs about 0,05 seconds. This explain that considering a good device the penalisation added is very small. While, a bad one obtains an relevant delay, but the service to it is always assured. Otherwise, a delay too high makes hardly impossible the connection, like as Denial of Service *(DoS)*.

Chapter 7

Extending nodes' lifetime

If we lived in world where energy was infinite we would not have to worry about battery-life in our mobile devices. Unfortunately the reality is different, and today the trend in Smartphones is that energy is a serious constraint. In particular, the number and range of applications that require a lot of power is increasing rapidly. Companies which build these devices, are trying to find different solutions to save as much as possible energy. Sometimes they impose limitations in device in order to save energy. A real example was the first generation of iPhone [45]. Apple at that time built its device without 3G connectivity but just with EDGE connection. This is slower, but may use less energy. Indeed it was show in [46] by Balasubramanian et al. that 3G communication can use much more energy than EDGE or Wi-Fi. Even though the new iPhone has 3G, it was not until three years after the first iPhone. Apple continues to impose various models for power-saving, for example, by not deploying multitasking.

Researchers today are pursuing several different sub-areas related to power-saving. In [47], Tseng et al. list the main fields as: *Transmission Power Control*, *Power-Aware Routing* and *Low-Power Mode*.

In the first field, we classify papers that focus on avoiding collision or trying to minimise energy during transmission. In [48, 49] the authors concentrate on a procedure that allows a node to save energy while keeping the net connected. Even during transmission, nodes are able to increase or decrease transmission power in order to reduce energy expended.

Using the same approach of tuning the energy consumption during transmission, Wang in [50] nodes establish the right amount energy level based on a particular formula. This is computed during the route discovery process run by a sender. When a route request is sent along a path, the setup packet stores the relevant information. This is then used to compute the power needed to forward a packet hop-by-hop. This procedure is repeated by all nodes that forward the request. The path setup is reflected back by the receiver, carrying all the information gathered in the forward path before. From now on, the sender can choose the path where the total residual energy of nodes is maximum.

Routing is a second area researchers have studied. Kim and Koo in [44] provide a power-aware routing. Their protocol modifies the original link between nodes reducing the energy consumption. By allowing the use of more hops if necessary, a node can send a packet to a neighbour closer to the destination, reducing the energy required for forwarding. *PARO* and *ZRP* are the procedures that implement this approach.

Continuing to look at routing, Tao and Luo in [51] propose an extension to DSR, to make the protocol energy-aware. In particular, DSR is used to obtain all route available from a sender to a receipt. When these are established they introduce a function called *dynamic priority-weight* (dpw) that is used to compare, in terms of energy, all routes found. The one which requires least energy will be the candidate for end to end communication.

In [52] Wang et al. propose a different way to allow a node of saving energy. They make use of global synchronization proposed in [53] to coordinate beacon intervals from devices. Doing this with a pre-chosen hash function and the MAC-address of the receiver, a node is able to discover the listening period of a neighbour. In particular, the MAC-address is used as input for the hash function and when a sender is ready to send a packet it runs the hash function. After that, it sends the packet to its neighbour. Finally, this process is reiterated until the packet arrives to the final node.

Another solution that relies on using different node states is proposed in [47] by Tseng et al. In their protocol the authors implement three distinct protocols to be used by a node in order to keep a device in power-saving mode as long as possible but maintaining network connectivity.

These last two works can be classified into the last category, *Low-Power Mode.*

From the previous works it is clear that energy is a limited resource, and there is no way to allow a device to live forever, hence techniques to maximise battery life are relevant. In particular, forwarding messages is carried out by participants in the network and no separate routers are required. Mobility of nodes is very important, and this could impact the role taken in the network. A device in a crowded location may have its resources called upon frequently, if the routing protocol chooses it for long time.

Finding the shortest path to send a message is the job of the routing protocol. In MANETs, routes are often built on-demand in order to get the most reliable path at a given moment. However the path is selected is not always best with respect to the energy of the device along that path. A node can end up using its energy rapidly way just because it is frequently chosen as a forwarder.

We think that the latter point is a limitation of some MANET routing protocols. To overcome this lack we introduce an adjunct protocol, which is run alongside a routing protocol, makes the overall system energy-aware. We start by paying attention to nodes that are forwarding data for a long time. When this happens, our protocol allows a node to request to quit its role, to save energy. If it is determined that the right conditions are respected, then the device will suspend its activity. Hence, a node less likely to run out of energy.

The protocol was evaluated *NS2*. We modified one existing routing protocol, by adding our scheme to it, and ran simulations using both energy-aware and energy-unaware nodes. We obtained results allowing us to compare the gains in terms of energy savings for each node.

Regarding to the behaviour of a node throughout a simulation, we want to point out that a participant is not able to suspend its activity arbitrarily. According to this it cannot take a selfish behaviour, saving energy improperly, but will work normally following our protocol.

7.1 DSR and Shortest Path First

Before to propose our idea, we want to focus the attention in a particular feature of *DSR*. The latter one, as the majprity of routing protocols, adopts the choice of Shortest Path First *(SPF)* to select a path. Indeed, after a route discovering a node can collect different path to reach the destination. These are stored locally by the sender inside its own cache route table. When it must send some data to a node, it will select the path using *(SPF)*. Although many ways are available for the sender only one route is always selected, i.e. the shortest.

Some of the limitations of this approach are discussed in [54]. In particular, they show that using SPF to select a path can lead to problems like congestion. To overcome this problem they suggest a solution that uses an *emergency exit* to redirect data toward an alternative route.

Nowadays these weaknesses are still present and a new aspect must be considered within SPF. We said that mobile nodes of MANETs are limited by a low energy capability and that one of the biggest challenge is to save as much as possible energy. Since SPF is used in DSR it is clear to understand how the nodes' energy of a selected path can be wasted quickly.

Fig.7.1 shows an example of the node *"S"* that has performed a route discovery for the node *"D"*. By means of it multiple paths are found but just the shortest path is selected, i.e. {S - A - B - D}. The others will be ignored by DSR until the previous path is not more available.

In the following image *(fig.7.2)* the remaining energy of each node of the selected path is added.

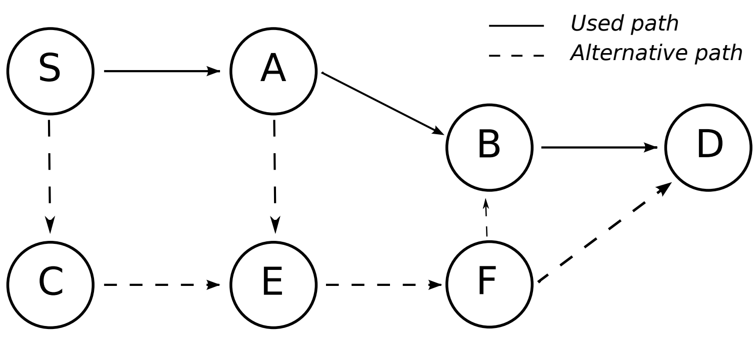

Figure 7.1: Network Topology

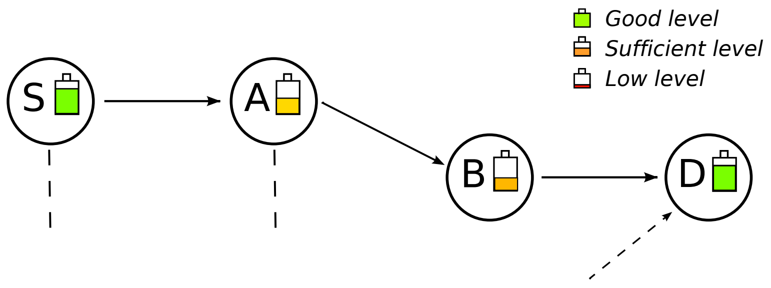

Figure 7.2: Network Topology with energy

DSR is a routing protocol and it does not consider the amount of energy of nodes. In the fig.7.2 the nodes that have to forward the data to the receipt have a moderate level of energy. If communication lasts for long time *"A"* and *"B"* will take the risk to finish their energy, in particular *"B"*. If this happens, the sender *"S"* will select another route to deliver the messages. From the point of view of the routing all system works perfectly since the communication $\{S \rightarrow D\}$ is still alive. But if we focus the attention to the energy and lifetime of each node we notice that it is unfair. The forwarder nodes have wasted their energy only for others aims. Now, *"B"* ran out completely its energy and it has not way to accomplish its own goals.

Another serious problem generated by this situation is the partitioning of the network. In this case, nodes utilized for long time can waste their energy rapidly making very important links not more available.

7.2 Dynamic Path Switching

We believe that the energy of a node must be preserved as long as possible, more so as the node reaches the end of its energy resources. In a MANET, after a route discovery, more routes are considered, but just the shortest is used. If we consider a situation in which one or more nodes try to save a part of their energy, this could be obtained changing a path dynamically when a pre-determined threshold is reached.

Let Ψ the set of all paths found by the sender *"S"* in order to deliver some data to the destination *"D"*. Let $\pi_{spf} \in \Psi$ the path selected by means of SPF and f_{spf} only the forwarding nodes of π_{spf}, i.e. nodes *"A"* and *"B"* in fig.7.2.

Our goal is to extend as much as possible the energy of f_{spf}. To do this we use other paths of Ψ during the same communication when determinate conditions appear. In fact, we believe that other nodes can be used to balance the energy consumption in the network. Hence, we propose a protocol, which works with the routing one, that change dynamically the path used to send the message to the receipt.

When a node, which is forwarding the message, sees that is consuming too much energy to forward the data, it would request to quit the forwarding state to save its remaining energy for its own goals. However when this happens, the node that would quit the path can do this only if others can take over its role. Otherwise its request will not be accepted. This crucial point must be taken into account because a node cannot decide arbitrarily to leave its role, otherwise, this could create a connectivity problems.

7.2.1 Managing energy warning situations

In the fig.7.3 it is shown the state of the forwarding nodes. After a period of data forwarding part of their energy has been used. Especially, the node *"B"* will risk in no longer time to not be able to send or forward messages anymore.

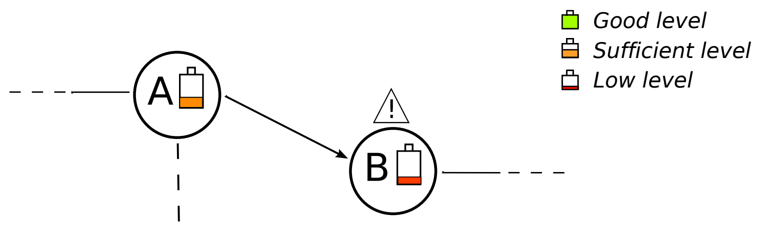

Figure 7.3: Node in a warning situation

In order to save the remaining energy, a node is able to require stopping its forwarding when one of the following two different situations occur during its job:

1. It has been working for long time.

2. It is finishing its energy.

In the first case we consider a situation where a node has given its contribute for long time. It has not been selfish but very collaborative and now it wants to save part of its own energy.

Let $E(j)$ the amount of energy consumed in forwarding, expressed as percentage, by the node j equal to:

$$E(j) = \frac{e_c}{e_s} * 100 \tag{7.1}$$

where:

- e_c: is the current value of energy.
- e_s: is the amount of energy available at the begin of the forwarding session.

In addition, we introduce λ as the energy consumption threshold and we compare it with the equation 7.1.

$$E(j) > \lambda \tag{7.2}$$

If the latter is true then a node can submit the request to stop its forwarding. To avoid that every node has the same λ, we assign different value of it to each node.

The second case consider the situation in which a node is forwarding some data and its energy is running out. This time we check the following situation:

$$e_c < \gamma \tag{7.3}$$

It means that if the remaining energy of a node, in percentage, is smaller than a threshold γ, it can ask to stop its forwarding.

7.2.2 How to perform the request

When a node forwards some data, it consumes a lot of energy due the receipt and the transmission of each message. To reduce this consumption as mush as possible a node can switch its state from active to sleeping if determinate situation are guaranteed.

Considering the state in fig.7.3 of the node B. Here, it recognizes that one of two condition is true and as consequence it will perform the request.

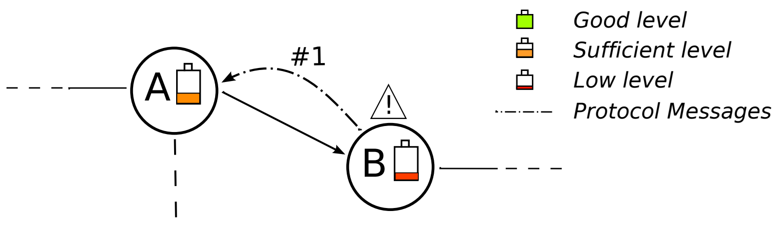

Figure 7.4: Protocol Message 1

In the fig.7.4 it is shown how the protocol starts. The first message sent from *"B"* to *"A"* is the follow:

1) B $\rightarrow$ A: *"Can I Sleep? :* x_d*"*

where:

- *Can I Sleep?*: is the request of the node.
- x_d: are the receipts of the communication in which the node that is doing the request is involved.[1]

[1]The value of d can be ≥ 1

At this step *"B"* does not know whether its request will be accepted because this depends on the feedback of *"A"*.

If the link {A - B} is unidirectional the node *"B"* can check in its route cache table an alternative way to reach *"A"*. If this is not found then *"B"* will start a route discovery for *"A"* in order to send the request to it.

In order to perform the previous request we implemented a method that allows a node to create a packet with particular data to a specified destination. The information, which are sent between devices, are about the state of a node. Indeed, they can complete that field using state messages, e.g. *"Can I Sleep?"*.

The method described above has been implemented in the following way:

```
void DSRAgent::sendMessageState(int dest, char strTmp[])
{
  RPPacket rrp;
  rrp.dest = net_id;
  rrp.src = net_id;
  rrp.pkt = allocpkt();
  hdr_rp *rph = hdr_rp::access(rrp.pkt);
  hdr_ip *iph = hdr_ip::access(rrp.pkt);
  hdr_cmn *cmnh = hdr_cmn::access(rrp.pkt);
  iph->daddr() = Address::instance().create_ipaddr(dest, RT_PORT);
  iph->dport() = RT_PORT;
  iph->saddr() = Address::instance().
                create_ipaddr(net_id.getNSAddr_t(), RT_PORT);
  iph->sport() = RT_PORT;
  cmnh->ptype() = PT_MESSAGE;
  cmnh->size() = size_ + IP_HDR_LEN; // add in IP header
  cmnh->num_forwards() = 0;
  cmnh->direction() = hdr_cmn::DOWN;
  PacketData* data = new PacketData(strlen (strTmp)+1);
  strncpy((char*) data->data(), strTmp, strlen (strTmp)+1);
  rrp.pkt->setdata(data);
  Scheduler::instance().schedule(ll, rrp.pkt, 1);
}
```

The packet created using the method of above is of a new type of message that we called *(PT_MESSAGE)*. The reason way we did that choice is to allow a node to filter easily this particular packet respect to others. So, when it recognised a packet of *(PT_MESSAGE)* type then a device handles it in a different way.

7.2.3 Managing the request

When a node receives the message *1*, it must check if alternative paths exist for each of x_d nodes. Even if the node *"A"* is not the sender of the message it can know the majority of the paths to the destination. This is due to the possibility to record routes from the route discovery of others, e.g. overhearing of messages.

When *"A"* receives the message, it checks whether the request of *"B"* can be accepted or not. In case of positive answer it will perform the second part of the protocol.

At this point two can be the answers proposed by *"A"* in the message of the fig.7.5:

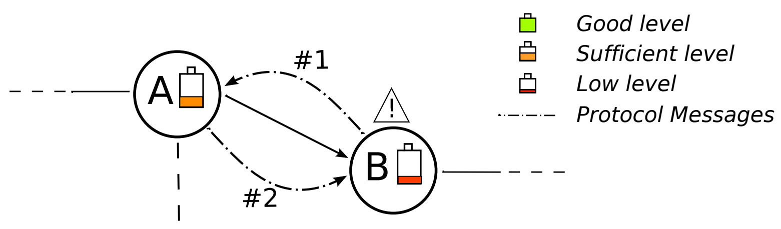

Figure 7.5: Protocol Message 2

2_I) A → B: *"Yes, You Can"*

or

2_{II}) A → B: *"No, You Cannot"*

In case of affirmative answer the node *"A"* will send the message 2_I according to the request of *"B"*. However, it must remove all link toward *"B"* from its cache route table and also, it should inform other nodes of the link that now is missing.

When that is done, the incoming messages, which before passed through *"B"*, are now redirect to another path. The latter could be different from the previous one or could skip just the node that now is sleeping using an alternative one.

Taking a look to the code implemented we considered the steps of above that a node must run before it goes to sleep.

```
if (strcmp(pktData,"YesYouCan")==0)
{
   ID from_id(Address::instance().get_nodeaddr(iph->saddr()),
         (ID_Type) Address::instance().get_nodeaddr(iph->saddr()));
```

```
    ID to_id(Address::instance().get_nodeaddr(iph->daddr()),
          (ID_Type) Address::instance().get_nodeaddr(iph->daddr()));
    route_cache->noticeDeadLink(to_id, from_id,
          Scheduler::instance().clock());
    flow_table.noticeDeadLink(to_id, from_id);
    xmitFailed(packet,"Error");
    em()->set_node_sleep(1);
    em()->setenergybeforesleeping(em()->energy());
    Packet::free(p.pkt);
    return;
}
```

On the other hand, eventually *"A"* does not find an alternative path, the message 2_{II} is sent. This means that the node cannot change its state because it is the only one that allows the message to reach the destination. Until the topology of the network will not undergo a change the node will not get a positive answer, e.g. a node could move next to *"B"* and after that it could take the place of *"B"*

7.2.4 Changing state or not

Depending on the answer of its neighbour the node could switch its state to sleeping. In particular, *"B"* could now turn off its wireless connection for a while and saving the remaining part of its energy.

After a period of sleeping, which it is determinate by a constant value, the node will be again on. Probably, it will not take part in the old communication but it could be either the sender of a new message or the forwarder of another communication.

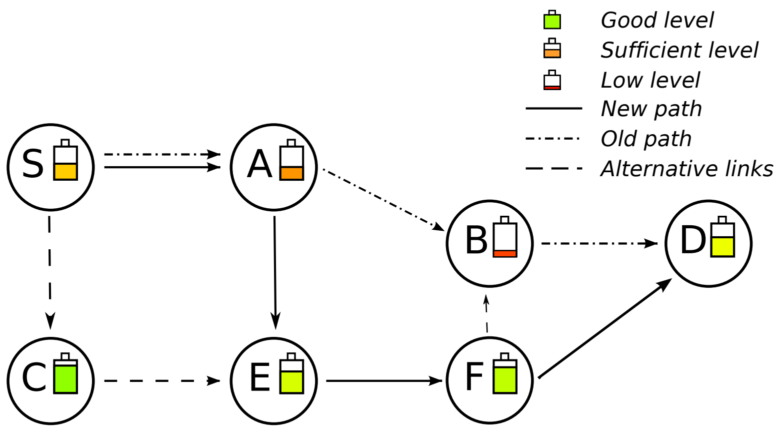

Figure 7.6: Network Topology with the new path

The fig.7.6 shows the new path followed by the sender, i.e. {S - A - E - F - D}. It is clear that this time the path is not the shortest, but it one hop long. In this case we overcome the problem of energy wasting using SPF. In addition, the new path selected can offer a better effort since it contains nodes with a high level of energy. However, the selection of the new path to use is entrusted to the routing protocol but this time *"B"* is excluded.

If this is the new situation, an other node could ask to change its state. In the fig.7.6 whether the communication is still on the node *"A"* could require to change its state.

7.3 Simulations

The protocol described above is suitable for reactive protocols, like *DSR* or *AODV*. For our purpose we tested it with *DSR* protocol. The protocol has been realised using the Network Simulator 2. In this case we used DSR and we implemented our protocol modifying part of it. In particular, we implemented the four important phases described in the previous section.

We focused all our simulations on energy consumption. Analyzing the behaviour of a node we verified how much energy it consumes during its job. This has been compared using the same simulation before when all nodes are *not Energy-Aware (non-EA)* and then with *Energy-Aware (EA)* nodes.

We used the energy model provided by NS2 setting as initial value of energy *200J* for each node. Also different energy consumption for sending and receiving a message. In particular, $1.5W$ of power used to send and $1.0W$ to receive a message. Others negligible quantity of energy are spent by a node when it switches its state. For example when it changes from active to sleep and vice-versa.

The scenario that we used for the simulation is shown in fig.7.7. We disposed 30 nodes with a particular layout. The communication range of each node is *200m* and according to this only one-hop neighbour can be reached.

When a node decide to send a message, it creates a route discovery for a destination node. After that the communication starts. The sender always delivers a CBR message to the receipt and this is done until the communication is not finished.

7.3.1 SPF vs DPS

To compare our protocol with the *SPF* used in *DSR* we realised several different tests. First of all we focus the attention in the different path selection

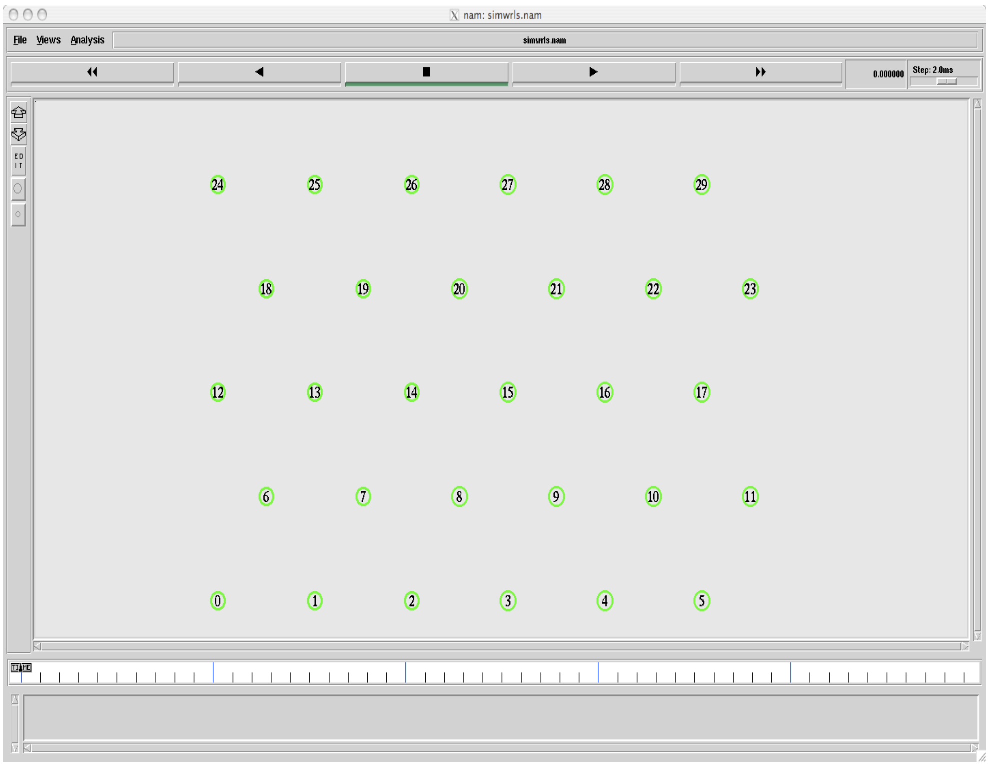

Figure 7.7: Network Topology

of two protocols when a communication is established. Then we show the energy consumption of nodes involved in forwarding. Finally we point out the trend of energy used when a node runs *Dynamic Path Switching* instead of *Shortest Path First*.

Path selection

As we said in the section (§7.1) a sender always uses the *SPF* to sends its messages. In our simulations we established a communication between the node 6 *(Sender)* and the node 21 *(Receipt)*. After the route discovery the sender selected the path shown in fig.7.8:

The simulation has been scheduled to send the message for 2000 seconds.

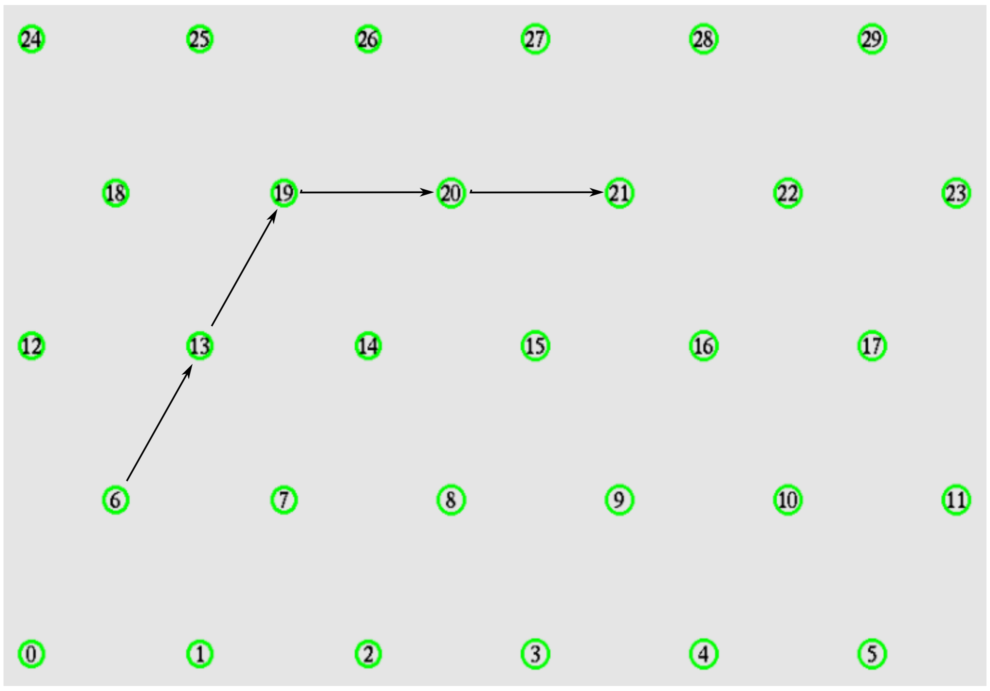

Figure 7.8: Path selected using Shortest Path First

During this period always the same forwarding nodes are selected, i.e. {13 -19 - 20}. They started with the maximum amount of energy but during their job this decreases rapidly. Indeed, the request to receive a packet and to send it to its neighbour requires a big effort for each f_i node. For this reason at the end of the communication their remaining energy is low. Instead other nodes, which did not take part in the forwarding, had a good amount because their energy is hardly used.

In this example it is possible to see the weakness of *DSR* using *SPF*. When the *Dynamic Path Switching* is used the result obtained is different. Considering the same model of transmission we show as other nodes are involved in the forwarding. The following images show all the paths changed during the same communication:

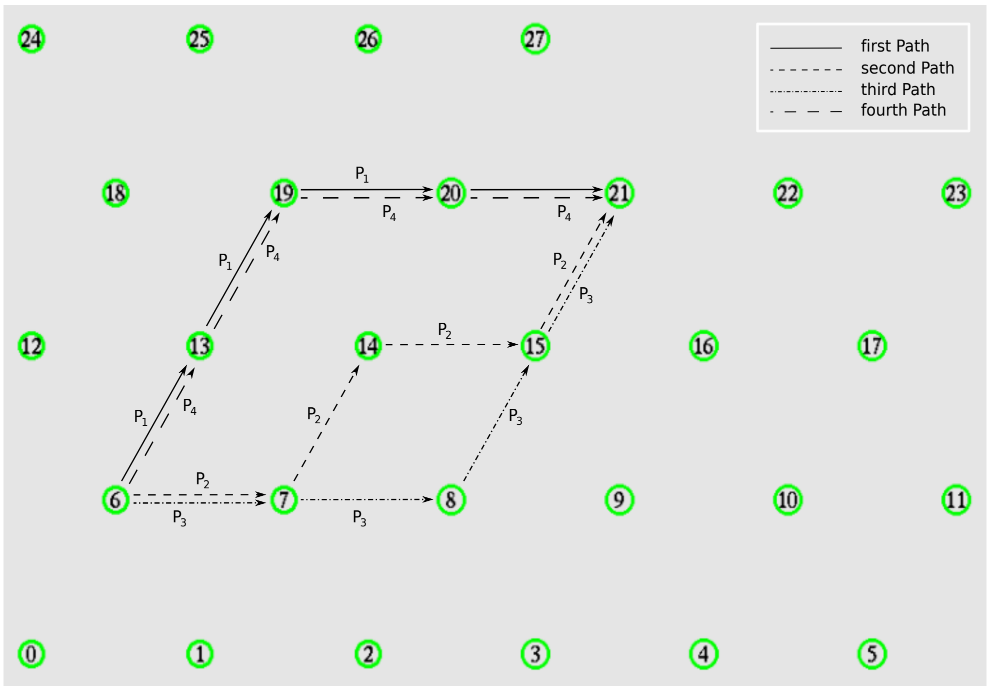

Figure 7.9: Paths selected using Dynamic Path Switching

When communication starts, the same *SPF* path is followed by the sender, i.e. P_1 in the fig.7.9. After some time forwarding, all f_i nodes will have spent some amount of energy. In the simulation the node 19, for first, verified that the expression 7.2 (§*7.2.1*) is true and according to this it ran the protocol DPS. Since an alternative way is found the sender selected another path, i.e. P_2. In particular, we noticed that the first transmission has lasted for around 850 seconds of simulation.

The new route followed did not contain any node that required to change its own state. In particular, the new path excluded also node 13 and 20. The communication represented in the second image had lasted for 440 seconds. After that the node 14 asked to change its state and the route P_3 is selected. In this case only it was excluded from the communication.

Finally, after 470 seconds the path changed. The last route P_4 will be used until the end of the transmission and it is the same path used in the first case.

Using *DPS* it is possible to observe that seven forwarding nodes were involved in the communication. Instead with the normal *SPF* only three. This explains how the *DPS* makes possible to spread the total energy consumption in more that *SPF's* nodes required.

Energy balancing

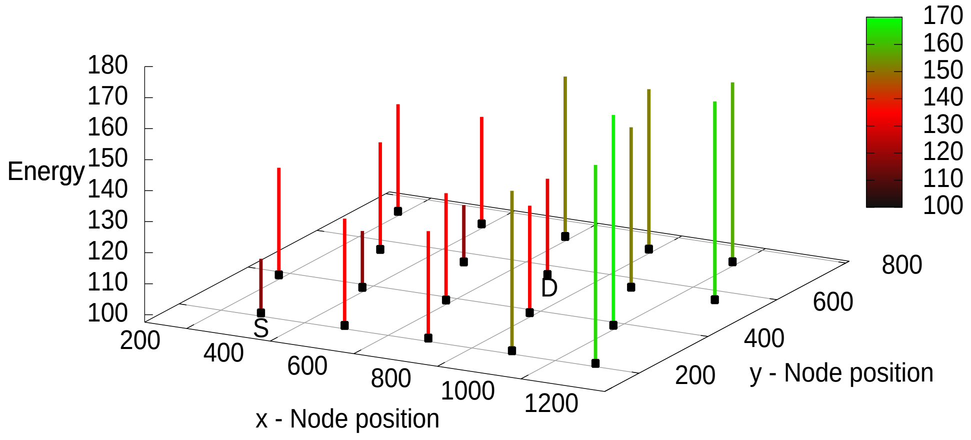

Figure 7.10: Energy Consumption using SPF

In the fig.7.10 it is shown the remaining energy of each node at the end of the communication. The path to reach the destination requires at least two forwarding nodes. With a label we marked the source and the destination node. In particular the latter consumed less energy of the sender because it receives only data. Instead, nodes selected using *SPF* wasted more energy of the others. This is represented by the brown lines. Nodes not involved in the communication have a good level of energy. In spite of this they used their own energy due to other reasons, such as overhearing.

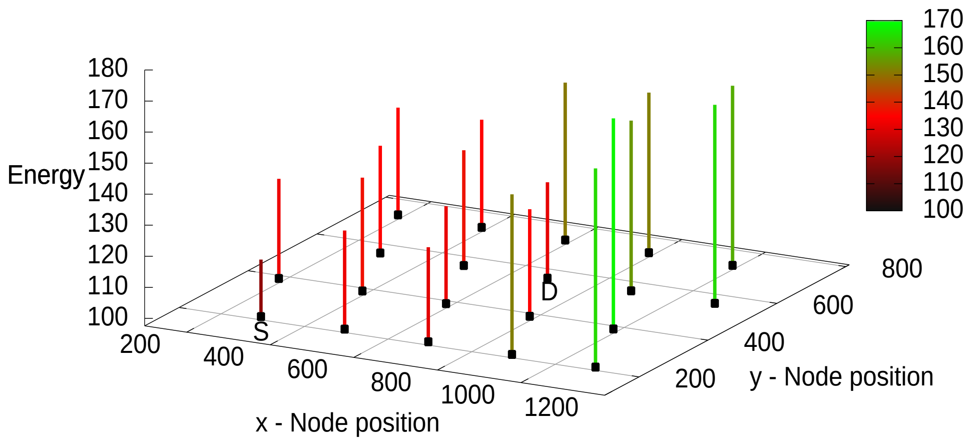

Figure 7.11: Energy Consumption using DPS

When *DPS* is ran we obtained the situation shown in fig.7.11. Except the sender and the receipt the remaining energy of the nodes is balanced in a different fashion. This is due to the balancing of the energy consumption. In the last case more nodes were required to forward the message and according to this the energy required has been spread on these ones.

Comparing energy consumption between EA and non-EA node

Throughout the simulation we observed the energy wasting of a node during its forwarding. We focused on it and measured its consumption every one hundred seconds of simulation. In this case the simulation lasted 1600 second. In the first time the node observed was not able to run the *DPS* protocol. Instead in the second time it ran the protocol. We obtained the graph shown in fig.7.12.

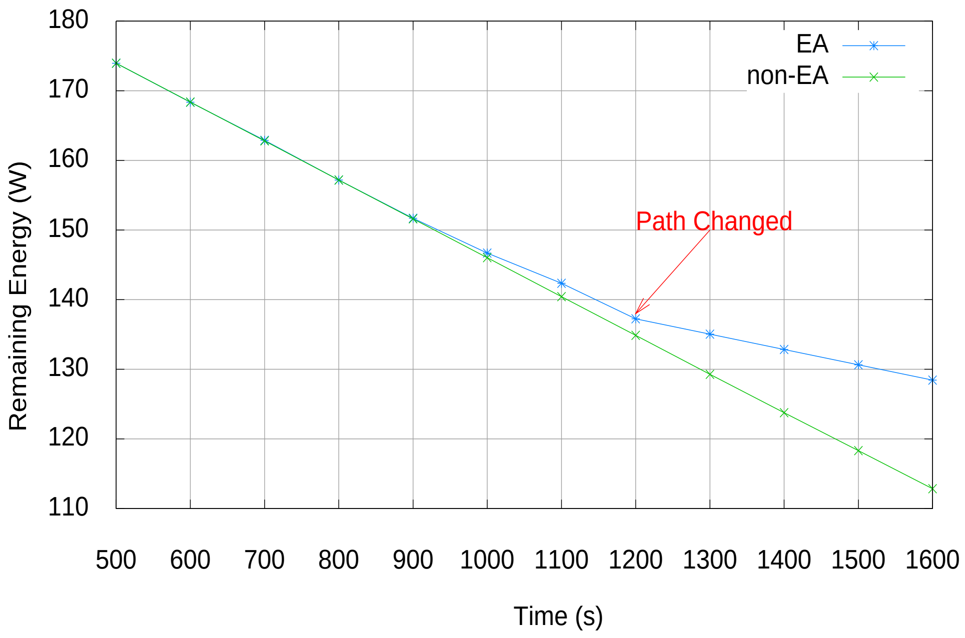

Figure 7.12: Energy Consumption of an EA and non-EA node

Both lines represent the energy used by a node that is forwarding some data to another one. In the green line the node, which has been selected, had to forward the message until the end of the communication. That is why in the network only *SPF* is used. In this case the energy consumption decrease in a constant way. The blue line shows the situation in which the observed node was able to run *DPS*. After a while of forwarding it required to change its state. The node asked for that and finally its request was accepted. Indeed, we can see that the line changed its trend. From that point the node is excluded by the communication. Also, the graph shows that the node is still consuming energy but this is due to other factors.

Chapter 8

Conclusion

Nowadays, the use of small portable devices, like smartphones, is becoming more and more widespread. They are rich of functionalities and people can use them in different ways. In particular, these devices are almost always connected to the Internet in order to exchange data. They look like laptops, in fact some features are similar although laptops typically are more powerful.

The main focus of this thesis was on mobile devices that are connected each other such as MANETs. Although the participating nodes have different interests, they must collaborate in order to carry out some distributed tasks. One of the most severe limitations of smartphones is the range of their antennas, which increases the need for collaboration in terms of packet forwarding.

We have observed that the behaviour of each participant is maybe arbitrary. Because the potential selfishness of a node is not known to the other *a priori*, the first goal of this thesis was exactly the development of the technique to signal selfishness. We have developed a metrics to signal a node's selfishness by means of its reputation. This is calculated exactly in terms of the node's contribution to the internal connection of the MANET. Of course, that contribution is treated with respect to the traffic originated in the node. The simulations showed as such a reputation technique carried out in NS2 confirm that it works effectively.

Another fundamental observation was that, despite the enforcement of a reputation mechanism, selfish nodes are not penalised. By contrast, they can live in the community and requires services as well as collaborative nodes. Obviously this does not seem fair and hence we set out to implement a mechanism of fairness for MANETs. We introduced a Quality of Services *(QoS)* technique throughout communications to penalise selfish devices. This must be interpreted as an extension to the reputation technique that we developed at first. The outcome is a combined reputation plus *QoS* technique to discourage selfish behaviour. Therefore, being collaborative becomes the behaviour to take in order to obtain a higher *QoS*.

However, it remains understood that the lifetime of a smartphone depends on its battery, and in consequence a collaborative behaviour cannot be kept for too long. An implication is that the network might excessively stress a node, which aiming at a high *QoS*, would run a serious risk of depleting its battery prematurely. Intuitively, if on one hand we require each node to behave fairly with the network, we must also make sure that the network is fair with each node. This is particularly realist due to the fact that the *DSR* protocol normally chooses a shortest path strategy, and would consequently always invoke the same set of nodes in case the network topology only varies negligibly. Our reputation plus *QoS* technique was then extended once more, the role of forwarders gets moved to different nodes. The resulting idea produces an intelligent routing protocol that dynamically chooses other nodes when a collaborative one raises a flag of over-participation. In conclusion the total consumed by the all network is distributed among a bigger set of devices than with the original *DSR*.

Reputation, Collaboration and Power-Saving have been the main topics of this Ph.D. thesis. They tackle the most important features and weaknesses of Mobile Ad hoc Networks, and our innovations in each of the three fields contribute to a, generally speaking, "better network". We remark that all finding were simulated to the Network Simulator 2, which made an objective evaluation of the improvements possible. Different ideas had to be conjectured, techniques tried, simulations rerun. However the general trend of the findings always was of steady progression. Moreover, it was possible to write various research papers during these three years. Most of them were accepted at different international conferences and journals.

Bibliography

[1] The official bluetooth technology info site - http://www.bluetooth.com/ [online] (June 2010).

[2] Defition of smartphone [online] (March 2010).

[3] Apple inc. - http://www.apple.com/ [online] (March 2010).

[4] G. Pei, M. Gerla, T.-W. Chen, Fisheye state routing: a routing scheme for ad hoc wireless networks, Communications, 2000. ICC 2000. 2000 IEEE International Conference on 1 (2000) pp. 70 – 74.

[5] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, L. Viennot, Optimized link state routing protocol for ad hoc networks, Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century. Proceedings. IEEE International (2002) pp. 62 – 68.

[6] R. G. Ogier, F. L. Templin, Topology broadcast based on reverse-path forwarding, IETF MANET Working Group Draft (2002) pp. 1–21.

[7] Q. Amir, V. Laurent, L. Anis, Multipoint relaying : an efficient technique for flooding in mobile wireless networks, Military sciences and Mis-

sile technology and Navigation, communications, detection and countermeasures and Ordnance.

[8] D. Johnson, Y. Hu, D. Maltz, The dynamic source routing protocol (DSR) for mobile ad hoc networks for ipv4, Internet RFC 4728 (2007) 107.

[9] C. R. Perkins, E. M. Belding-Royer, S. R. Das, Ad hoc on-demand distance vector (AODV) routing, Internet RFC 3561 (2003) 37.

[10] G. Pei, M. Gerla, X. Hong, C.-C. Chiang, A wireless hierarchical routing protocol with group mobility (1) (1999) pp. 1538–1542.

[11] G. Pei, M. Gerla, X. Hong, Landmark ad-hoc routing protocol, MobiHoc '00 (2000) pp. 11 – 18.

[12] S. Basagni, I. Chlamtac, V. R. Syrotiuk, B. A. Woodward, A distance routing effect algorithm for mobility (dream) (1998) pp. 76 – 84.

[13] J. C. Navas, T. Imielinski, Geocast—geographic addressing and routing (1997) pp. 66 – 76.

[14] Y.-B. Ko, N. H. Vaidya, Location-aided routing (lar) in mobile ad hoc networks, MobiCom '98: Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking (1998) pp. 66 – 75.

[15] B. Karp, H. T. Kung, Gpsr: greedy perimeter stateless routing for wireless networks, MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking (2000) pp. 243 –254.

[16] Napster - http://en.wikipedia.org/wiki/napster [online] (June 2010).

[17] Mpeg-3 - http://en.wikipedia.org/wiki/mp3 [online] (June 2010).

[18] J. Liang, R. Kumar, K. Ross, Understanding kazaa.

[19] M. Ripeanu, Peer-to-peer architecture case study: Gnutella network, Peer-to-Peer Computing, 2001. Proceedings. First International Conference on (2001) 99–100.

[20] Chris, T. Hargreaves, M. Kern, Fasttrack p2p network.
URL `http://gift-fasttrack.berlios.de/`

[21] I. Clarke, M. Toseland, O. Sandberg, F. Daignière, S. Miller, S. Starr, M. Rogers, D. Baker, R. Hailey, D. Sowder, Freenet: A distributed anonymous information storage and retrieval system, Lecture Notes in Computer Science 2009/2001 (ISSN: 0302-9743 (Print) 1611-3349 (Online)) (2001) pp. 46 – 66.
URL `http://freenetproject.org/`

[22] Skype - http://www.skype.com [online] (June 2010).

[23] An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. `doi:10.1109/INFOCOM.2006.312`.
URL `http://dx.doi.org/10.1109/INFOCOM.2006.312`

[24] B. Cohen, Incentives build robustness in bittorrent (May 2003).

[25] Y. Kulbak, D. Bickson, The eMule protocol specification (January 2005).

[26] R. G. Cole, J. H. Rosenbluth, Voice over ip performance monitoring, SIGCOMM Comput. Commun. Rev. 31 (2) (2001) pp. 9 – 24.

[27] Advanced encryption standard (aes), National Institute of Standards and Technology (NIST) - Federal Information Processing Standard Publication 197.

[28] Free software foundation - http://www.gnu.org/licenses/gpl.html [online] (October 2010).

[29] Real network simulator - http://www.cs.cornell.edu/skeshav/real/overview.html [online] (October 2010).

[30] Defense advanced research projects agency - http://www.darpa.mil/ [online] (October 2010).

[31] J. Liu, V. Issarny, Enhanced reputation mechanism for mobile ad hoc networks, in: iTrust, Vol. In Proceedings of 2nd International Conference on Trust Management, 2004, pp. 48–62.

[32] A. Jøsang, R. Ismail, C. Boyd, A survey of trust and reputation systems for online service provision, Decision Support Systems 43 (2) (2007) pp. 618 – 644.

[33] S. Marsh, Formalising trust as a computational concept, PhD thesis, University of Stirling, UK (1994).

[34] S. Bistarelli, F. Santini, Propagating multitrust within trust networks, Proceedings of the 23th Annual ACM Symposium on Applied Computing (2008) 5.

[35] S. Buchegger, J.-Y. L. Boudec, Performance analysis of the CONFIDANT protocol: Cooperation of nodes — fairness in dynamic ad-hoc networks.

[36] P. Michiardi, R. Molva, Core: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks, Proceedings of the IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security (2002) pp. 107–121.

[37] E. Papalilo, B. Freisleben, Managing behaviour trust in grid computing environments, Journal of Information Assurance and Security (JIAS) 3 (1) (2008) pp. 27–37.

[38] G. Bella, S. Bistarelli, F. Massacci, Retaliation: Can we live with flaws?, Proceedings of the Nato Advanced Research Workshop on Information Security Assurance and Security.

[39] S. Mccanne, S. Floyd, K. Fall. The network simulator 2 [online] (June 2009).

[40] S. Kurkowski, T. Camp, M. Colagrosso, Manet simulation studies: the incredibles, ACM SIGMOBILE Mobile Computing and Communications Review 9 (4) (2005) pp. 50 – 61.

[41] V. Park, S. Corson, I. Flarion Technologies, Temporally-ordered routing algorithm (tora), Internet draft 04 (2001) pp. 23.

[42] S. G. Glisic, Advanced Wireless Networks: 4G Technologies, John Wiley & Sons, 2006.

[43] H. Yan, D. Lowenthal, Towards cooperation fairness in mobile ad hoc networks, Wireless Communications and Networking Conference, 2005 IEEE 4 (2005) pp. 2143–2148.

[44] K.-J. Kim, H.-W. Koo, Optimizing power-aware routing using zone routing protocol in manet, Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on (2007) pp. 670 – 675.

[45] Apple inc. - http://www.apple.com/iphone/ [online] (March 2010).

[46] N. Balasubramanian, A. Balasubramanian, A. Venkataramani, Energy consumption in mobile phones: a measurement study and implications for network applications, in: IMC '09: Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, ACM, New York, NY, USA, 2009, pp. 280–293. `doi:http://doi.acm.org/10.1145/1644893.1644927`.

[47] Y.-C. Tseng, C.-S. Hsu, T.-Y. Hsieh, Power-saving protocols for ieee 802.11-based multi-hop ad hoc networks, Computer Networks: The International Journal of Computer and Telecommunications Networking 43 (3) (2003) 317–337. `doi:http://dx.doi.org/10.1016/S1389-1286(03)00284-6`.

[48] R. Ramanathan, R. Rosales-Hain, Topology control of multihop wireless networks using transmit power adjustment, INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE 2 (2002) pp. 404 – 410.

[49] R. Wattenhofer, L. Li, P. Bahl, Y. min Wang, Distributed topology control for power efficient operation in multihop wireless ad hoc networks, IEEE INFOCOM (2001) pp. 1388 – 1397.

[50] Y. Wang, Study on energy conservation in manet, Journal of Networks Vol. 5 (No. 6) (No. 6) pp. 708 – 715.

[51] Y. Tao, W. Luo, Modified energy-aware dsr routing for ad hoc network, in: Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on, 2007, pp. pp. 1601 – 1603. `doi:10.1109/WICOM.2007.403`.

[52] C.-Y. Wang, C.-J. Wu, G. N. Chen, p-manet: Efficient power saving protocol for multi-hop mobile ad hoc networks, in: ICITA '05: Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05) Volume 2, IEEE Computer Society, Washington, DC, USA, 2005, pp. pp. 271 – 276.

[53] J.-P. Sheu, C.-M. Chao, C.-W. Sun (Eds.), A clock synchronization algorithm for multi-hop wireless ad hoc networks, Vol. Clock synchronization - IEEE 802.11 - multihop wireless ad hoc networks, Springer Netherlands, 2007.

[54] Z. Wang, J. Crowcroft, Shortest path first with emergency exits, in: SIGCOMM '90: Proceedings of the ACM symposium on Communications architectures & protocols, ACM, New York, NY, USA, 1990, pp. 166–176. `doi:http://doi.acm.org/10.1145/99508.99548`.