UNIVERSIDADE NOVA DE LISBOA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

INCLUDING CONTEXT IN A ROUTING ALGORITHM

FOR THE INTERNET OF THINGS

POR:

VÍTOR BRUNO HORTA CARVALHO

DISSERTAÇÃO APRESENTADA NA FACULDADE DE CIÊNCIAS E TECNOLOGIA DA UNIVERSIDADE NOVA DE LISBOA PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA INFORMÁTICA

ORIENTADOR: PROF. DOUTOR PEDRO ABÍLIO DUARTE DE MEDEIROS

LISBOA

2010

AGRADECIMENTOS / ACKNOWLEDGEMENTS

Sir Isaac Newton, em 1676 escreveu o seguinte: "*If I have seen further it is only by standing on the shoulders of giants.*". Desta forma inicio assim os agradecimentos, a todos vocês que me permitiram estar aqui neste momento, um muito obrigado, vocês são os meus gigantes.

Em primeiro lugar um agradecimento a toda a minha família que independentemente de tudo, sempre acreditou em mim e deu-me a possibilidade e a força para continuar a estudar. Em especial à minha mãe por toda a força que sempre teve e ao meu pai que embora não me possa ver neste momento, sempre lutou pela minha felicidade, a ambos que sempre me guiaram pelo caminho do amor e da verdade, este trabalho também é vosso.

Aos amigos! Aos amigos de A a Z, de Norte a Sul, mas sempre intemporais e que realmente sentem a mensagem! Às noitadas de estudos (e nem sempre), ao famoso semicírculo, à alegria, às leituras dramáticas nocturnas, às noites na praia, às pausas, e ao companheirismo que sempre demonstraram, um muito obrigado! Mas também… à minha equipa de Taekwondo por me mostrar o verdadeiro espírito de equipa e claro, ao David Nunes por toda a sua mestria e amizade: obrigado a vós!

Um muito obrigado ao Mestre Pedro Maló por todo o conhecimento transmitido e por me ter aberto as portas do mundo da investigação dentro do GRIS e claro, sem esquecer, aos Putos!

Por fim, agradeço ao Prof. Pedro Medeiros toda a dedicação e orientação que permitiu a este trabalho ter sido realizado, contribuindo não só para a minha vida académica, mas também, pessoal.

SUMÁRIO

SUMÁRIO

A chamada “Internet of Things” assume que um grande número de dispositivos usados no dia a dia venham a estar ligados à Internet. Este cenário abrirá espaço a um conjunto alargado de novas aplicações, mas a ligação à rede de um conjunto enorme de nós, que podem ser ligados e desligados, mover-se e que têm limitações quanto às capacidades de processamento e comunicação, levanta a necessidade de serem colocados no terreno algoritmos de encaminhamento de mensagens diferentes daqueles que são usados hoje em dia. Nesta tese, procura-se contribuir para o desenvolvimento desse tipo de algoritmos.

Em particular, é ensaiada a ideia de os algoritmos de encaminhamento poderem aumentar o seu desempenho a vários níveis – tempo de entrega de mensagens, número de mensagens perdidas, consumo de energia, etc. – se nas decisões sobre o encaminhamento das mensagens puder ser utilizado o *Contexto*. No âmbito desta tese, o *Contexto* corresponde a uma colecção de informação organizada que o algoritmo de encaminhamento recolhe do ambiente que rodeia os nós que fazem parte da rede, e que lhe permite tomar decisões de encaminhamento mais correctas. A informação usada pode estar relacionada com questões de baixo nível – localização do nó, potência necessária para enviar uma mensagem, etc – como com constrangimentos associados à aplicação – prioridade de uma mensagem, tempo máximo para entregar, etc.

Com o objectivo de avaliar esta abordagem, nesta tese propõe-se um algoritmo de encaminhamento de mensagens chamado C-AODV. Como o nome sugere, este algoritmo baseia-se no algoritmo AODV, modificando-o em vários aspectos; em particular é introduzida a possibilidade de usar a informação extraída do contexto para efectuar um melhor encaminhamento das mensagens.

Para avaliar a capacidade da solução proposta, foi utilizado o simulador NS-3 e feitos diversos testes que permitiram verificar a funcionalidade do algoritmo e que indiciam que a proposta feita é válida.

ABSTRACT

The “Internet of Things” assumes that a large number of devices which are used on a daily basis will eventually become connected to the Internet. This scenario will provide room for a large set of new applications, however the network connections of an enormous set of nodes, which can be connected and disconnected, can move around and which have limitations with regards to their processing and communication capabilities, raises the need for the development of new message routing algorithms, different from those being in use today. In this thesis, a contribution is made towards the development of this type of algorithms.

In particular, the idea which is tested is whether routing algorithms can improve their performance at various levels, such as, message delivery time, number of messages lost, power consumption, etc., if in the routing decisions these algorithms can make use of the concept of “Context”. Within the framework of this thesis, the “Context” is the organized collection of information which the routing algorithm collects from the environment surrounding the network nodes, and which allows it to make better routing decisions. This information can be related to low-level issues, such as, node location, power required to send a message, etc., as well as, with constraints related to the application, such as, message priority, maximum delivery time, etc.

In order to evaluate this approach, this thesis proposes a routing algorithm called C-AODV. As the name suggests, it is based on the ADOV algorithm, however it is modified in several aspects; in particular, the possibility of using information collected from the context can be utilized to improve message routing.

In order to test the proposed solution, several tests were performed on the NS-3 simulator which allowed the evaluation of the algorithm functionalities. The tests performed indicate that the proposed solution is valid.

To Bianca

SYMBOLOGY AND NOTATIONS

IoT	Internet of Things
RWI	Real World Internet
MANET	Mobile Ad hoc Networks
LAN	Local Area Network
RTD	Research, Technology and Development
TCP/IP	Transmission Control Protocol / Internet Protocol
RFID	Radio Frequency Identification
W3C	World Wide Web Consortium
RDF	Resource Description Framework
RDFS	RDF Schema
OWL	Web Ontology Language
XML	Extensible Markup Language
IPv4	Internet Protocol version 4
QoS	Quality of Service
ISO	International Organization for Standardization
DHT	Distributed Hash Table
ACM	Association for Computing Machinery

WWW World Wide Web

URL Uniform Resource Locator

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. GENERAL INTRODUCTION AND MOTIVATION

In the vision of the Commission of the European Communities (European Parliament, 2008), the Internet of Things (IoT) is one of the most promising explorations of the next generation of the Internet as we know it. In this vision, objects ("*things*") can take a part in the Internet by exchanging, gathering, storing or processing information, leading to new forms of services and new business opportunities. Citizens, society and environment will all benefit from it (Table 1.1).

According to (European Commission, 2010) the IoT is a "*dynamic global network infrastructure with self capabilities based on a standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network*", the Internet.

TABLE 1.1 IOT APPLICATION DOMAINS - DESCRIPTION AND EXAMPLES

Domain	Description	Indicative examples
Industry	Activities involving financial or commercial transactions between companies, organisations and other entities	Manufacturing, logistics, service sector, banking, financial governmental authorities, intermediaries, etc.
Environment	Activities regarding the protection, monitoring and development of all natural resources	Agriculture & breeding, recycling, environmental management services, energy management, etc.
Society	Activities/ initiatives regarding the development and inclusion of societies, cities, and people	Governmental services towards citizens and other society structures (e-participation), e-inclusion (e.g. aging, disabled people), etc.

This idea was even awarded Time Magazine's #30 Best Invention of 2008: "[…] *intends to create a new kind of network that will allow sensor-enabled physical objects — appliances in your home, products in a factory, cars in a city — to talk to one another, the same way people communicate over the Internet*" (30. The Internet Of Things - TIME's Best Inventions of 2008 - TIME, 2010).

According to (European Commission, 2010), show that today, there are almost 1.5 billion PCs and over 1 billion cell phones connected to Internet. These devices will move towards the IoT in which 50 to 100 billion devices will be connected to the Internet by 2020. Some projections indicate that in the same year, the number of mobile machine sessions will be 30 times higher than the number of mobile person sessions. Not considering only machine-to-machine communications but also communications among all kinds of objects, then the number of networked object will be orders of magnitude above those of today.

Most of the nodes of this network will have the following characteristics:

- **Limited power capability**;
- **Wireless receivers and transmitters** with limited range, facing the use of multi-hop communication;
- **Mobility**: nodes will move, possibly becoming disconnected;
- **Volability**: nodes can be switched on and off often.

These network objects will be organized in networks, these networks will be very different regarding criteria like the number of nodes, topology, organization, etc.

More information about network topologies can be found in ANNEX A.

1.2. PROBLEM DESCRIPTION AND CONTEXT

In the context described above, networks may look similar to what we call now Mobile Ad hoc Network (MANET), due to its ad hoc nature and the possibility of node's joining and leaving the network at any time.

Relatively to these kinds of networks, some characteristics must be considered, like dynamism, rapid-changing patterns, random and multi-hop (Corson & Macker, 1999). The

nodes are equipped with some kind of communication receiver and transmitter, and depending on the transmission power or channel frequency, a multi-hop network can exist between them.

In order to support the communication between all the nodes, the communication mechanism must tackle the following aspects:

Issue 1. **Partial or local notion of network information**: the solution must work efficiently despite the partial knowledge of its surroundings and constant changes in the network. The definition of a route must take care of heuristics that could help the system decide the best choice in the current situation. This restriction is imposed because devices may have limitations, like storage, power or processing power, not allowing the devices to have a notion of the entire network due to the number of participants;

Issue 2. **Different routing decisions based on multiple routes**: the solution must make different routing decisions based on the possibility of different kinds of networks with different properties. Those decisions must be aware of the paths that messages may travel and choose the more useful route to destination;

Issue 3. **Zero management network with self-configuring behaviour**: when a device joins or leaves the network, it must be aware of its neighbours and their properties. Also, the neighbours must be aware of new devices and changes in the network, due to failures, devices leaving the network or mobility. With this approach, participants may configure themselves in order to adapt to changes and new parameters;

Issue 4. **Information exchange within the whole system**: the solution must acquire information from all parts of the system and also allow the system to acquire information from it. This exchange of information between layers works in a loop which can improve the efficiency of the device and in this case, routing decisions. The bottom layers (network, logical and physical) can use top layer information, like traffic information, in order to change routes and the top layers can use routing information in order to have a more accurate notion of the network participants.

With this property, devices may acquire more information about themselves and the network at different levels, adapted to the current situation.

The proposed work is to implement a routing algorithm in a context of the IoT. This is mandatory due to the possibility of the nodes join and leave network leading it to an incoherent state of routing and so, an inefficient network. This network can be used on small and micro enterprises in many industries fields. These industries fields will be able to improve its logistical tracking and tracing, production, monitoring, maintenance, product safety, quality and information; as example, food and construction industries can take advantages of this work.

Briefly, the research question of this work is: **how to conceive a routing algorithm that works and operates efficiently in a network environment that changes arbitrarily?**

1.3. PROPOSED SOLUTION AND WORK SCOPE

The base line solution of this work is acknowledging the existence of two major classes of routing algorithms: adaptive and nonadaptive algorithms (Tanenbaum, 1996). Nonadaptive algorithms do not make routing decisions based in traffic or topology, this is also called static routing; in contrast, adaptive algorithms make their decisions answering the changes in network topology, traffic analysis or some after periodic evaluation. Another facet of routing is the metric for routing that can be based on shortest-path routing, network usage optimization and policy routing.

It is clear that this thesis will be based in an adaptive algorithm because nodes can join and leave the network, changing the topology and traffic flow. Since in the IoT the nodes are mobile, a mix between shortest-path routing and network usage optimization will be explored. This class of routing algorithms can also be divided in two other types of routing: the *global/proactive routing* and the reactive/on-demand routing algorithms (Abolhasan,

2003). In proactive routing all the routes being calculated at start-up and maintained in a periodical process; in reactive routing when a node wants to communicate with another, the route is calculated at that moment; another type of routing, *hybrid routing*, which consists of a mix of the two already mentioned.

Due to concern of saving nodes' capabilities, like power or storage, the reactive routing protocols were chosen for this work, however a more complete research and analysis between proactive, reactive and hybrid routing algorithms can be found in ANNEX B.

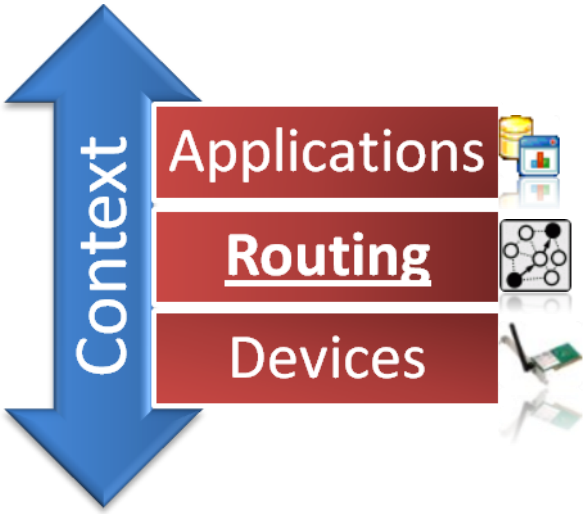


FIGURE 1.1 WORK SCOPE

In order to bring advantages and innovations when compared to other solutions, the proposed work aims to use the notion of Context to improve the efficiency of the routing algorithm, by giving it some knowledge about its surroundings. The Context is the information that can be used to characterize a situation of a person, object or the surroundings. This knowledge can be retrieved in many ways, in particular, it may be retrieved from the upper layers, like the application layer, or even the messages that pass through the network, but it can also be retrieved from the lower layers, like physical information from the device where it operates, power information or processing capabilities. A graphical vision of this approach is presented in Figure 1.1.

1.4. MAIN CONTRIBUTIONS EXPECTED

The main contribution expected is the design, implementation and preliminary tests of a routing algorithm that aims at being scalable, efficient and fault tolerant IoT.

This study will also make a contribution to the project FP7-216420 CuteLoop (CuteLoop - Customer in the Loop, 2010) entitled "*Customer in the Loop: Using Networked Devices enabled Intelligence for Proactive Customers Integration as Drivers of Integrated Enterprise*", partially funded by European Commission, that meant to explore the interaction between enterprise actors and devices, to realize distributed and autonomous control of business processes; but also in a contribution for an emerging field of studying, the IoT where common objects are embedded in the environment work together and in synergy to accomplish added-value goals that improve business performance and the overall quality of life.

This work can also be used to achieve more knowledge for the RWI Network Architecture (ANNEX C) around the Communication and Routing problem, facing the driver “Maximum connectivity with minimum consumption” in the scope of (FIA, 2009).

1.5. RESEARCH METHODOLOGY

The research methodology of this work is based on the scientific method, composed by the following steps (Schafersman, 1994):

1. Definition of Research Question;
2. Information Gathering and requirements;
3. Hypothesis formulation;
4. Experience perform;
5. Analysis of results and conclusions;
6. Publications.

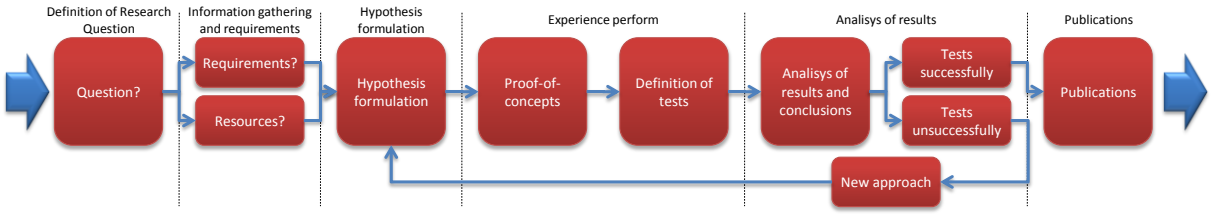


FIGURE 1.2 RESEARCH METHODOLOGY

1. DEFINITION OF RESEARCH QUESTION

Every research work begins with the definition of a problem, more specifically with the Research Question. The Research Question of this work, as told before is "*how to conceive a routing algorithm that works and operates efficiently in a network environment that changes arbitrarily?*" and tries to give answers about how it is possible to develop a method to find resources in a network composed by many devices with energy, storing, processing concerns, among others.

2. INFORMATION GATHERING AND REQUIREMENTS

The requirements for a research work must be identified by a junction of the needs of both industry and research challenges. In the vision of industry, it is necessary to evaluate the applicability of the solution in the real world. The major application of this work is in small and micro enterprises, namely in the food and construction industry. Therefore, it is necessary to gather information about the requirements of these industries and also gather scientific and technological information about the existing technical information, in this case, routing algorithms. In this document, this step can be found in the **Related work** chapter, where all the information was gathered and used as base to this work;

3. HYPOTHESIS FORMULATION

Based on the information gathered and in the knowledge of the requirements, the formulation of an hypothesis is mandatory. In this phase, the definition of this conceptual realization that will be the base for the analysis of the problem and the base for an implementation of an experience in order to evaluate the hypothesis proposed. In the chapter **Proposed algorithm,** the hypothesis for this dissertation is presented. This chapter is all dedicated to a technological explanation of the proposed solution in order to understand all the decisions made and enable the implementation phase.

4. EXPERIENCE PERFORM

This phase aims to evaluate the hypothesis, by the experimental implementation of the proposed solution as a proof-of-concept in order to gather information, like metrics, behaviours or other kinds of results to evaluate them facing the hypothesis. All the results must be taken in a controlled environment in order to minimize the samples' noise but also, to control all the results of the experiment. This step can be found in this document in chapter **C-AODV proof-of-concept**. It presents the implementation details of this thesis, the decisions made, the technologies used, in order to perform the experiment;

5. ANALYSIS OF RESULTS AND CONCLUSIONS

Facing the results obtained in the proof-of-concepts against the hypothesis, it is possible to verify if this solution meets the requirements defined. The chapter **Tests and Validation** presents the tests and the results obtained after the implementation. These results may be addressed against the goals proposed on chapter 1.2. Finally, the chapter **Conclusions and future work** presents the final conclusions of this dissertation. A critic evaluation of the results obtained is made and a final conclusion for the overall work is performed.

6. PUBLICATIONS

The final phase of the scientific method is the publication and the exchange of the information gathered on the experiment. This exchange is critical not only for avoiding the redundancy of the tests for the proposed hypothesis, but also for the evolution of knowledge and form new formulations of hypothesis based on the work done. Although this work did not originate any publication, it is planned to do it, or in order to raise new questions within the scientific community and enabling new discussions and future works based on the conclusions taken for this work.

1.6. DOCUMENT STRUCTURE

This document is composed by the following main chapters:

- **Related work**: This chapter presents the state-of-the-art in the areas of reactive routing protocols and network simulators;

- **Proposed algorithm**: Where the proposed routing algorithm (C-AODV) is defined;
- **C-AODV proof-of-concept**: In this chapter an implementation of C-AODV is described;
- **Tests and Validation**: In this chapter, an assessment of C-AODV is performed;
- **Conclusions and future work**: Here the final conclusions of the work are outlined and future work is discussed.

2. RELATED WORK

This chapter presents the state-of-research analysis to the proposed work, made by performing a comprehensive literature review, mainly composed of reference papers in selected journals and conferences, consulted in the Web of Science or in b-on libraries, in reference books in the work domain, but also in the knowledge and experience acquired with the involvement in the CuteLoop project (CuteLoop - Customer in the Loop, 2010).

The study is performed in the environment described in chapter 1.2 that is summarized here:

- Nodes have heterogeneous characteristics, but most of them have low capabilities either in computation capabilities and communication range. Also, nodes have power limitations;
- Nodes do not have a full vision of the entire network. This happens because nodes can be switched on/off and can move; in this last case, nodes can communicate with the node that changed;
- The above node characteristics imply that when communicating between two nodes a multi-hop route must be used. This route can change over time. In a given moment, the route used can be influenced by parameters not relevant in traditional networks, like the power consumed in the message transmitted over the hops.

2.1. REACTIVE ROUTING PROTOCOLS

These protocols calculate their routes only when required by the source, through a process which is usually called *on-demand* route discovery. Due to the possible changes in the network, a minimum of static information related to the network must be stored. This class of algorithms can handle this problem.

2.1.1. DYNAMIC SOURCE ROUTING

This analysis is based on the Dynamic Source Routing (DSR) proposal presented in (Johnson, Maltz, & Hu, 2004).

This protocol is targeted for multi-hop ad-hoc networks that may have mobility problems. It is composed by two mechanisms, “Route Discovery” and “Route Maintenance”. This protocol is entirely on-demand, allowing it to scale automatically and supporting multiple routes, for example, for load balancing or to increase robustness. It is also a loop free protocol, handles unidirectional links and is suitable for dynamic medium sized networks.

When a node S (*initiator*) wants to find a destination D (*target*), it starts the route discovery process. It creates a route request with its own information, the target information, the route request identification and space for including the identification of each of the nodes already visited by the message. If a node is the target of a route request, it returns a route reply to the initiator along the reverse route existing in the request, and, when the initiator receives this route reply, it caches the route in the Route Cache for future sends. But, if a node is not the target of a request message, it appends its address to the message and re-transmits the message to its neighbours. In order to discard duplicates, if a node detects a duplicated message, that message is discarded.

If a node does not know the route to a destination, it stores the packet in a Send Buffer, and initiates the route discovery process as described above. There must be a limit for pending discoveries processes because a node may be completely unreachable.

Regarding the maintenance mechanism, when a packet is exchanged, each node must confirm that transaction. But the algorithm can also uses acknowledgment requests in order to maintain connectivity. If after a maximum number of retries, no acknowledgment reply has been received, then that link is treated as “broken” and removed from the Route Cache, a “Route Error” packet is sent to every node that have used that link to forward a message.

A list of DSR family algorithms (algorithms that use DSR basics) is presented next.

DSR Variants

The **Adaptive Multi-path QoS Aware Dynamic Source Routing Protocol for Mobile Ad-Hoc Network** builds highly disjoint path, distributing the packets among those paths; the distribution of messages by the available paths is adjusted through the monitoring of node mobility and message loss and round trip time (Hashim, Nasir, & Harous, 2006). This protocol shows less number of dropped packet and better throughput, when compared to the basic DSR.

The **Adaptive algorithm for increasing the efficiency of DSR algorithm in Ad Hoc network** is another algorithm based on the DSR algorithm (Shqeerat, 2008). This protocol can adapt its routing quickly in the presence of node mobility. This protocol also requires little overhead during periods in which node does not move. This algorithm enhances the caching strategy of DSR and extends it in order to improve error handling, load balancing, re-routing during transmission and re-routing notification. Simulations show that this protocol can perform well in a network with a high number of nodes, high load and mobility, and also reduced overhead when compared with original version.

The **Enhanced Reactive Dynamic Source Routing Algorithm Applied to Mobile Ad Hoc Networks** (ERDSR) protocol is another variant of DSR algorithm that chooses the route by using the bandwidth and the number of hops of the available paths, and regulates dynamically the value of Send_Timeout (Zhao, Zhan, Yao, & Yi, 2005). Compared to the basic DSR, this protocol improves the average route length, the transmission delay and the packet delivery ratio. Simulations show that this algorithm decreases the transmission delay and the average path length and increases the packet delivery rate.

2.1.2. AD HOC ON-DEMAND DISTANCE VECTOR

This analysis is based on the Ad hoc on-demand distance vector (AODV) proposal presented in RFC3561 (Perkins, Belding-Royer, & Das, 2003).

AODV is a routing algorithm that uses periodic message exchange to maintain the connections and sequence numbering to avoid loops; the route discovery based on a flooding mechanism. Each node L maintains a routing table where each record has the following fields:

- **Destination**: address of a remote node R;

- **Next Hop**: address of a node J which is the first node in a route in between L and R;
- **Hop Count**: number of hops between L and R;
- **Sequence Number**: virtual timestamp allowing the disposal of duplicates.

This algorithm is based on three kinds of messages: *Route Requests* (RREQ), *Route Replies* (RREP), and *Route Errors* (RERR):

- **RREQ**: these messages are emitted by a node S that wants to initiate the route discovery for an unknown node D; the message is sent to all the neighbours[1]. The main fields of these messages are:
  - RREQ ID: unique identifier for the search of a node;
  - RREQ Dest: Address of D;
  - RREQ Origin: Address of S;
  - RREQ Hop Count: number of hops already performed;
  - SeqNo: Sequence number.
- **RREP**: reply to a RREQ message; originated by the node D or by an intermediate node K that knows the route to D; the message is sent to the neighbour that is the source of the RREQ message received. The content of each RREP message is:
  - RREP Dest: Address of S;
  - RREP Origin: Address of D;
  - RREP Hop Count: number of hops already performed;
  - SeqNo: Sequence number.
- **RERR**: service message originated when a node loses direct contact to a node Y; the message is sent to all the neighbours. The most important fields of this message are:
  - RERR Dest: Address of node Y that became unreachable;
  - SeqNo: Sequence number.

When a node S needs a route to a destination D, a RREQ is broadcasted, with the contents indicated above; RREQ Hop Count is set to zero. Node S stores the tuple (D, RREQ ID) and waits for a matching RREP message; the maximum waiting time is PATH_DISCOVERY_TIME

[1] Neighbors of node N are all the nodes that are one-hop distant of N.

which is a constant calculated according to the network characteristics. If no RREP message is received, the tuple is removed and D is set to *Destination Unreachable*. In order to reduce the overhead in the network, repeated attempts of RREQ are considered. Before sending a new RREQ message trying to find node D, S must wait NET_TRAVERSAL_TIME milliseconds; in case of a new failure S must wait 2 * NET_TRAVERSAL_TIME milliseconds to the next retransmission; for each additional attempt the node must wait two times the previous waiting time for the response - this strategy is called by *binary exponential back-off*.

When a node K receives a RREQ message the following situations must be considered:

- K address is the equal to RREQ Dest: K prepares a RREP message where RREP Dest is RREQ Origin and RREP Origin is K address; RREP Hop Count is set to 0;
- K knows a route to RREQ Dest: K prepares a RREP message where RREP Dest is RREQ Origin and RREP Origin is RREQ Dest; RREP Hop Count is the hop count associated to the route to D;
- K does not know a route to RREQ Dest: K broadcasts the RREQ message received after incrementing RREQ Hop Count.

When a node K receives a RREP message the following situations must be considered:

- K address is the equal to RREP Dest: K drops the packet and updates the routing table;
- K knows a route to RREP Dest: K forwards the RREP message to the next hop for the RREP Dest and increments the hop count;
- K does not know a route to RREQ Dest: K drops the packet.

On receiving RREQ or RREP messages, the steps done for updating the routing table are the same:

- The entry corresponding to the field RREP/RREQ Origin is considered for update;
- The update is only made if both of the two conditions following are true:
    - The *SeqNo* in the message is greater than *Sequence Number* of the entry;
    - The *Hop Count* in the message is lower than the *Hop Count* of the entry.

When a node K receives a RRER message the following situations must be considered:

- The SeqNo was already processed: K drops the packet;
- The SeqNo was not yet handled:
    - K forwards the RERR message to the neighbours;
    - K marks the routing table entry corresponding to RRER Dest as invalid.

To maintain connectivity, a node sends every HELLO_INTERVAL milliseconds a RREQ message to each neighbour. This message receives immediately a RREP reply. This implies the update of the routing tables.

As final conclusion, and following the information gathered on (Perkins, Belding-Royer, & Das, 2003), AODV is an excellent choice for the establishment of an ad hoc network because:

- Low resources usage: The nodes only store the needed routes, broadcast is minimized and memory requirements are low;
- Quick response to link failure: New routes are quickly established and inactive routes are quickly aged because intermediate nodes can return a new routes;
- The usage of sequence numbers minimizes duplicates and prevents the creation of network loops.

All these properties make AODV scalable to a large number of nodes. The only important drawback is the possibility of long latency in route establishment. A list of AODV family algorithms (algorithms that use AODV basics) is presented next.

AODV Variants

The current state-of-the-art in the AODV family includes many different modifications of the original protocol are presented next.

The **Reverse Adhoc On Demand Distance Vector Routing Algorithm** (RAODV) discovers many reverse routes from the source to the destination (Gowrishankar, Sarkar, &

Basavaraju, 2009). When the destination receives the RREQ message, it floods the network with ReverseRouteRequest messages, creating multiple paths to the source and selecting the best path based on the sequence number and the least hop count. If an intermediate route in the reverse path fails, a RERR message is sent, letting the source and destination to choose other paths.

Modified Reverse Ad Hoc On Demand Distance Vector (MRAODV) (Zarei, Faez, & Nya, 2008): in this protocol when the source wants to communicate with the destination, like in AODV, it sends a RREQ message, but when the destination receives the RREQ message, it broadcasts a ReverseRouteRequest (R-RREQ) message to find the source. When a node receives the R-RREQ message, it calculates a metric called *route stability* (this metric is related to the probability of route to persist for a certain time span). When the source node receives the R-RREQ message, it will have multiple routes to the destination and it will selects the best stable route to the destination. According to (Zahary & Ayesh, 2007), MRAODV waits too much time to check if there are more routes; receiving so many routes leads to memory overhead and delays.

Threshold Routes AODV (TRAODV) (Zahary & Ayesh, 2007) is similar to MRAODV just with the addition of a time limit for waiting for alternative routes.

The **AODV with Path Accumulation** (AODV-PA) is proposed by (Gwalani, Belding-Royer, & Perkins, 2003). AODV-PA is similar to DSR, by including source route accumulation in AODV as a route discovery technique. This is made when RREQ and RREP messages are passed through the network, as each node appends its own address on the message and letting the nodes update their routing tables with the information contained in that message (Figure 2.1).

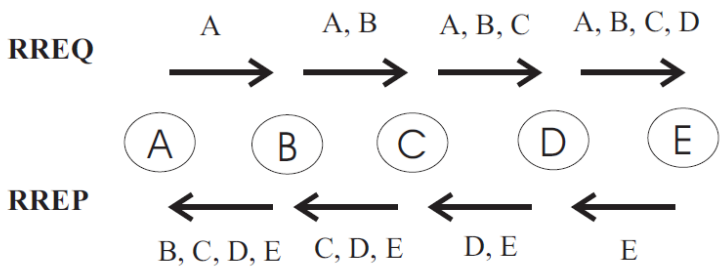


FIGURE 2.1 ACCUMULATION IN RREQ AND RREP MESSAGES

Multiple Next Hops (MNH) (Porekar, 2003) and (Jiang & Jan, 2001) is another variant of AODV. According to the authors, the main goal was a reduction of the bandwidth consumption of AODV when searching a new route in the presence of a link failure. In the routing table of each node, the entry corresponding to a given destination can have more than one pair (*nextHop*, sequence number). The multiple routes can be detected by receiving multiple RREP messages from different neighbours. The excessive number multiples routes can lead to unjustified overhead related to the maintenance of this type of routing tables.

AODV Variants comparison

According to (Jiang & Jan, 2001), the MNH algorithm has a good behaviour in new path discovery after a link failure, because the routes are maintained by intermediate nodes. With this property the path length is increased by one hop. However, this algorithm reduces time on route discovery and route reconstruction.

The AODV-PA proposed by (Gwalani, Belding-Royer, & Perkins, 2003) shows good results when compared to AODV in a network characterized by high load and high mobility networks. This algorithm scales well for large networks and has a good packet delivery rate and low delay. Its routing information decreases with an increase in the load and is suitable for high load scenarios.

Comparisons made between TRAODV and MRAODV show better results of TRAODV with route availability as a metric. This algorithm reduces the overall routing delay and waiting and routing reconstruction overhead.

Between RAODV and MRAODV, the results show that RAODV outperforms MRAODV when a route fails (Zarei, Faez, & Nya, 2008). In the presence of network changes, RAODV can easily select new routes with minimum path length in the routes previously discovered. Compared to AODV, RAODV has a good packet delivery ratio, is scalable in large networks and is suitable for high mobility networks.

(Gowrishankar, Sarkar, & Basavaraju, 2009) performed some comparisons and the results shows that RAODV shows better results than AODV.

With this analysis a few conclusions can be made about **Issue 2** and **Issue 3**. **Issue 3** is directly accomplished by MNH because it adapt itself to network changes and using information gathered from **Issue 4**, it can adapts intelligently improving routing. AODV already enables **Issue 3**, because when a device enters in the network, it will search for its neighbours with Hello messages, but MNH can adapt to link failures in an efficient way. In order to adapt to the dynamic network, such as link failures but not device failures, this can be achieved with flags in the routing tables, as proposed in AODV, instead of just deleting the entries.

Issue 2 can be accomplished with AODV-PA, but with some changes in the algorithm. AODV-PA only accumulates the references of the nodes where the message has been. To achieve **Issue 2**, this algorithm must accumulate more information in order to be used in routing decisions; this information can be the type of link between the node after and before, a metric that represents the quality of the link or others, etc... With this behaviour, the routing decisions can be more accurate and reliable because more information is exchanged but only information that could be used for immediate decisions, not long term decisions, as long term decision are not reliable because the network is dynamic.

2.1.3. LOCATION-AIDED ROUTING

This analysis is based on the Location-aided routing (LAR) proposal presented in (Ko & Vaidya, 1998). LAR is a routing protocol that considers that the network is divided in *request zones*; the criteria to join a given request zone is the location of a node (GPS coordinates). Route discovery procedures occur in the context of a zone.

When a node wants to find a route, it initiates the route discovery phase based on the *flooding* technique similar to the one used by DSR and AODV, by sending a *route request* message to the neighbours. If a neighbour is not the destination, it re-broadcasts the message to its neighbours. In order to avoid redundant transmissions, the nodes only broadcast the request once; the duplicates are detected by the usage of sequence numbers.

If a node is the destination of a route request message, it sends a *route reply* to the sender as in AODV. If a path discovery has failed, the requester must be aware (after a given timeout), that is necessary to retransmit a route request.

The usage of location information in order to reduce routing overhead is the main goal of this algorithm. Node positions may be acquired by some kind of positioning system (like Global Positioning System – GPS).

Suppose that node S tries to find a route to node D, and that node S knows the location (X_0, Y_0) of node D at time t_0; if v is the speed of node S and also known, S can expect to find D at time t_1, in a circle centred in (X_0, Y_0) with radius $v(t_1-t_0)$. This circle is called "*expected zone*".

The notion of "*request zone*" is also important in the protocol. This means that a node only forwards a route request for a node D, if D belongs to the request zone. In order to be able to find a destination D, the expected zone should be included in the request zone.

A list of LAR family algorithms (algorithms that use LAR basics) is presented next.

LAR Variants

The **Location Aided Cluster Based Energy-efficient Routing** (LACBER) is an algorithm proposed by (Deb, Roy, & Chaki, 2009) based on the LAR protocol. LACBER is a location-aided and energy efficient routing algorithm and can work in areas with low GPS coverage. The authors claim that this protocol has better location properties and lower energy and bandwidth consumption than the LAR solution.

The **Distance-Based Location-Aided Routing** (DBLAR) (Wang, Wu, Weifeng, Pengrui, & Shen, 2008) protocol monitors the nodes positions (LI) and when location changes are detected the process of route discovery is adjusted accordingly, thus lowering the number of network floods. The results published suggest that this protocol performs better than LAR in packet delivery ratio, average end-to-end delay and routing-load.

(Xue & Li, 2001) proposed the **Location-aided Power-aware Routing Protocol in Mobile Ad Hoc Networks** (LAPAR). The practical and theoretical analysis made by the authors show that

this algorithm is power-efficient and enhances previous solutions but in some cases inaccurate location information is produced.

2.1.4. ANT-COLONY-BASED ROUTING

The text below analyses the Ant-colony-based routing (ARA) proposal presented in (Günes, Sorges, & Bouazizi, 2002).

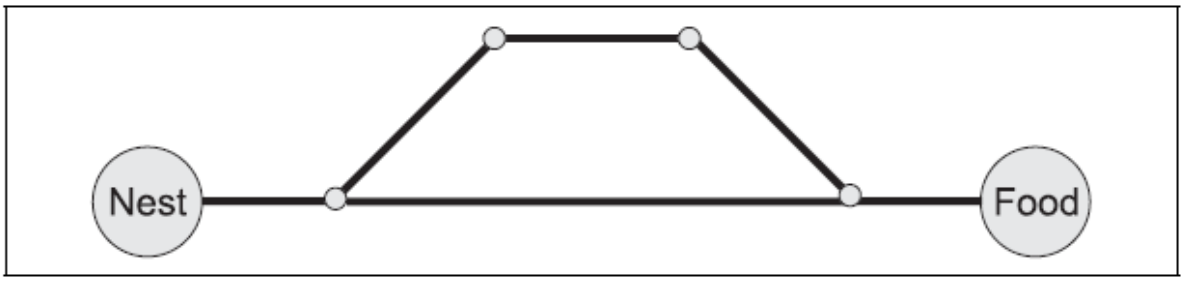


FIGURE 2.2 TWO ROUTES FROM THE NEST TO THE FOOD PLACE

The ARA protocol attempts to reduce the communication overhead by simulating the food searching behaviour of the ants, using swarm intelligence, more specifically, the ant colony based meta heuristic. This algorithm is distributed, loop-free, has a demand-based operation and allows that the nodes sleep for part of the time.

The ants start from their nest and walk to the food deploying *pheromone* to mark the travelled path. If a path is used by many ants, it has a huge concentration of pheromone. In Figure 2.2 an example of two routes to the food is shown: the first ant chooses randomly which path to follow and on their way back, it uses the already known path. After some time, the concentration of pheromone on the shorter path will be higher and then all the ants will only use this path.

When the algorithm wants to find a route, it initiates the discovery phase with a *forward ant* (FANT), which is an agent that establishes a track of pheromone to the source; and a *backward ant* (BANT), which is an agent that establishes a track of pheromone to the destination. First the source sends a FANT to the neighbours, and the neighbour that receives for the first time a FANT, creates a record in the routing table with (destination address, next hop, pheromone value), and then re-broadcasts the FANT to its neighbours. Each FANT has a sequence number that is used to avoid cycles. When the FANT reaches the

destination, it is discarded and a BANT is created and sent back to the source with the same behaviour as the FANT. When the source receives the BANT, the path is established and communication can take place.

Once the path is acquired, the route has to be maintained, i.e. the pheromone values have to be refreshed because they fade along the time. The data packets can update these values. When a link fails, for instance, due to mobility, the algorithm recognizes the failure through a missing acknowledgment. If a node receives a ROUTE_ERROR messages from a link, it deactivates the link and sets the pheromone to zero, then searches for another path in the routing table, otherwise, the node informs the neighbours that it cannot relay packets.

A list of ARA family algorithms (algorithms that use ARA basics) is presented next.

ARA Variants

The **Enhanced Ant Colony Based Algorithm for Routing in Mobile Ad Hoc Network** is a protocol proposed by (N. K. & Viswanatha, 2008) aiming at improving the performance of the ARA algorithm. The main change seems to be the use of data packets for maintaining information about the link behavior.

(Liu, Zhang, Ni, Zhou, & Zhu, 2008) proposed another Ant-Colony based routing algorithm for mobile ad-hoc network called **AMQRA**. This protocol maintains a set of QoS parameters for each link namely time delay, packet loss rate, effective bandwidth, queue buffer length, etc… this information is used for improving packet delivery ratio and reduce the end-top-end delay, according to the authors by, respectively 9%-22 and 14%-16%, when compared to ARA.

The **Position Based ANT Colony Routing Algorithm** (PBANT) is a variant of the ARA algorithm proposed by (Sujatha, V.P, Namboodiri, & Sathyanarayana, 2008). This algorithm is very similar to ARA, but the position of nodes is known. This algorithm uses position information to build a heuristic in order to reduce the time needed to establish a route to the destination and the number of control messages. The authors claim that this approach is robust, scalable and suitable for ad hoc networks with irregular transmission ranges.

2.2. CONTEXT ANALYSIS

2.2.1. THE NOTION OF CONTEXT

The notion of Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between the user and application, including the user and applications themselves (Soylu, De Causmaecker, Desmet, & Leuven, 2009). To introduce the notion of context in routing protocols, the nodes must have mechanisms to obtain information about their environment (Madhavapeddy, Scott, & Sharp, 2003). This is important because the network is dynamic and may have different behaviors over the time, so nodes may need to change some operational parameters in order to adapt themselves to those changes (Ay, 2007). This kind of behavior awareness, or context awareness, is known as *adaptivity* (Soylu, De Causmaecker, Desmet, & Leuven, 2009). Therefore, nodes can gather information, infer about if it is important to any type of context; additionally nodes can extract information about other nodes from the contents of received messages. This information can also trigger changes in the node behaviour.

The context is defined by a set of context dimensions; an example of a context dimension is the location of entities, but others can be considered. Other ones such as, communication power, location, node's velocity, distance between nodes, link costs, processing or storage capabilities, which can also be used. Besides context dimensions related with nodes, messages can also have context information associated, namely source, destination, priority, delivery deadline, behavior after a delivery failure, among others.

The context may be acquired by an explicit declaration, for example as user input, or by implicit declaration, namely by monitoring user behaviour (Schmidt, Beigl, & Gellersen, 1999). This leads us to the need to infer and reason about information in order to create information about context and also, a way to reason about a given context. With this kind of reasoning, nodes can infer explicit and implicit information in order to change their parameters.

In order to model this information, the approach of "ontologies" is a suitable choice. The term ontology originates from philosophy and refers to the discipline that deals with the

existence of things (Ay, 2007). The use of this approach in network routing can improve and enable interoperability between nodes.

```
<owl:ObjectProperty rdf:ID="hasBankAccount">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person"/>
        <owl:Class rdf:about="#Corporation"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
```

Since that Ontology Web Language (OWL) (OWL, 2010) is more expressive, allows more interoperability and also supports RDF,

In computer science knowledge/ontologies can be expressed in Resource Description Framework (RDF) (RDF, 2010) or OWL, among others. The code above, presents an example of the use of OWL to describe an ontology. RDF is a specification of a precise semantics, and corresponding complete systems of inference rules, for RDF and RDFS and OWL, and is designed for use by applications that need to process the content of information, instead of just presenting information to humans. OWL facilitates greater machine interpretability of web content than that supported by XML, RDF, and RDFS by providing additional vocabulary along with formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

2.2.2. THE CONTEXT IN ROUTING

There are a few approaches of routing algorithms that use natively context information from the network or from other layers. Many routing algorithms do not consider the context of the network, but, the context influences the routing performance (Saeed, Abbod, & Al-Raweshidy, 2008).

A mechanism that uses the concept of context is presented in (Saeed, Abbod, & Al-Raweshidy, 2008). This article describes a system that can adapt itself to changes in the network context and select the relevant parameters for choosing the best routing algorithm to handle the mentioned changes. In (Nickray, Dehyadgari, & Afzali-kush, 2009) the use of a society of context-aware agents to take decisions about packet routing is presented. (Das, Wu, Chandra, & Charlie Hu, 2008) describe a procedure (including context defined metrics) for finding efficient routes in a self-organizing way. Finally, (Shah & Qian, 2009) present a solution based on simple parameters like the velocity of nodes and the distance between them to determine the route lifetime in the network. This last solution shows good results when compared with another algorithms that do not use context information; this suggests that context can be useful when defining a new routing algorithm.

2.3. NETWORK SIMULATORS ANALYSIS

As explained before, in this thesis the experimentation with routing algorithms will be performed over a network simulator. In the following some network simulators are presented.

2.3.1. NETWORK SIMULATORS

TOSSIM

TOSSIM is a discrete event simulator for TinyOS sensor networks (Levis, 2010). It allows users to compile the applications in the simulator before testing them on the TinyOS environment. With this, it is possible to debug and analyze the algorithms in a controlled environment. This simulator does not consider aspects like radio propagation and power consumption.

OMNeT++

OMNeT++ is not a network simulator itself, but it can be described as a framework to build network simulators. This construction is performed by combining modules available from different origins (Community, OMNeT++ Community Site, 2010). It is composed by a

simulation kernel library (discrete-event environment simulator), a compiler, an IDE based on Eclipse IDE, a GUI for the simulation execution, a command line, some utilities and documentation. There are some models implemented namely:

- INET Framework, which contains models for IP, TCP UDP and other protocols;
- Mobility Framework that supports the simulations of wireless and mobile network;
- PAWIs which is a wireless sensor network simulator developed by the Institute of Computer Technology, University of Technology, Vienna.

J-SIM

J-Sim (also known as JavaSim) is an open-source, component based network simulator, written entirely in Java (J-Sim Official, 2010). This simulator is implemented on top of a component software architecture called *autonomous component architecture* (ACA) (Sobeih, et al., 2005). The Java implementation and ACA organization, makes J-Sim platform independent, extensible and reusable. This simulator can be integrated with some languages like Perl, TCL or Python, and the version 1.3 of the simulator has been integrated with a full implementation of a TCL interpreter called Jacl. This simulator defines its classes in Java and uses TCL/Java to assemble them together.

NS-2

The NS-2 is the most popular network simulator in academic and research environments. This tool is a discrete-event network simulator that can be used for research and educational purposes, because it supports the simulation of TCP, routing and multicast protocols in many different settings, such as, sensor networks, 802.11 and satellite protocols (The Network Simulator - ns-2, 2010). This simulator objects are written in C++ to guarantee efficiency and oTCL scripting can be used for simulation definitions, like objects configuration and event schedule.

NS-3

The NS-3 simulator (nsnam) is a discrete-event network simulator that can be used in research and education, for the study of Internet protocols and large-scale systems (The ns-3 network simulator, 2010). It aims to be a replacement of the NS-2 simulator, but it is not compatible with it; and the nsnam acronym represents the concatenation of *ns* (network simulator) with *nam* (network animator).

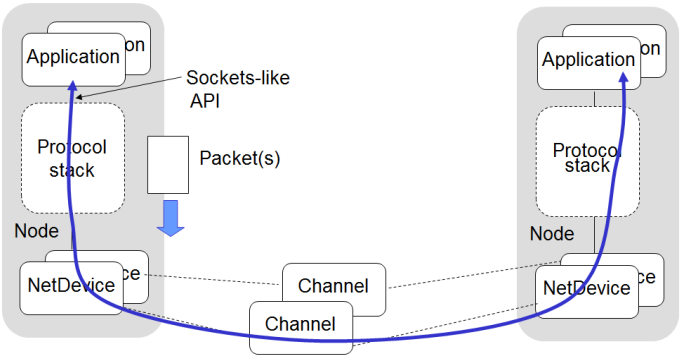


FIGURE 2.3 NS-3 BASIC MODEL

The Figure 2.3 represents the basic model used in NS-3 simulator. It can be divided in three key objects:

- **Nodes**: contain Applications, Protocol stacks and NetDevices, and can be seen as a computer on which applications, stacks and NICs are added;
- **Packets**: each network packet contains:
  - **Byte buffer**: representation of headers;
  - **Tags**: user-provided structures like flow identifiers;
  - **Metadata**: used to describe the header that have been serialized.
- **Channels**: are connected to net devices and are the abstraction of communication channels, like WiFi or CSMA channels.

This modular simulator is written in C++, with some Python scripting, and tries to enhance the NS-2 simulator in areas related to interoperability, memory management and debugging facilities.

2.3.2. NETWORK SIMULATOR COMPARISON

As a conclusion of the network simulators analysis, and according to (Ruzzelli, 2008) and (Weingärtner, Lehn, & Weh, 2009) and (Varga & Hornig, 2008), NS-2 and NS-3 allow an easy development of custom simulators and have a well active community, that supplies good quality documentation, online help and contributes with code. The only drawbacks are the limited GUI and the complex debugging. The OMNeT++ is another well-known simulator that has good support from the community, has a good GUI, but suffers from scalability problems in large networks. J-SIM is extensible, platform independent and supports several protocols, but has a smaller supporting community when compared with the previous ones. TOSSIM is scalable to large networks and the code can be deployed in TinyOS based motes. This simulator can be only simulate TinyOS-based devices and does not have models for battery and variable CPU consumption. The following table presents a comparison between the simulators presented; in the columns the criteria relevant for this thesis work are included:

- **Mobility model**: this criteria is necessary, because the algorithm should be tested in a mobile environment as presented in section 1.2;
- **AODV implementation**: if the simulator already implements the AODV algorithm, is desirable. Using that implementation as a correct implementation can help the development and improvement of the proposed solution;
- **Routing access**: this is crucial, without the access to the routing layer, this work cannot be done;
- **Results**: the simulator has to present results and should let debug and let define output results;
- **Documentation**: a good documentation, online and offline is desirable. A network simulator is a complex application and is hard to know all the details within the proposed time.

TABLE 2.1 SIMULATORS COMPARISON

	Mobility model	AODV implementation	Routing access	Results	Documentation
TOSSIM			•	•	Few
J-SIM	•	•	•	•	Acceptable
OMNeT++	• (with model)	• (with model)	•	•	Good
NS-2	•	•	•	•	Good

NS-3	•	•	•	•	Good

Looking at this table is possible to conclude that J-SIM, NS-2 and NS-3 are the only possible choices to this work. The availability of an AODV implementation is a vital element in the decision. This consideration eliminated TOSSIM; OMNeT++ has an AODV component but its use is complex. The J-Sim simulator was also abandoned due to the lack of good documentation and the smaller community support. NS-2 apparently is being replaced by NS-3.

The chosen network simulator was NS-3, because it satisfies all the requisites, it has good documentation and has a record of large recent development efforts.

3. PROPOSED ALGORITHM

This chapter presents a routing algorithm that is the focus of this work. In the following, the proposed algorithm will be called by C-AODV, which stands for Context in AODV.

The C-AODV implements the same basic mechanisms as AODV. First, it populates the routing table with the information about its neighbours, however, the major change is the fact that the routing tables are different because some entries disappear and two other entries are created, related to context information (CA, CB). After this, when the application wants to lookup a route, it will ask the routing mechanism for a route and if the route for the destination is unknown, the algorithm will start a new discovery phase (Figure 3.1).

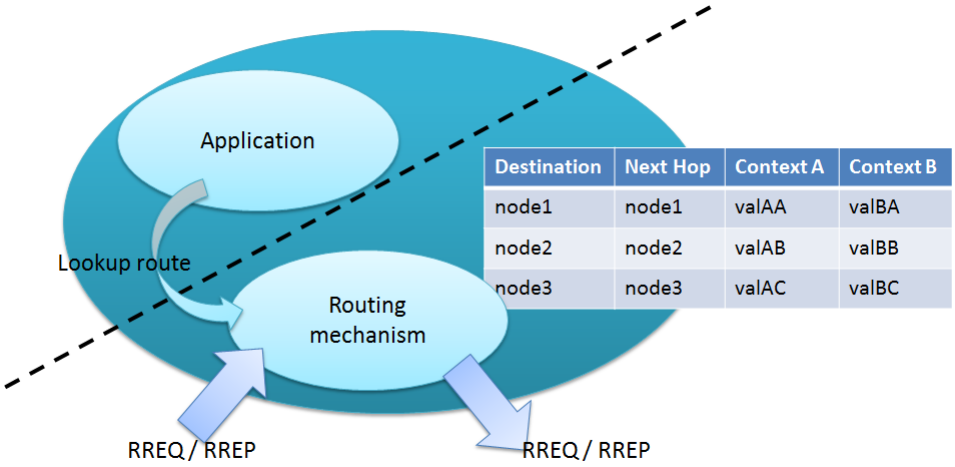


FIGURE 3.1 LAYER COMMUNICATION SCHEME

3.1. ASSESSING REACTIVE ROUTING PROTOCOLS

In this section, the features of the reactive routing algorithms described in the previous chapter are assessed against the requisites of a routing algorithm for the IoT presented in section 1.2.

All the proposed goals can be accomplished.

- **Issue 1** is achieved using the AODV algorithm or a variant (like AODV-PA), because it only requires small amounts of information regarding the network in order to make routing decisions. The AODV-PA is a good approach, since it shows better results than AODV or DSR (Gwalani, Belding-Royer, & Perkins, 2003). According to (Bouhorma,

Bentaouit, & Boudhir, 2009) AODV shows better results than DSR and "*A combination of the protocols can be used for good result*". This observation supports our decision of implementing a routing algorithm that takes features from both AODV and DSR;

- **Issue 2** is achieved with AODV-PA, and with some improvements are needed. The algorithm only accumulates nodes' references, while this work needs more information in order to make more intelligent routing decisions;
- **Issue 3** is directly achieved by MNH because it has self-repairing properties that will allow energy and processing savings without extra message exchange;
- **Issue 4** can be achieved by the routing algorithm proposed in this work. This algorithm incorporates elements from AODV-PA, MNH; additionally it will incorporate the notion of Context information from other layers, namely the application layer (for example, message deadlines) and the physical layer (namely, power consumption).

3.2. EXTENDING AODV

3.2.1. INCLUDING CONTEXT

An issue to be addressed is how to evaluate routes based on network and message context. Due to possible multiple routing table entries for the same destination, the traditional approach would be to choose the hop count field to choose the best route. Due to the complexity of the network and its dynamicity, the notion of context must be introduced in the routing algorithm (see 2.2.2), this allowing the consideration of a context information like message priority.

Figure 3.2 gives an overview of the use of Context in routing decisions:

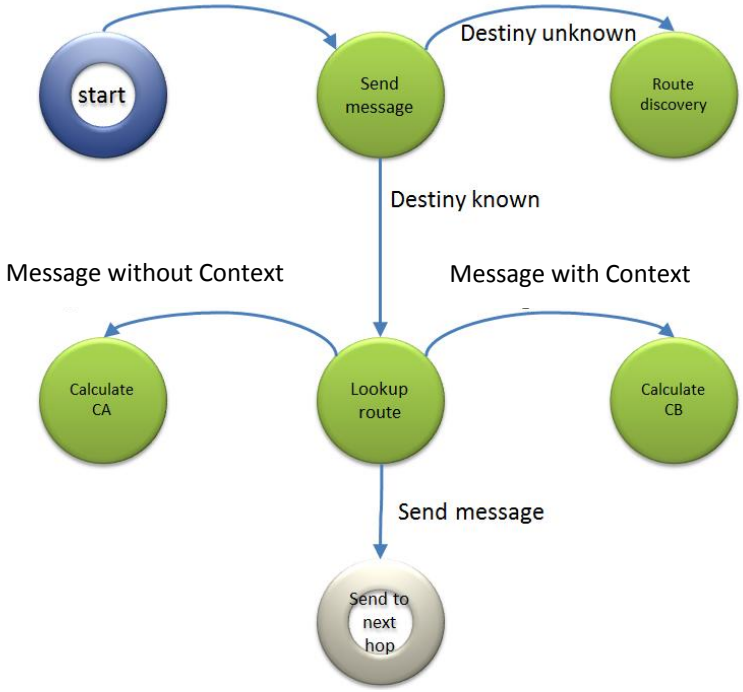


FIGURE 3.2 ROUTING BEHAVIOUR

Based on Figure 3.2, it is possible to see more easily the two phases of the algorithm: the first consists of route discovery process, followed by a process to populate the routing table. The second is to choose the best route for a message, given its context.

In order to create a Context-aware routing algorithm one must associate context information to each entry in the routing table. In the current version the context information is stored as a pair of integer values - Context-A (CA) and Context-B (CB). An example of the use would be:

- CA, represents the “physical” effort to a node communicates with another;
- CB, represents an application defined value for a deadline for delivery a message.

The number and type of context may be user defined or system defined as presented previously in chapter 2.2 - Context analysis. Therefore, when the source node N1 wants to discover a route to a destination, before sending the RREQ message to a neighbour N2, it calculates CA and CB using a function (ƒ(N1, N2) → valueOfCA), where *valueOfCA* would be an integer value, in which 0 represents a good quality link (like *localhost*) and larger values represent lower quality links. The CB is calculated in a similar way using another function. The integration of the context analysis in the routing algorithm is as follows:

```
function RecvRREQ(RREQPacket rreqPacket) {
    if(rreqPacket.isDuplicated())
        return;
/* calculate new routes based on the path accumulation in packet */
    int numberOfContexts = rreqPacket.getNoContexts();
    Iterator it = rreqPacket.getPathAccumulation().iterator();
    while(it.hasNext()){
        Iterator<E> itCtx = rreqPacket.getPathAccumulation().iterator();
        Context ctx = new Context(numberOfContexts);
        while(itCtx.hasNext()){
            ctx.add(itCtx.next());
        }
        Path pa = it.next();
        this.routingTable.addEntry(pa.getNode(), pa.getPrevious(), pa.getSeqNo(), ctx);
    }
/* I am the destination for this packet */
    if(rreqPacket.getDestination() == this){
        SendRREP(rreqPacket);
        return;
    }
/* forward packet to neighbors */
    Iterator it = myNeighbors().iterator();
    while(it.hasNext()){
        Node neighbour = it.next();
        Context ctx = new Context(numberOfContexts);
        for(int i = 0; i < numberOfContexts; i++){
            ctx.add(ContextHelper.getValue(i, this, neighbour));
        }
        RREQPacket newRreqPacket = new RREQPacket (rreqPacket);
        newRreqPacket.addPathAccumulator(this, ctx);
        ForwardRREQ(newRreqPacket, neighbour);
    }
}
public function RecvRREP(RREPPacket rrepPacket) {
    if(rrepPacket.isDuplicated())
        return;
/* calculate new routes based on the path accumulation in packet */
    int numberOfContexts = rrepPacket.getNoContexts();
    Iterator it = rrepPacket.getPathAccumulation().iterator();
    while(it.hasNext()){
        Iterator<E> itCtx = rrepPacket.getPathAccumulation().iterator();
        Context ctx = new Context(numberOfContexts);
        while(itCtx.hasNext()){
            ctx.add(itCtx.next());
        }
        Path pa = it.next();
        this.routingTable.addEntry(pa.getNode(), pa.getPrevious(), pa.getSeqNo(), ctx);
    }
/* I am the destination for this packet */
    if(rrepPacket.getDestination() == this){
        SendRREQ(rrepPacket);
        return;
    }
/* forward packet to next hop, must exist */
    Node nextHop = this.routingTable.lookupRoute(rrepPacket.getDestination());
    Context ctx = new Context(numberOfContexts);
    for(int i = 0; i < numberOfContexts; i++){
        ctx.add(ContextHelper.getValue(i, this, nextHop));
    }
    RREPPacket newRrepPacket = new RREQPacket (rrepPacket);
    newRrepPacket.addPathAccumulator(this, ctx);
    ForwardRREP(newRrepPacket, nextHop);
}
```

As in AODV-PA, the control messages include information about each node belonging to the path. In our algorithm besides the node references the context information calculated in each node is also included.

3.2.2. INCLUDING AODV-PA AND MNH IDEAS

As explained in 3.1, with AODV as background, AODV-PA and MNH seem to complement each other. AODV-PA allows the nodes to have a notion of the path followed by the

messages, while MNH creates new hops in the routing table entries. The proposed solution is to merge these protocols and create a more robust algorithm with the properties of these two. An improvement in the routing tables, concerning the MNH protocol, seems necessary in order to consider the sequence numbers in the multiple hops, leading to more effective routing.

The routing properties and periodical beaconing of AODV will still be used because it is a requisite for a reactive protocol, to maintain its routing tables fresh and evaluate environment changes. However, with the properties of AODV-PA, the overhead and bandwidth consumption are reduced and the proposed algorithm will still be light and require less storage capabilities from the nodes.

To integrate both algorithms, a modification of the AODV-PA must be done. AODV-PA assumes that RREP messages are sent via unicast back to the source, but MNH assumes that they are multicasted. This question is solved by the AODV-PA proposal.

So, when the RREQ and RREP messages are generated or forwarded, the solution acts like nodes in AODV-PA, by appending their own address on those messages, before forwarding them to the next node.

Every time that a node receives a RREQ or a RREP message, which contains all the nodes traversed, including the source node, it will update the routing information about those nodes. So, if a route to a node exists and if the base metric to any of the intermediate nodes is less than the previous existing base metric the entry is updated. If the node is unknown, a new entry is created, the sequence number is set to zero and the hop count is retrieved by the message.

The route request procedure is the same for AODV, AODV-PA and MNH: the source sends the RREQ message to its neighbours, and when one of the nodes knows the route to the destination or it is the destination itself, it sends back a RREP message and discards the RREQ message.

With this mechanism, each node can have more than one forward link and therefore when a link fails, the AODV route maintenance procedure can be initiated (which can be inefficient). If a link fails, the forwarding nodes can detect and invalidate the next hop of that failure in

the routing table entry. In using this, the RERR messages can be avoided, leading to greater network performance.

Taking an example of a source (S) to a destination (D) with the following routes:

- S → A → F → G → K → D
- S → B → E → I → J → D
- S → C → F → H → K → D

F has two next hops to D, G and H. If G fails, node F can choose immediately H as the next hop and eliminate node G from the routing table entry. Next, if H fails, node F eliminates node H from the routing table entry and informs the previous nodes (A and C), as they do not have a route to D. They will also inform S, which will choose the other route. This will avoid having to reinitiate the route discovery procedure.

Given the scenario presented in Figure 3.3, when S starts a route discovery these are the path accumulations that arrive to D:

- S $_{(0+8);(0+8)}$→ 1 $_{(8+1);(8+8)}$→ 2 $_{(9+2);(16+08)}$→ D $_{(12);(24)}$
- S $_{(0+8);(0+7)}$→ 3 $_{(8+8);(8+6)}$→ 4 $_{(16+8);(14+7)}$→ D $_{(24);(21)}$
- S $_{(0+8);(0+9)}$→ 5 $_{(8+8);(8+5)}$→ 6 $_{(16+8);(13+4)}$→ 7 $_{(24+8);(17+7)}$→ D $_{(32);(24)}$
- S $_{(0+8);(0+8)}$→ 1 $_{(8+7);(8+5)}$→ 3 $_{(15+8);(13+6)}$→ 4 $_{(23+8);(19+7)}$→ D $_{(31);(26)}$

The CA/CB values are calculated right before the message propagation.

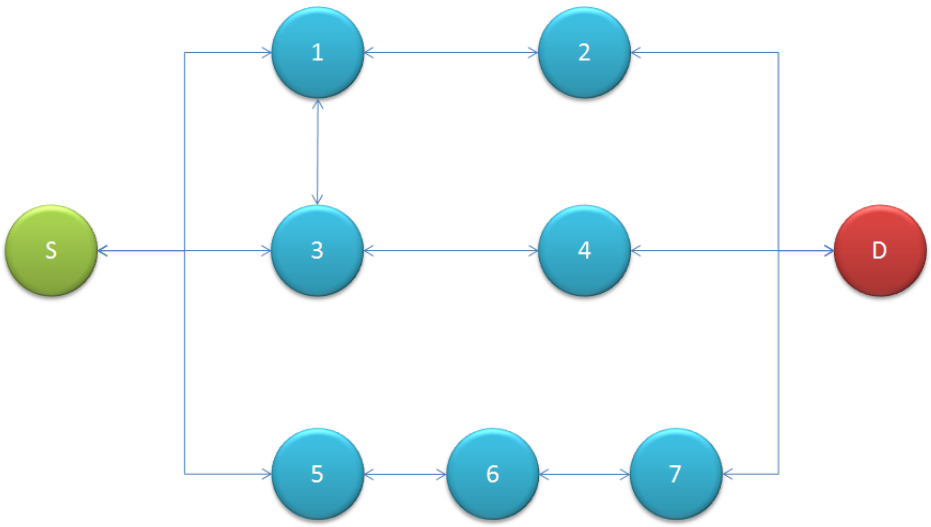


FIGURE 3.3 A POSSIBLE NETWORK

When S receives all the RREP messages, its routing table to D is the following table (without the *rank* entry given by the CA):

TABLE 3.1 EXAMPLE OF A PROPOSED ROUTING TABLE FOR NODE S

Node	NextHop	SeqNo	CA	CB	rank
D	1	1256	12	24	#1
D	3	1256	24	21	#2
D	5	1256	32	24	#3

These messages can have duplicate information due to the path accumulation used in the AODV-PA. This can be a problem due to extra data in the messages, but it brings more information to S; this extra information can be used for example to unidirectional links issue. An example of duplicated information in the context of Figure 3.3 is when S receives two RREP from node 3, with different information; one of them corresponding to the route (D-4-3) and the other (D-2-1-3).

When we choose to include in C-AODV the path accumulation typical of AODV-PA, we opted by having more information in the messages and simplified routing tables in each node. This was motivated by the consideration that the nodes can have limitation in storage and processing capabilities.

An alternative solution would be to force the S node, to keep information about all the pending RREQs. This way, when S sends a RREQ to a node, it stores that information based on an RREQ ID in a structure like (RREQ_ID, NextNode, Destination, SequenceNo). With this approach, when D receives a route request, it sends back the route reply, as usual, but also with the information about the RREQ_ID and the NextNode (relatively to S). So, when S receives a RREP, it can search in the RREQ waiting messages structure for a pair (RREQ_ID, NextNode) and then retrieve the information and update the routing table information.

Due to the nature of the problem, the last approach seems to be a bad choice, because there may be a high number of route discovery messages and that structure can become

very large, and requiring lots of resources of the node, which must be avoided. So, preferentially, a more complex message system is preferable.

4. C-AODV PROOF-OF-CONCEPT

This chapter presents the choices made for the C-AODV proof-of-concept implementation. First, the context dimensions chosen and their sources are presented. As explained before, the C-AODV implementation will be built over the basic AODV algorithm implementation included in NS-3 simulator. Before explaining how the extensions were performed, and an explanation how to extend or create a routing algorithm in the NS-3 simulator is given.

4.1. CHOOSING CONTEXT TYPES

Picking upon the concept presented in Figure 1.1, the scenario that is proposed to be tested is the following:

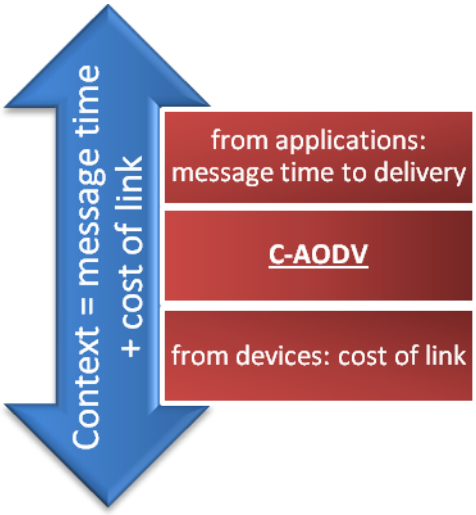


FIGURE 4.1 IMPLEMENTATION SCENARIO

The proof-of-concepts aims to prove the advantage of the use of knowledge from upper and lower layers in order to improve the routing behavior. As explained before, the joint use of upper layer context information (the context of the messages and its properties), and lower layer context information (the physical status of the device or its links), allow better routing decisions. In the implementation described:

- For lower layer context information (CoL), power consumption on sending the message is considered;
- For upper layer context information (ToL), message delivery deadline is used.

4.2. C-AODV IN NS-3

The proposed solution is implemented and was simulated in the nsnam (NS-3) simulator. The Figure 4.2 presents an overview of NS-3 main components:

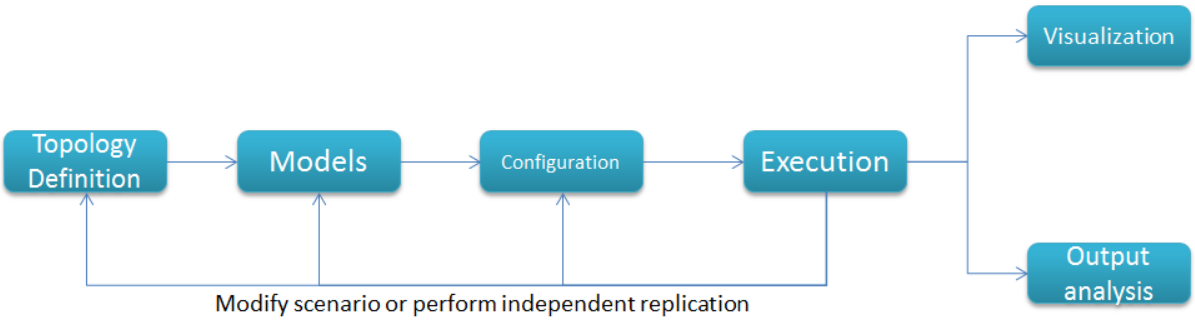


FIGURE 4.2 OVERVIEW OF NS-3 FEATURES

The definition of a routing algorithm is part of the **Models** feature of the NS-3 (Figure 4.2), and after the configuration of the topology, it is possible to use it.

The routing information, as well as, the communication models such as *WiFi* or *TCP schemes* are defined in **Models**. The **Topology Definition** is where the *helper APIs* and *containers* are defined. The **Configuration** part is where the *attributes* and *names* are defined. The **Execution** is the definition of the schedulers and *emulation modes*. And finally, the **Visualization** and **Output Analysis** are responsible for the presentation of the simulation results.

NS-3 includes an implementation of AODV that identifies the nodes using IPv4 addresses. Of course the addresses are only used for guaranteeing an unique node identification, but the traditional Internet routing protocols are not applicable.

4.2.1. INTRODUCING A NEW ROUTING ALGORITHM IN NS-3

Due to the difficulty of understanding all the internals of a network simulator like nsnam, the solution passed by changing the previous implementation of the AODV algorithm that is provided in the NS-3 package.

The basic implementation of the AODV algorithm can be found in the directory *~/repos/ns-3-allinone/ns-3-dev/src/routing/aodv*.

The files *waf* and *wscript* are the files responsible for the compilation of the code in the simulator. The files that were changed in order to implement the solution were the *aodv-packet.h* and *.c* that handle the packets of the algorithm, like the RREQ and RREP packets; the *aodv-routing-protocol.h* and *aodv-routing-protocol.c* is the main class for all the algorithms and the *aodv-rtable.h* and *aodv-rtable.c* are responsible for the management of the routing table. In order to implement and compile new classes to be used in the previous implementation of the algorithm, some lines must be inserted in the *wscript* file in order to indicate the building script to compile them too.

In order to use a new routing algorithm, the main class must extend the ipv4 "generic" routing protocol existing in the simulator; the header to this class can be found in *ipv4-routing-protocol.h* and the two main methods that must be extended are:

```
(…)
/**
  * \brief Query routing cache for an existing route, for an outbound packet
  *
  * This lookup is used by transport protocols.  It does not cause any
  * packet to be forwarded, and is synchronous.  Can be used for
  * multicast or unicast.  The Linux equivalent is ip_route_output()
  *
  * \param p packet to be routed.  Note that this method may modify the packet.
  *        Callers may also pass in a null pointer.
  * \param header input parameter (used to form key to search for the route)
  * \param oif Output interface Netdevice.  May be zero, or may be bound via
  *          socket options to a particular output interface.
  * \param sockerr Output parameter; socket errno
  *
  * \returns a code that indicates what happened in the lookup
  */
 virtual Ptr<Ipv4Route> RouteOutput (Ptr<Packet> p, const Ipv4Header &header, Ptr<NetDevice> oif, Socket::SocketErrno &sockerr) = 0;
 (…)
```

And the second:

```
(...)
/**
  * \brief Route an input packet (to be forwarded or locally delivered)
  *
  * This lookup is used in the forwarding process.  The packet is
  * handed over to the Ipv4RoutingProtocol, and will get forwarded onward
  * by one of the callbacks.  The Linux equivalent is ip_route_input().
  * There are four valid outcomes, and a matching callbacks to handle each.
  *
  * \param p received packet
  * \param header input parameter used to form a search key for a route
  * \param idev Pointer to ingress network device
  * \param ucb Callback for the case in which the packet is to be forwarded
  *        as unicast
  * \param mcb Callback for the case in which the packet is to be forwarded
  *        as multicast
  * \param lcb Callback for the case in which the packet is to be locally
  *        delivered
  * \param ecb Callback to call if there is an error in forwarding
  * \returns true if the Ipv4RoutingProtocol takes responsibility for
  *       forwarding or delivering the packet, false otherwise
  */
 virtual bool RouteInput  (Ptr<const Packet> p, const Ipv4Header &header, Ptr<const NetDevice> idev,
                           UnicastForwardCallback ucb, MulticastForwardCallback mcb,
                           LocalDeliverCallback lcb, ErrorCallback ecb) = 0;
(...)
```

These methods are self-explanatory, and with them it is possible to define the handling of an incoming packet including the decision about the routing.

As explained the packet headers come in the form of IPv4 headers, and are defined in the file *ipv4-header.h*. To change this file, it is necessary to go down in the simulator kernel and change many classes just to meet the "new" IPv4 packet. Facing this, the chosen strategy was to use a field of the source identification for placing the context information. With this, it is possible to change the lookup procedure in the *aodv-rtable.cc* and *aodv-rtable.h* files to be aware of new parameters and to implement the new lookup behaviour.

This new routing algorithm can be mapped in the simulator stack in the Routing Module as shown in the following figure:

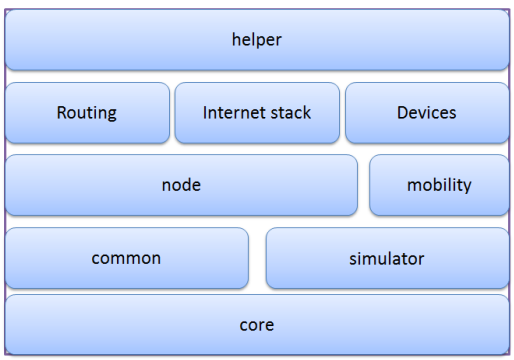
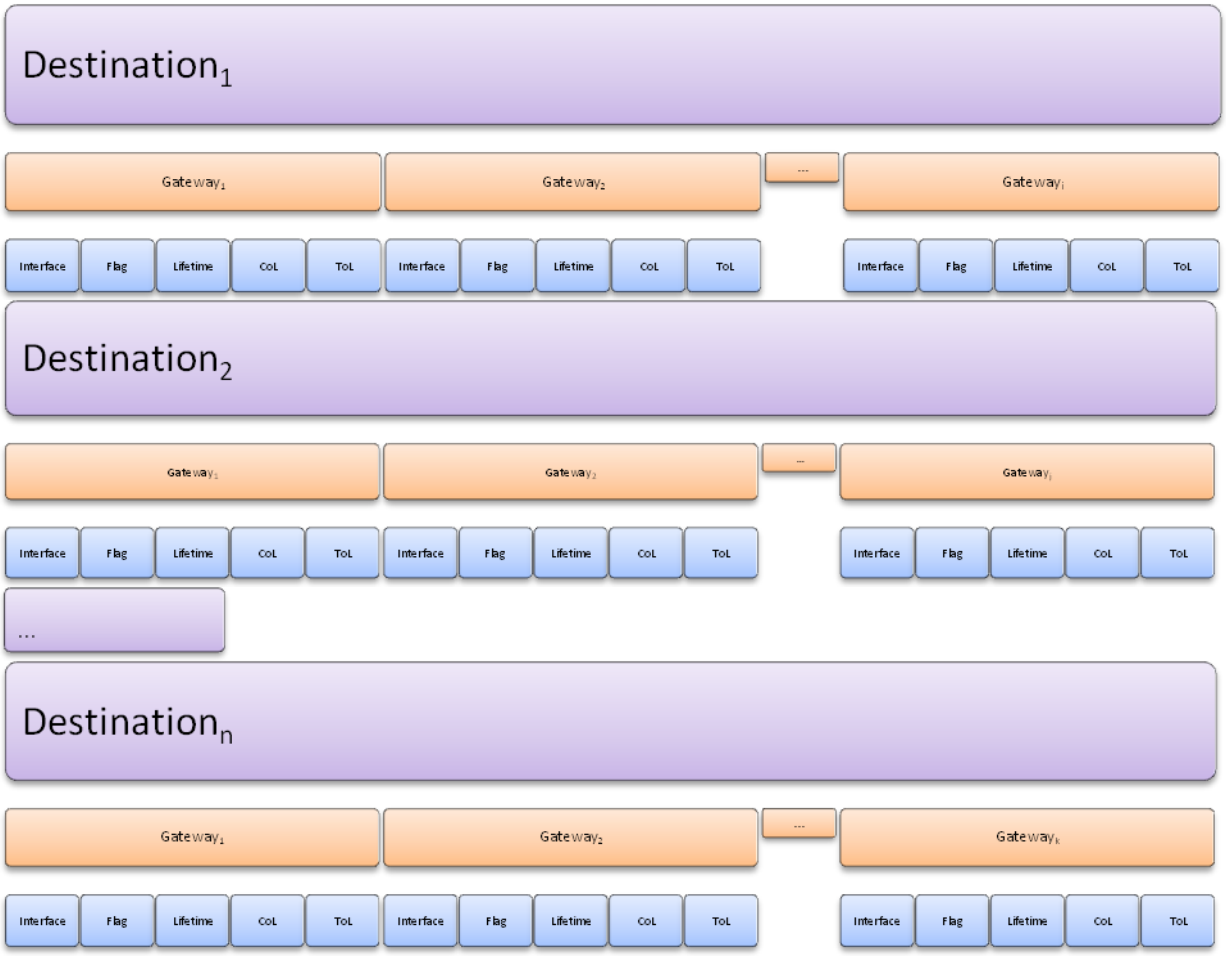


FIGURE 4.3 NS-3 MODULES

4.2.2. CHANGING NS-3 AODV IMPLEMENTATION

Here the changes corresponding to MNH part of C-AODV are explained; a known destination may have different entries in the routing table if there are different next hops (*gateway*). In order to deal with the Context, it has to include more attributes, and the changes made to accommodate this issue, were to reformulate all the data structure of the routing table. The routing table was changed to:

TABLE 4.1 THE ROUTING TABLE



The CoL and ToL values represent the Cost of Link and Time of Link, which represent the communication effort and the propagation time between hops. With this information, it is possible to have different next hops to the same destination and a routing table entry that can store information about the total cost of a link (power consumption corresponding to

CoL) and the time spent on the use of the link (time that must subtracted from the deadline indicated in ToL), as introduced in section 4.1.

Finally, and in order to implement the path accumulation with the notion of Context, the AODV RREQ and RREP packets must be changed. The files responsible for the structure and behavior of the packets are the *aodv-packet.cc* and the *aodv-packet.h*. A datastructure was created in each RREQ and RREP packet for the representation of the path accumulation. This data structure is *std::list<PathAccumulation>*, and the class *PathAccumulation* is defined in the file *PathAccumulation.cc* and *PathAccumulation.h*. These two files represent a pair of CoL/ToL to be inserted in the path accumulation of a packet.

When a node wants to forward a packet to another destination the code used is in *aodv-routing-protocol.cc*; the changes to this file correspond to: instead of incrementing the hopCount of the packet, the cost and the time of the link are updated; a new entry in the path accumulation that includes CoL and ToL is added.

```
(…)
hop = m_ContextHelper.getCostOfLink(src, receiver);
rreqHeader.SetHopCount (rreqHeader.GetHopCount () + hop);

time += m_ContextHelper.getTimeOfLink(src, receiver);
rreqHeader.SetTimeCount (rreqHeader.GetTimeCount () + time);

rreqHeader.AddToPA(receiver, hop, time);

(…)
```

The class ContexHelper is defined in the files *contextHelper.cc* and *contextHelper.h*. It only simulates how to obtain the values of cost and time between links with the methods *int getCostOfLink(Ipv4Address alice, Ipv4Address bob)* and *int getTimeOfLink(Ipv4Address alice, Ipv4Address bob)*.

When a packet is transmitted between nodes, it must be serialized for transmission and deserialized on reception. For a correct behaviour the nodes must know the size the serialized versions of RREQ and RREP packets; the size of a packet can be obtained by invoking the *RreqHeader::GetSerializedSize ()* method:

```
(…)
uint32_t
 RreqHeader::GetSerializedSize () const
 {
   int sizelist = 0;
   std::list<PathAccumulation> ll = m_pa;
   uint32_t sizeOfLinkedList = ll.size();
   for(uint32_t idx = 0; idx < sizeOfLinkedList; idx++)
   {
       sizelist += 12;
   }
   return (43 + sizelist);
 }
 (…)
```

After all these changes, the file *wscript* must be changed in order to guide the *waf* procedure to compile the new classes and all the changes made. The *wscript* file is the following:

```
## -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*-

   def build(bld):
       module = bld.create_ns3_module('aodv', ['internet-stack', 'contrib'])
       module.includes = '.'
       module.source = [
           'aodv-id-cache.cc',
           'aodv-dpd.cc',
           'aodv-rtable.cc',
           'aodv-rqueue.cc',
           'aodv-packet.cc',
           'aodv-neighbor.cc',
           'aodv-routing-protocol.cc',
           'aodv-test-suite.cc',
           'test/aodv-regression.cc',
           'test/bug-772.cc',
           'test/loopback.cc',
           'pathAccumulation.cc',
           'contextHelper.cc',
           ]

       headers = bld.new_task_gen('ns3header')
       headers.module = 'aodv'
       headers.source = [
           'aodv-id-cache.h',
           'aodv-dpd.h',
           'aodv-rtable.h',
           'aodv-rqueue.h',
           'aodv-packet.h',
           'aodv-neighbor.h',
           'aodv-routing-protocol.h',
           'pathAccumulation.h',
           'contextHelper.h',
           'linkedListPA.h',
           'linkedListRTable.h',
           ]
```

To compile the changes, it is only necessary to build the nsnam as shown before, or run a test script.

5. TESTS AND VALIDATION

The goal of performing tests and validating their results is to verify if the proposal solution satisfies the objectives previously defined. Testing is the process of searching for errors in an implementation, by running experiments in a controlled environment. These tests are used to gain maturity and confidence in the implementation, in order to use it in a real environment. Tests can prove the existence of errors in the implementation, but the inexistence of any error cannot prove that the experiment does not have errors (Tretmans, 2001). In order to perform the validation of the implementation, it is necessary to define a methodology of tests and a set of tests to be performed as a proof of concept.

Each individual test performed is presented in a separate section with the following information:

- **Setup and configuration**;
- **Results obtained**;
- **Test Conclusions**;

We chose the tests enumerated below, trying to define a separated goal to each one. The performed tests were:

- **Brief communication test**: to prove the simple communication capability of the algorithm;
- **Test in a controlled network**: the goal is to compare the performance of C-AODV with the base AODV, in a fixed setting;
- **Mobility test**: to prove the correctness of the algorithm when the network topology changes or is segmented;
- **Test for extensibility**: aims to analyse the behaviour of the algorithm against AODV when the network has a random behaviour (nodes can change position and links can fail).

The metrics used for evaluation were:

1-**Message delivery time**: the traditional *ping* command can be used;

2-**Number of lost messages**: the *ping* command also supplies this information;

3-**Cost in power to deliver a message**: in a fixed network configuration, the results of the *ping* command can be used to calculate this metric; in situations when the route between the nodes can change, the *ping* command must be extended for supplying this information; this modification was not possible;

4-**Number of messages that could not be delivered in time**: post processing of the *ping* results allows the calculation of these values.

As explained above, all the tests were made running the **ping** (*V4Ping*) application existent in the NS-3 simulator. According to the official NS-3 documentation, it is "*an application which sends one ICMP ECHO request, waits for a REPLYs and reports the calculated RTT.*". Its inheritance diagram is the following:

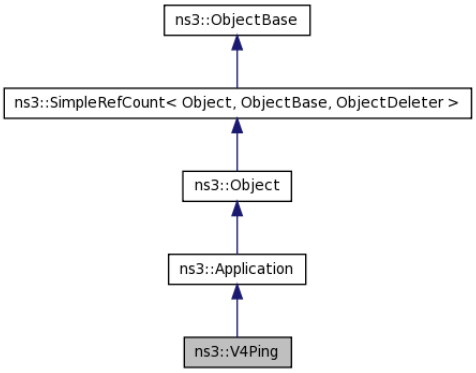


FIGURE 5.1 INHERITANCE DIAGRAM OF V4PING

Each test corresponds to the running of C++ program ("script" in NS-3 terminology) that invokes methods for:

- Creating the required number of nodes and defining its characteristics;
- Establishing the topology needed; this includes the location of the node, its behaviour (fixed or mobile); if a node is mobile initial and final positions as well as speed must be defined;
- Creating and instance of *ping* command (implemented in the *V4PingHelper* class) in the source node, in the invocation the number ICMP ECHO requests sent and the waiting time between them is defined.

5.1. BRIEF COMMUNICATION TEST

This test presents a basic communication test between nodes; the goal is to show that using the C-AODV the nodes can communicate.

5.1.1. SETUP AND CONFIGURATION

In this test, the configuration used can be found in the example file, located in *aodv.cc*. Figure 5.2 presents a flat topology where the nodes are distributed in a 1-dimensional grid and the first node tries to communicate with the last node with the **ping** application; in the beginning Node 1 does not know the route to Node 6.

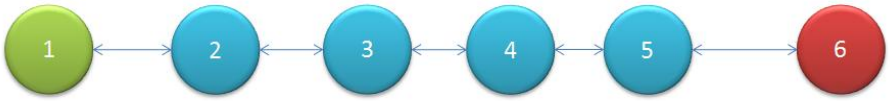


FIGURE 5.2 BRIEF COMMUNICATION TOPOLOGY

The script for performing the test was built according to the steps described before. The script defined the distance between the nodes and the total time of the test.

5.1.2. RESULTS OBTAINED

The results obtained were the following:

/ C-AODV */* *Creating 6 nodes 150 m apart.* *Starting simulation for 20 s ...* *PING 11.1.1.6 56(84) bytes of data.* *64 bytes from 11.1.1.6: icmp_seq=0 ttl=60 time=16 ms* *64 bytes from 11.1.1.6: icmp_seq=1 ttl=60 time=5 ms* *64 bytes from 11.1.1.6: icmp_seq=2 ttl=60 time=5 ms* *64 bytes from 11.1.1.6: icmp_seq=3 ttl=60 time=4 ms* *64 bytes from 11.1.1.6: icmp_seq=4 ttl=60 time=6 ms* *64 bytes from 11.1.1.6: icmp_seq=5 ttl=60 time=5 ms* *64 bytes from 11.1.1.6: icmp_seq=6 ttl=60 time=5 ms* *--- 11.1.1.6 ping statistics ---* *20 packets transmitted, 7 received, 65% packet loss, time 19999ms* *rtt min/avg/max/mdev = 4/6.571/16/4.198 ms*	*/* AODV */* *Creating 6 nodes 150 m apart.* *Starting simulation for 20 s ...* *PING 11.1.1.6 56(84) bytes of data.* *64 bytes from 11.1.1.6: icmp_seq=0 ttl=60 time=15 ms* *64 bytes from 11.1.1.6: icmp_seq=1 ttl=60 time=5 ms* *64 bytes from 11.1.1.6: icmp_seq=2 ttl=60 time=5 ms* *64 bytes from 11.1.1.6: icmp_seq=3 ttl=60 time=4 ms* *64 bytes from 11.1.1.6: icmp_seq=4 ttl=60 time=4 ms* *64 bytes from 11.1.1.6: icmp_seq=5 ttl=60 time=4 ms* *64 bytes from 11.1.1.6: icmp_seq=6 ttl=60 time=4 ms* *--- 11.1.1.6 ping statistics ---* *20 packets transmitted, 7 received, 65% packet loss, time 19999ms* *rtt min/avg/max/mdev = 4/5.857/15/4.059 ms*

In the end of the test, the routing table of Node 2 was:

/ ***C-AODV***
Routing table of 11.0.1.2
**/*
AODV Routing table:

Destination	*Gateway*	*Interface*	*Flag*	*Expire*	*Cost*	*Time*
0.0.0.61	*11.1.1.3*	*11.1.1.2*	*UP*	*0.8*	*60*	*570*
11.1.1.1	*11.1.1.1*	*11.1.1.2*	*UP*	*5.59446*	*0*	*0*
11.1.1.3	*11.1.1.3*	*11.1.1.2*	*UP*	*0.8*	*60*	*570*
11.1.1.4	*11.1.1.3*	*11.1.1.2*	*UP*	*0.8*	*60*	*570*
11.1.1.5	*11.1.1.3*	*11.1.1.2*	*UP*	*0.8*	*60*	*570*
11.1.1.255	*11.1.1.255*	*11.1.1.2*	*UP*	*9.22337e+09*	*1*	*1*
127.0.0.1	*127.0.0.1*	*127.0.0.1*	*UP*	*9.22337e+09*	*1*	*1*

For comparison purposes, the routing table for the same node using the AODV algorithm is also presented:

/ ***AODV***
Routing table of 11.0.1.2
**/*
AODV Routing table:

Destination	*Gateway*	*Interface*	*Flag*	*Expire*	*Hops*
11.1.1.1	*11.1.1.1*	*11.1.1.2*	*UP*	*6.0085*	*1*
11.1.1.3	*11.1.1.3*	*11.1.1.2*	*UP*	*6.99954*	*1*
11.1.1.6	*11.1.1.3*	*11.1.1.2*	*DOWN*	*3.0085*	*4*
11.1.1.255	*11.1.1.255*	*11.1.1.2*	*UP*	*9.22337e+09*	*1*
127.0.0.1	*127.0.0.1*	*127.0.0.1*	*UP*	*9.22337e+09*	*1*

5.1.3. TEST CONCLUSIONS

Analyzing the results obtained by the output of the application, we can conclude:

- C-AODV works correctly;
- C-AODV shows slightly worst values than plain AODV. This may occur because the complexity of processing the algorithm and the structures used; while in AODV the routing table lookup is direct, the C-AODV has to navigate through a data structure and retrieve the value, and it has to do the same to insert new values existing in the path accumulation of messages.

The output of the routing tables of node *11.1.0.2* was chosen because it is an intermediate node and so, it was crossed by many control messages. The routing table of the AODV algorithm was as expected and very similar to the one for C-AODV. Despite some erroneous entries, like *0.0.0.61*, which will only contribute for to the increase of the routing table, it is possible to see that its routing table is bigger than the equivalent for AODV. The path accumulation component of the algorithm worked as expected.

5.2. TEST IN A CONTROLLED NETWORK

This test uses a fixed configuration with the goal of comparing C-AODV and AODV performance.

5.2.1. SETUP AND CONFIGURATION

The network topology used in this test is the following:

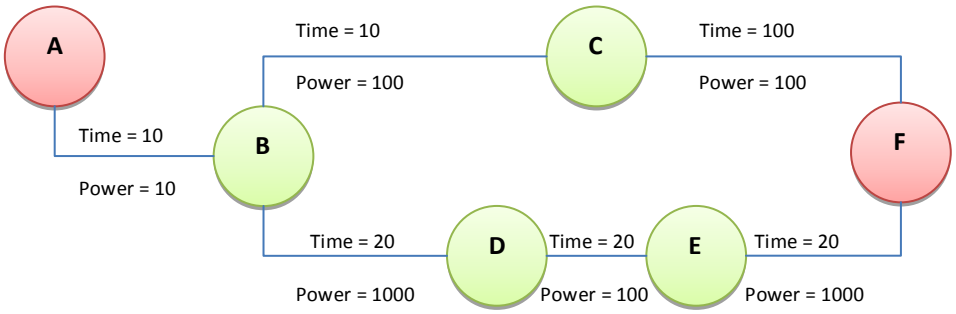


FIGURE 5.3 SIMPLE TEST CONFIGURATION

The values of Time and Power are the ones used in C-AODV for context information. These values were previously defined in the file *contextHelper.cc*.

In order to create this topology, the following script was implemented:

```
(…)
// creation of node containers for the point-to-point (p2p) pairs
  NodeContainer nodesFN;
  nodesFN.Create (2);

  NodeContainer nodesNA;
  nodesNA.Create (2);

  NodeContainer nodesAB;
  nodesAB.Add (nodesNA.Get (1));
  nodesAB.Create (1);

(…)
// creation of a p2p connection
  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

// install p2p in node containers
  NetDeviceContainer devicesFN;
  devicesFN = pointToPoint.Install (nodesFN);

  NetDeviceContainer devicesNA;
  devicesNA = pointToPoint.Install (nodesNA);

  NetDeviceContainer devicesAB;
  devicesAB = pointToPoint.Install (nodesAB);

(…)
// install stack
  AodvHelper aodv;
  InternetStackHelper stack;
  stack.SetRoutingHelper (aodv);
  stack.Install (nodesNA.Get (0));
  stack.Install (nodesNA.Get (1));

(…)
// assign ip addresses
  Ipv4AddressHelper addressFN;
  addressFN.SetBase ("11.1.123.0", "255.255.255.0");
  Ipv4InterfaceContainer interfacesFN = addressFN.Assign (devicesFN);

  Ipv4AddressHelper addressNA;
  addressNA.SetBase ("11.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer interfacesNA = addressNA.Assign (devicesNA);

  Ipv4AddressHelper addressAB;
  addressAB.SetBase ("11.1.2.0", "255.255.255.0");
  Ipv4InterfaceContainer interfacesAB = addressAB.Assign (devicesAB);

(…)
// install and run Ping application
  V4PingHelper ping1 (interfacesFN.GetAddress (0));
  ping1.SetAttribute ("Verbose", BooleanValue (true));

  ApplicationContainer p1 = ping1.Install (nodesNA.Get (1));
  p1.Start (Seconds (0));
  p1.Stop (Seconds (20) - Seconds(0.001));

  V4PingHelper ping2 (interfacesBC.GetAddress (0));
  ping2.SetAttribute ("Verbose", BooleanValue (true));

  ApplicationContainer p2 = ping2.Install (nodesFN.Get (0));
  p2.Start (Seconds (40));
  p2.Stop (Seconds (60) - Seconds(0.001));
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

The node containers are necessary because the Point-to-Point connections in nsnam must have two nodes in each container. This assertion can be found in the file *point-to-point-helper.cc* and stands for the assertion to that size when installing a point-to-point connection at a container:

```
(…)
NetDeviceContainer
PointToPointHelper::Install (NodeContainer c)
{
  NS_ASSERT (c.GetN () == 2);
  return Install (c.Get (0), c.Get (1));
}
(…)
```

Therefore, the devices need to have more than one interface to communicate with other devices, which is similar to the situation were a desktop computer has multiple network cards. So, Node A will have two interfaces (*11.1.1.2* and *11.1.2.1*), Node B will have three interfaces (*11.1.2.2*, *11.1.3.1* and *11.1.5.1,* Figure 5.4), node C will have two interfaces (*11.1.3.2* and *11.1.4.1*), and so on, as presented in the script and in Figure 5.3. Finally, two instances of the **ping** application were created: the first where Node A *pings* Node F, and a second where Node F *pings* Node B. The expected result is that Node F can re-use the information of the path accumulation that travelled the first *ping*, and so, it is not necessary to send a RREQ message to discover the Node B.

FIGURE 5.4 NODE B IN CONTROLLED NETWORK

Finally, and in order to implement awareness in context information, the routing mechanism must acquire information from a message in order to infer the best route to forward it. To do this, another method was created in the *rtable.cc* and *rtable.h*. With this additional method, C-AODV can separate the lookup in the message discovery and maintenance phases, from the message forwarding. The corresponding code is presented below:

```
bool
RoutingTable::LookupRouteForMessage (Ipv4Address id, RoutingTableEntry & rt,
const Ipv4Header & header){
(…)
      unsigned int b = header.GetSource().Get();
      int d = (int)b;
      int mod = d;
      if( mod > 4 ){
 (…)
            for (; j != listOfRoutingTableEntriesFound.end (); j++)
            {
                if( (*j).GetTimes() < bestTime )
                {
                    bestTime = (*j).GetTimes();
                    rt = (*j);
                }
            }
 (…)
      }
 (…)
      for (; j != listOfRoutingTableEntriesFound.end (); j++)
      {
          if( (*j).GetHop() < bestCost )
          {
              bestCost = (*j).GetHop();
              rt = (*j);
          }
      }
      return true;
      }
(…)

bool
RoutingTable::LookupValidRouteForMessage (Ipv4Address id, RoutingTableEntry &
rt, const Ipv4Header & header){
    if (! LookupRouteForMessage (id, rt, header)){
        return false;
    }
   return (rt.GetFlag () == VALID);
}
```

Because changing a packet in the core of the simulator can introduce many dependency problems, it was decided to use the integer value that represents the packet source in the header to define the message's priority. The messages originated by the **ping** command can be:

- "normal" (without deadline specification): in this case the routing algorithm only uses the traditional cost considerations. In this case AODV and C-AODV should give similar results;
- "special" (with deadline specification): here the C-AODV uses the context information. In this type of test C-AODV should get a better rate of delivered messages in time.

For performing these tests we had to modify *aodv-routing-protocol.cc* in order to modify the lookup methods for a message; the methods *RoutingTable::LookupValidRouteForMessage* and *RoutingTable::LookupRouteForMessage* perform the routing table management.

5.2.2. RESULTS OBTAINED

The application **ping** was also used in this test, not only in order to guarantee a simple connectivity test, but because it shows some values that can be used for later processing. The routing tables for both algorithm were as follows:

/ **C-AODV***
Routing table of Node F after the discovery phase
**/*
AODV Routing table:

Destination	*Gateway*	*Interface*	*Flag*	*Expire*	*Cost*	*Time*
11.1.2.1	*11.1.4.1*	*11.1.4.2*	*UP*	*-10.4024*	*210*	*30*
11.1.3.1	*11.1.4.1*	*11.1.4.2*	*UP*	*-10.4022*	*200*	*110*
11.1.3.1	*11.1.7.1*	*11.1.7.2*	*UP*	*-162.4*	*2100*	*50*
11.1.3.2	*11.1.4.1*	*11.1.4.2*	*UP*	*-2.40236*	*100*	*10*
11.1.4.1	*11.1.4.1*	*11.1.4.2*	*UP*	*-2.4022*	*100*	*100*
11.1.4.255	*11.1.4.255*	*11.1.4.2*	*UP*	*9.22337e+09*	*1*	*1*
11.1.5.2	*11.1.7.1*	*11.1.7.2*	*UP*	*-154.4*	*2000*	*40*
11.1.6.2	*11.1.7.1*	*11.1.7.2*	*UP*	*-2.40018*	*100*	*20*
11.1.7.1	*11.1.7.1*	*11.1.7.2*	*UP*	*-74.4*	*1000*	*20*
11.1.7.255	*11.1.7.255*	*11.1.7.2*	*UP*	*9.22337e+09*	*1*	*1*
11.1.123.2	*11.1.123.2*	*11.1.123.1*	*UP*	*2.99831*	*0*	*0*
11.1.123.255	*11.1.123.255*	*11.1.123.1*	*UP*	*9.22337e+09*	*1*	*1*
127.0.0.1	*127.0.0.1*	*127.0.0.1*	*UP*	*9.22337e+09*	*1*	*1*

/ **NS3 AODV***
Routing table of Node F after the first discovery phase
**/*
AODV Routing table

Destination	*Gateway*	*Interface*	*Flag*	*Expire*	*Hops*
11.1.2.1	*11.1.4.1*	*11.1.4.2*	*UP*	*2.00772*	*3*
11.1.4.1	*11.1.4.1*	*11.1.4.2*	*UP*	*2.99741*	*1*
11.1.4.255	*11.1.4.255*	*11.1.4.2*	*UP*	*9.22337e+09*	*1*
11.1.7.1	*11.1.7.1*	*11.1.7.2*	*UP*	*2.00051*	*1*
11.1.7.255	*11.1.7.255*	*11.1.7.2*	*UP*	*9.22337e+09*	*1*
11.1.123.2	*11.1.123.2*	*11.1.123.1*	*UP*	*3*	*1*
11.1.123.255	*11.1.123.255*	*11.1.123.1*	*UP*	*9.22337e+09*	*1*
127.0.0.1	*127.0.0.1*	*127.0.0.1*	*UP*	*9.22337e+09*	*1*

The **ping** results were:

/* **C-AODV**	/* **NS3 AODV**
First ping	*First ping*
/	*/*
PING 11.1.123.1 56(84) bytes of data.	*PING 11.1.123.1 56(84) bytes of data.*
64 bytes from 11.1.123.1: icmp_seq=0 ttl=62 time=25 ms	*64 bytes from 11.1.123.1: icmp_seq=0 ttl=62 time=25 ms*
64 bytes from 11.1.123.1: icmp_seq=1 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=1 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=2 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=2 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=3 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=3 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=4 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=4 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=5 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=5 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=6 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=6 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=7 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=7 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=8 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=8 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=9 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=9 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=10 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=10 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=11 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=11 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=12 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=12 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=13 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=13 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=14 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=14 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=15 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=15 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=16 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=16 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=17 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=17 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=18 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=18 ttl=62 time=12 ms*
64 bytes from 11.1.123.1: icmp_seq=19 ttl=62 time=12 ms	*64 bytes from 11.1.123.1: icmp_seq=19 ttl=62 time=12 ms*
--- 11.1.123.1 ping statistics ---	*--- 11.1.123.1 ping statistics ---*
20 packets transmitted, 20 received, 0% packet loss, time 19999ms	*20 packets transmitted, 20 received, 0% packet loss, time 19999ms*
rtt min/avg/max/mdev = 12/12.65/25/2.907 ms	*rtt min/avg/max/mdev = 12/12.65/25/2.907 ms*

5.2.3. TEST CONCLUSIONS

With this test proved that the C-AODV can use context information. The following table compares the context information in Node F in the end of simulation with the expected ones (the erroneous values (*red*) are shown against the correct ones (*green*)):

TABLE 5.1 CONTROLED TEST CONCLUSIONS

Node F	*Cost*	*Time*	*Cost (expected)*	*Time (expected)*
11.1.2.1	**210**	30	**210**	70
11.1.3.1 (from Node C)	**200**	**110**	**200**	**110**
11.1.3.1 (from Node E)	**2100**	50	**2100**	60
11.1.3.2	**100**	10	**100**	100
11.1.4.1	**100**	**100**	**100**	**100**
11.1.5.2	2000	**40**	1100	**40**
11.1.7.1	1000	**20**	100	**20**

As mentioned before, the context values were defined previously; in order to set the desired values for cost and time; as the algorithm is not responsible for its calculations, it only has to retrieve them from the context mechanism and apply them in the routing tables. All the values were well inserted, however some values are wrong despite the correct differences between paths, which could still lead to the correct solution.

The paths "seen" by the algorithm for the Node B were two with the respective <CoL, ToL> pairs: <2100, 60> and <200, 100>, and as the message was flagged with priority, the algorithm choose the path that starts from Node E aside from Node C, as expected. The behavior of AODV base, was the expected because it does not take consider the context information; it also does not use multiple hops to a same destination.

This test was also useful for presenting the importance of the path accumulation existent in the algorithm. When trying to find node B, C-AODV did not use the flooding procedure while AODV did it; the AODV algorithm in Node F had to calculate the route to Node B, performing another discovery phase which generated the following routing table:

/ ***NS3 AODV***
Routing table of Node F after the second discovery phase
**/*
AODV Routing table

Destination	*Gateway*	*Interface*	*Flag*	*Expire*	*Hops*
11.1.3.1	*11.1.4.1*	*11.1.4.2*	*UP*	*2.01139*	*2*
11.1.4.1	*11.1.4.1*	*11.1.4.2*	*UP*	*2.97865*	*1*
11.1.4.255	*11.1.4.255*	*11.1.4.2*	*UP*	*9.22337e+09*	*1*
11.1.7.1	*11.1.7.1*	*11.1.7.2*	*UP*	*9.99744*	*1*
11.1.7.255	*11.1.7.255*	*11.1.7.2*	*UP*	*9.22337e+09*	*1*
11.1.123.2	*11.1.123.2*	*11.1.123.1*	*UP*	*3*	*1*
11.1.123.255	*11.1.123.255*	*11.1.123.1*	*UP*	*9.22337e+09*	*1*
127.0.0.1	*127.0.0.1*	*127.0.0.1*	*UP*	*9.22337e+09*	*1*

This proves that AODV does not take in consideration the multiple hops to a given destination nor the context information. Routing the packets through Node C would have less power consumption, but would also waste more time in communication, which is not desirable to the proposed message priority. AODV cannot adapt itself to context information, while the C-AODV can.

Another important feature shown is that this protocol does not introduce load in the network, because the outputs from both AODV and C-AODV are exactly the same.

The following table makes an overview of the advantages of C-AODV over AODV:

TABLE 5.2 COMPARISON IN CONTROLLED NETWORK

	Multiple routes	*Path accumulation*	*Save network flooding*	*Context aware*	*Ad hoc*	*Light performance*
AODV					•	•
C-AODV	•	•	•	•	•	•

5.3. MOBILITY TEST

In this test, one or more nodes can change their location; this change can destroy and create connections with neighbours. This is very important in order to evaluate the conformance of the algorithm facing a network that can change its topology.

5.3.1. SETUP AND CONFIGURATION

The first procedure was to create the usual script that deploy a certain number of nodes according to a known topology, and then change a certain node in order to evaluate the behavior of the algorithm.

The script created is very similar to the script in the **Brief communication test**, however it was necessary to schedule a task to change the node's position:

```
(…)
   static void
   CallChanges(TestMobility *model)
   {
     model->ChangePos();
   }

   int main (int argc, char **argv)
   {
(…)
     Simulator::Schedule(Seconds(5.0), &CallChanges, &testMob );
(…)
   }

(…)
   void
   AodvExample::ChangePos()
   {
     Ptr<Node> node = nodes.Get(size - 1);
     Ptr<MobilityModel> mob = node->GetObject<MobilityModel>();
     Vector pos = mob->GetPosition();
     Vector newPos = Vector(2000.0,0.0,0.0);
     mob->SetPosition(newPos);

     node = nodes.Get(0);
     mob = node->GetObject<MobilityModel>();
     pos = mob->GetPosition();
     newPos = Vector(2030.0,0.0,0.0);
     mob->SetPosition(newPos);
   }
(…)
```

With this configuration, after five seconds, the last node changes its position from (750, 0, 0) to (2000, 0, 0) and the first node from (0, 0, 0) to (2030, 0, 0). This configuration was chosen because the last node and the first node can change their position and communicate without the influence of the other nodes. A more schematic presentation is presented next:

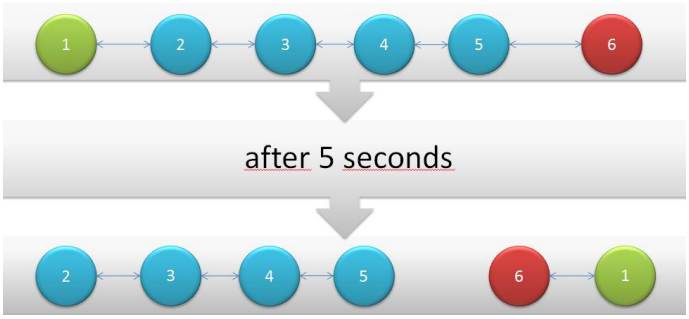


FIGURE 5.5 MOBILITY TEST CHANGES

5.3.2. RESULTS OBTAINED

The **ping** results were the following:

```
/* C-AODV
*/
Creating 6 nodes 150 m apart.
Starting simulation for 30 s ...
PING  11.1.1.6 56(84) bytes of data.
64 bytes from 11.1.1.6: icmp_seq=0 ttl=60 time=16 ms
64 bytes from 11.1.1.6: icmp_seq=1 ttl=60 time=5 ms
64 bytes from 11.1.1.6: icmp_seq=2 ttl=60 time=5 ms
64 bytes from 11.1.1.6: icmp_seq=3 ttl=60 time=4 ms
64 bytes from 11.1.1.6: icmp_seq=4 ttl=60 time=6 ms
                (changing nodes position)
64 bytes from 11.1.1.6: icmp_seq=6 ttl=60 time=1 ms
64 bytes from 11.1.1.6: icmp_seq=7 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=8 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=9 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=10 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=11 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=12 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=13 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=14 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=15 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=16 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=17 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=18 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=19 ttl=60 time=0 ms
--- 11.1.1.6 ping statistics ---
20 packets transmitted, 19 received, 5% packet loss, time
19999ms
rtt min/avg/max/mdev = 0/1.947/16/3.993 ms
```

```
/* AODV
*/
Creating 6 nodes 150 m apart.
Starting simulation for 30 s ...
PING  11.1.1.6 56(84) bytes of data.
64 bytes from 11.1.1.6: icmp_seq=0 ttl=60 time=15 ms
64 bytes from 11.1.1.6: icmp_seq=1 ttl=60 time=5 ms
64 bytes from 11.1.1.6: icmp_seq=2 ttl=60 time=5 ms
64 bytes from 11.1.1.6: icmp_seq=3 ttl=60 time=4 ms
64 bytes from 11.1.1.6: icmp_seq=4 ttl=60 time=4 ms
                (changing nodes position)
64 bytes from 11.1.1.6: icmp_seq=6 ttl=60 time=1 ms
64 bytes from 11.1.1.6: icmp_seq=7 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=8 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=9 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=10 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=11 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=12 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=13 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=14 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=15 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=16 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=17 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=18 ttl=60 time=0 ms
64 bytes from 11.1.1.6: icmp_seq=19 ttl=60 time=0 ms
--- 11.1.1.6 ping statistics ---
20 packets transmitted, 19 received, 5% packet loss, time
19999ms
rtt min/avg/max/mdev = 0/1.789/15/3.706 ms
```

And the corresponding routing tables:

```
/* C-AODV
*/
AODV Routing table:
```

Destination	Gateway	Interface	Flag	Expire	Cost	Time
0.0.0.61	11.1.1.2	11.1.1.1	UP	-18.1926	60	570
11.1.1.2	11.1.1.2	11.1.1.1	UP	-11	0	0
11.1.1.3	11.1.1.2	11.1.1.1	UP	-18.1926	60	570
11.1.1.4	11.1.1.2	11.1.1.1	UP	-18.1926	60	570
11.1.1.5	11.1.1.2	11.1.1.1	UP	-18.1926	60	570
11.1.1.6	11.1.1.6	11.1.1.1	UP	2.08074	0	0
11.1.1.255	11.1.1.255	11.1.1.1	UP	9.22337e+09	1	1
127.0.0.1	127.0.0.1	127.0.0.1	UP	9.22337e+09	1	1

```
/* AODV
*/
AODV Routing table:
```

Destination	Gateway	Interface	Flag	Expire	Hops
11.1.1.2	11.1.1.2	11.1.1.1	DOWN	0.930417	1
11.1.1.6	11.1.1.6	11.1.1.1	UP	2.92068	1
11.1.1.255	11.1.1.255	11.1.1.1	UP	9.22337e+09	1
127.0.0.1	127.0.0.1	127.0.0.1	UP	9.22337e+09	1

5.3.3. TEST CONCLUSIONS

With this test, it was possible to conclude that the proposed algorithm could adapt itself with the same behaviour as the basic AODV. This is desirable due to the simple configuration

of the network and the mobility model. The issue that not all entries in the routing tables of the changing nodes in C-AODV were correct is still present, however, the test was successful and it was possible to prove that the algorithm can work under changes in the network topology.

5.4. TEST FOR EXTENSIBILITY

This test aims to present the behaviour of the algorithm in a dynamic network, with *n* nodes that may change their locations during time in a random way. In order to create this environment, a different script was created. Its explanation is giving the following.

5.4.1. SETUP AND CONFIGURATION

The NS-3 simulator implements some models in order to simulate mobility. The simulation consists in deploying a number of nodes in a confined area and allowing these nodes to "move" to a random position, with random velocity. This model is called *ns3::RandomWalk2dMobilityModel* and, as shown in the documentation of the simulator, "*each instance moves with a speed and direction chosen at random with the user-provided random variables until either a fixed distance has been walked or until a fixed amount of time. If we hit one of the boundaries (specified by a rectangle), of the model, we rebound on the boundary with a reflexive angle and speed. This model is often identified as a brownian motion model.*" This Brownian model is a process inspired in random movement of small particles suspended in a fluid, studied by Robert Brown in 1827 (Chang, 1999). This "random" process was tested and has always has the same behaviour, which allows the reproduction of tests.

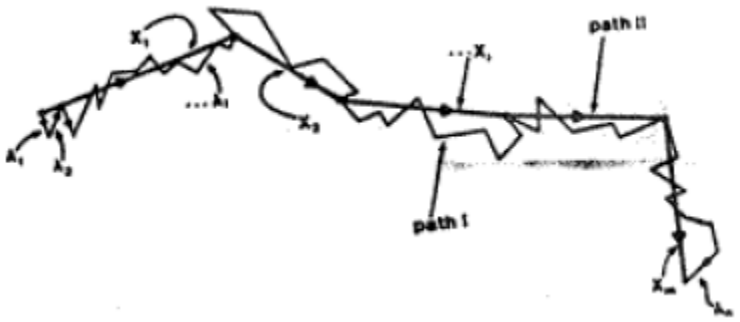


FIGURE 5.6 BROWNIAN MOVEMENT IN TWO PATHS (COHEN, 1986)

The script for this test consists in the definition of a given number of wireless devices; this value was changed in order to present different test results and after this, they were deployed in a grid were can move around. According to the official documentation of NS-3, the grid has the following parameters:

- ***GridWidth***: The number of objects layed out on a line;
- ***MinX***: The x coordinate where the grid starts;
- ***MinY***: The y coordinate where the grid starts;
- ***DeltaX***: The x space between objects;
- ***DeltaY***: The y space between objects;
- ***LayoutType***: The type of layout. Row-oriented or column-oriented.

In preliminary tests the influence of these parameters in the behaviour of network was tested. The conclusion was that the variation of the *GridWidth* parameter implied more changes in the behaviour of the network. According to this, *GridWidth* was chosen as a parameter in the tests, along with the number of nodes and ping running time, since the **ping** application was used again.

A part of the implemented script is presented next:

```
(…)
  int nWifi = 5;
  bool verbose = true;
  int totalTime = 10;
  int gridWidth = 3;

  (…)
  NodeContainer wifiStaNodes;
  wifiStaNodes.Create (nWifi);

  NodeContainer wifiApNode;
  wifiApNode.Create(1);

  YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
  YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
  phy.SetChannel (channel.Create ());

  WifiHelper wifi = WifiHelper::Default ();
  wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

  NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();

  Ssid ssid = Ssid ("ns-3-ssid");
  mac.SetType ("ns3::NqstaWifiMac",
    "Ssid", SsidValue (ssid),
    "ActiveProbing", BooleanValue (false));

  NetDeviceContainer staDevices;
  staDevices = wifi.Install (phy, mac, wifiStaNodes);

(…)
  MobilityHelper mobility;
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
    "MinX", DoubleValue (0.0),
    "MinY", DoubleValue (0.0),
    "DeltaX", DoubleValue (5.0),
    "DeltaY", DoubleValue (10.0),
    "GridWidth", UintegerValue (gridWidth),
    "LayoutType", StringValue ("RowFirst"));

  mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
    "Bounds", RectangleValue (Rectangle (-5000, 5000, -5000, 5000)));
  mobility.Install (wifiStaNodes);

  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
  mobility.Install (wifiApNode);

  AodvHelper aodv;
  InternetStackHelper stack;
  stack.SetRoutingHelper (aodv);
  stack.Install (wifiApNode);
  stack.Install (wifiStaNodes);

  Ipv4AddressHelper address;

  address.SetBase ("11.1.0.0", "255.255.0.0");
  Ipv4InterfaceContainer wifiInterfaces;
  wifiInterfaces = address.Assign (staDevices);
  address.Assign (apDevices);


  V4PingHelper ping1 (wifiInterfaces.GetAddress ( 0 ));
  ping1.SetAttribute ("Verbose", BooleanValue (true));

  ApplicationContainer p1 = ping1.Install (wifiStaNodes.Get ( nWifi - 1 ));
  p1.Start (Seconds (0));
  p1.Stop (Seconds (totalTime) - Seconds(0.001));
 (…)
```

In order to simulate some context information, the *ContextHelper* class was changed to create different values for different communication costs for a given pair of nodes. To do this in an uncontrolled network, the integer values of both identifiers were added and then the module operator of 2 was applied, as shown below:

```
int ContextHelper::getCostOfLink (Ipv4Address alice, Ipv4Address bob)
{
   if ( (alice.Get() + bob.Get()) % 2 == 0 ){
      return 10;
   }
   return 1000;
}
```

The same behaviour was used for the *ContextHelper::getTimeOfLink (Ipv4Address alice, Ipv4Address bob)* method, but the values used were 10 and 100, respectively. The changes introduced in this way were not very. A good option would be to use the Euclidean distance between two nodes, but the methods to retrieve a node location are only accessible at a high level layer and the simulator does not allow them to be used in the routing layer.

In this test, it is difficult to evaluate the joint effect of changes in context information, number of hops and path accumulation. Therefore, a set of test were made, in order to create some tables and draw some conclusions. These are the tests with their respective parameters:

TABLE 5.3 PARAMETERS FOR THE TEST

#test	No of nodes	Time	Gridwidth	*#test*	No of nodes	Time	Gridwidth
01	10	20	3	*13*	10	20	10
02	20	20	3	*14*	20	20	10
03	50	20	3	*15*	50	20	10
04	60	20	3	*16*	60	20	10
05	10	50	3	*17*	10	50	10
06	20	50	3	*18*	20	50	10
07	50	50	3	*19*	50	50	10
08	60	50	3	*20*	60	50	10
09	10	100	3	*21*	10	100	10
10	20	100	3	*22*	20	100	10
11	50	100	3	*23*	50	100	10
12	60	100	3	*24*	60	100	10

To perform all these tests, a shell script was created due to the high number of tests and their durations. This script is called *MobilityTestScript.sh* and a part of the script is as follows:

```
#!/bin/bash
echo "start 1"
START=$(date +%s)
./waf --run 'aodvtest --nWifi=10 --totalTime=20 --gridWidth=3' > res/aodv10-
20-3.out
END=$(date +%s)
DIFF=$(($END - $START ))
echo "it took $DIFF seconds"
(…)
echo "start 24"
START=(date +%s)
./waf --run 'aodvtest --nWifi=60 --totalTime=100 --gridWidth=10' >
res/aodv60-100-10.out
(…)
```

Finally, for the given script, the node accessed by the method *wifiInterfaces.GetAddress (0)* is at the coordinates (0,0) and the last, for the configuration "*--nWifi=10 --totalTime=20 --gridWidth=3*" is at the coordinates (5,60). This configuration could be tested by inserting the following code in the script:

```
Ptr<Node> node = wifiStaNodes.Get ( nWifi - 1);
Ptr<MobilityModel> mob = node->GetObject<MobilityModel> ();
Vector pos = mob->GetPosition ();
std::cout << pos.x << ", " << pos.y << "\n";

node = wifiStaNodes.Get ( 0 );
mob = node->GetObject<MobilityModel> ();
pos = mob->GetPosition ();
std::cout << pos.x << ", " << pos.y << "\n";
```

5.4.2. RESULTS OBTAINED

The set of all tests for both algorithms originated the following table:

TABLE 5.4 EXTENSIBILITY RESULTS

Parameters			% loss	
Nodes No	Ping time	*gridWidth*	AODV	C-AODV
10	20	3	0	0
10	20	10	0	0
10	50	3	0	0
10	50	10	0	0
10	100	3	0	0
10	100	10	0	0
20	20	3	0	0
20	20	10	0	0
20	50	3	0	0
20	50	10	0	0
20	100	3	0	0
20	100	10	0	0
50	20	3	0	0
50	20	10	0	0
50	50	3	0	0
50	50	10	0	0
50	100	3	1	0
50	100	10	0	0
60	20	3	100	100
60	20	10	0	0
60	50	3	100	100
60	50	10	0	0
60	100	3	100	100
60	100	10	0	0

5.4.3. TEST CONCLUSIONS

This test accessed the basic functionally of the proposed solution in a dynamic network where nodes can change their location, thus introducing mobility and size concerns. Changing some parameters of the simulation it was possible to obtain some conclusions and compare them to the implementation of the AODV algorithm in NS3. The C-AODV shows the same results in packet delivery rate as AODV for the same changes in the parameters, which allow us to conclude that C-AODV does not introduce significant overhead in the network with the tested properties. The context information could only tested in a simple way, and unfortunately, it was not very accurate to reach reliable conclusions, but following the

previous tests, the C-AODV will always search for the best route for a priority message.

5.5. VERDICT AND SOLUTION CONFORMANCE

The results obtained in all tests show that the proposed algorithm can operate in an ad-hoc network, like the AODV algorithm. Despite some erroneous values explained by the initial phase of the algorithm, it was possible to prove that this is a distance vector algorithm for ad hoc networks, that can meet all the issues presented in chapter 1.2.

All the tests were performed and compared to the results obtained by the AODV basic algorithm, however with the Test for extensibility, it was more difficult to evaluate the C-AODV because AODV base does not use any context information nor multiple routes to the same destination.

Tests 5.2 and 5.3 were the most important because they prove the true nature of the C-AODV, its reactive nature based on AODV, its network topology awareness, with AODV-PA and MNH and finally, the concept of Context introduced in the routing mechanism, that enables a more intelligent routing behaviour in order to provide a better quality of service in the network.

This initial propose of the solution meets the expectations because it is only a primary approach to the algorithm and not a final solution for education or industrial usage.

6. CONCLUSIONS AND FUTURE WORK

With this dissertation, it was possible to obtain a better understanding of routing algorithms, more specifically, the reactive routing algorithms and the variants of the AODV algorithm, such as AODV-PA and MNH.

After the study above it was possible to propose C-AODV, a new routing algorithm based on AODV-PA and MNH. The main novelty was the introduction of the concept of Context, which can bring environmental information to the routing decisions. C-AODV was tested using the NS-3 simulator. Although, the lack of time prevented us from conducting extensive tests, the data obtained in the simulations suggest that C-AODV can perform well in dynamic situations, without significant performance overheads.

With the limitations associated to the lack of more extensive tests we can claim that AODV has the characteristics that were required in the analysis of requirements made in chapter 1.

The future development around this work includes three different dimensions:

- Better implementation of C-AODV: one of the already planned changes is the modification of the mechanism that controls duplicate RREQ and RREP messages in the algorithm. After this, for example, in test 5.2, it will be possible after the first route discovery, for the request node to know all the multiple routes to the destination. Another suggestion regards in the lookup procedure for getting context information; if the message does not have priority, the algorithm could search for the best cost and in the case of a tie, the best time;
- More extensive experimentation using NS-3, or other simulator: tests using NS-3 take a long time and different simulations can help the location of the weak and strong characteristics of C-AODV. Experiments could also identify the situations where C-AODV works well and also the ones where its use should be regarded with caution;
- Tests in real environments using a test-bed; this could be done in the context of CuteLoop.

7. BIBLIOGRAPHY

30. The Internet Of Things - TIME's Best Inventions of 2008 - TIME. (2010). Retrieved February 2010, 3, from 30. The Internet Of Things - TIME's Best Inventions of 2008 - TIME: http://www.time.com/time/specials/packages/article/0,28804,1852747_1854195_1854158,00.html

CuteLoop - Customer in the Loop. (2010). Retrieved January 24, 2010, from www.cuteloop.eu

J-Sim Official. (2010). Retrieved June 23, 2010, from http://sites.google.com/site/jsimofficial/

The Network Simulator - ns-2. (2010). Retrieved June 23, 2010, from http://www.isi.edu/nsnam/ns/

The ns-3 network simulator. (2010). Retrieved June 23, 2010, from http://www.nsnam.org/

Abolhasan, M. (2003). A review of routing protocols for mobile ad hoc networks.

Aggelou, G., & Tafazolli, R. (1999). RDMAR: a bandwidth-efficient routing protocol for mobile ad hoc networks. *ACM International Workshop on Wireless Mobile Multimedia (WoWMoM)*, 26–33.

Ay, F. (2007). Context Modeling and Reasoning using Ontologies.

Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks.

Barabási, A.-L., & Bonabeau, E. (2003). ScaleFree Networks. *Scientific American*.

Basagni, S., Chlamtac, I., Syrotivk, V., & Woodward, B. (1998). A distance effect algorithm for mobility (DREAM).

Bellur, B., Ogier, R., & Templin, F. (2003). Topology broadcast based on reverse-path forwarding routing protocol (tbrpf). *Internet Draft, draft-ietf-manet-tbrpf-06.txt*.

Bouhorma, M., Bentaouit, H., & Boudhir, A. (2009). Performance comparison of ad-hoc routing protocols AODV and DSR. *International Conference on Multimedia Computing and System*.

Chang, J. (1999). *Stochastic Processes.*

Chen, T.-W., & Gerla, M. (1998). Global state routing: a new routing scheme for ad-hoc wireless networks. *Proceedings of the IEEE ICC*.

Chiang, C.-C. (1997). Routing in clustered multihop mobile wireless networks with fading channel. *Proceedings of IEEE SICON*, 197–211.

Cohen, R. (1986). Self Similarity in Brownian Motion and Other Ergodic Phenomena. *Journal of Chemical Education 63*, 933–934.

Community, O. (2010). *OMNeT++ Community Site*. Retrieved June 23, 2010, from http://www.omnetpp.org/

Community, O. (2010). *OMNeT++ Community Site*. Retrieved June 23, 2010, from http://www.omnetpp.org/

Corson, M., & Ephremides, A. (1995). A distributed routing algorithm for mobile wireless networks. *ACM/Baltzer Wireless Networks*, 61–81.

Corson, S., & Macker, J. (1999, January). Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. *IETF RFC 2501*.

Das, S., Perkins, C., & Royer, E. (2002). Ad hoc on demand distance vector (AODV) routing. *Internet Draft, draft-ietf-manetaodv-11.txt*.

Das, S., Wu, Y., Chandra, R., & Charlie Hu, Y. (2008). Context-based Routing: Techniques, Applications and Experience. *USENIX Symposium on Networked Systems Design and Implementation (NSDI '08).* San Frascisco.

Deb, D., Roy, S. B., & Chaki, N. (2009). LACBER: A new location aided routing protocol for GPS scarce MANET. *International Journal of Wireless & Mobile Networks (IJWMN)*.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematlk 1*, 269–271.

Dube, R., Rais, C., Wang, K., & Tripathi, S. (1997). Signal stability based adaptive routing (ssa) for ad hoc mobile networks. *IEEE Personal Communication*, 36–45.

El-Ansary, S., & Haridi, S. (2005). An Overview of Structured Overlay Networks. *Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless and Peer-to-Peer Networks*.

European Commission. (2010). *Vision and Challenges for Realising the Internet of Things.* CERP-IoT: Cluster of European Research Projects on the Internet of Things.

European Parliament, t. C. (2008). *Future networks and the internet.*

FIA. (2009, May 6). *Position Paper - RwiWiki*. Retrieved February 2, 2010, from Position Paper - RwiWiki: http://rwi.future-internet.eu/index.php/Position_Paper

Garcia-Luna-Aceves, J., & Spohn, C. (1999). Source-tree routing in wireless networks. *Proceedings of the Seventh Annual International Conference on Network Protocols*, 273.

Garcia-Luna-Aceves, S. M. (1995). A routing protocol for packet radio networks. *Proceedings of the First Annual ACM International Conference on Mobile Computing and Networking*, 86–95.

Gerla, M. (2002). Fisheye state routing protocol (FSR) for ad hoc networks. *Internet Draft, draft-ietf-manet-aodv-03.txt*.

Gowrishankar, S., Sarkar, S., & Basavaraju, T. (2009). Performance analysis of AODV, AODVUU, AOMDV and RAODV over IEEE 802.15.4 in wireless sensor networks. *2nd IEEE International Conference on Computer Science and Information Technology*.

Günes, M., Sorges, U., & Bouazizi, I. (2002). Ara– The ant-colony based routing algorithm for manets. *ICPP workshop on Ad Hoc Networks (IWAHN 2002)*, 79–85.

Günes, M., Sorges, U., & Bouazizi, I. (2002). Ara—the ant-colony based routing algorithm for manets. *ICPP workshop on Ad Hoc Networks (IWAHN 2002)*, 79–85.

Gwalani, S., Belding-Royer, E., & Perkins, C. (2003). AODV-PA: AODV with path accumulation. *IEEE International Conference on Communications*.

Hashim, R., Nasir, Q., & Harous, S. (2006). Adaptive Multi-path QoS Aware Dynamic Source Routing Protocol for Mobile Ad-Hoc Network. *Innovations in Information Technology IIT2006*, 1-5.

Hass, Z., & Pearlman, R. (1999). Zone routing protocol for ad-hoc networks. *Internet Draft, draft-ietf-manet-zrp-02.txt*.

Jacquet, P., Muhlethaler, P., Clausen, T., Laouiti, A., Qayyum, A., & Viennot, L. (2001). Optimized link state routing protocol for ad hoc networks. *IEEE INMIC*.

Jiang, M., & Jan, R. (2001). An Efficient Multiple Paths Routing Protocol for Ad-hoc Networks. *Proc. of the 15th International Conference on Information Networking*, 544-549.

Jiang, M., Ji, J., & Tay, Y. (1999). Cluster based routing protocol. *Internet Draft, draft-ietf-manet-cbrp-spec-01.txt*.

Joa-Ng, M., & Lu, I.-T. (1999). A peer-to-peer zone-based two-level link state routing for mobile ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 1415–1425.

Johnson, D., Maltz, D., & Hu, Y.-C. (2004). The dynamic source routing protocol for mobile ad hoc networks. *Internet Draft, draft-ietf-manet-dsr-10.txt*.

Johnson, D., Maltz, D., & Jetcheva, J. (2002). The dynamic source routing protocol for mobile ad hoc networks. *Internet Draft, draft-ietf-manet-dsr-07.txt*.

Kasera, K., & Ramanathan, R. (1997). A location management protocol for hierarchically organised multihop mobile wireless networks. 158–162.

Ko, Y.-B., & Vaidya, N. (1998). Location-aided routing (LAR) in mobile ad hoc networks. *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom_98)*.

Ko, Y.-B., & Vaidya, N. (1998). Location-aided routing (LAR) in mobile ad hoc networks. *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 98)*.

Levis, P. (2010). *Simulating TinyOS Networks*. Retrieved June 23, 2010, from http://www.cs.berkeley.edu/~pal/research/tossim.html

Lin, Y., Rad, A., Wong, V., & Song, J. (2005). Experimental Comparisons between SAODV and AODV Routing Protocols. *Proceedings of the 1st ACM workshop on Wireless Multimedia Networking and Performance Modeling*, 113-122.

Liu, Y., Zhang, H., Ni, Q., Zhou, Z., & Zhu, G. (2008). An Effective Ant-Colony Based Routing Algorithm for Mobile Ad-hoc Network. *4th IEEE International Conference on Circuits and Systems for Communications*.

Madhavapeddy, A., Scott, D., & Sharp, R. (2003). Context-Aware Computing with Sound. *The Fifth International Conference on Ubiquitous Computing (UbiComp 2003)*.

N. K., C., & Viswanatha, K. (2008). Enhanced Ant Colony Based Algorithm for Routing in Mobile Ad Hoc Network. *World Academy of Science, Engineering and Technology 46*.

Nickray, M., Dehyadgari, M., & Afzali-kush, A. (2009). Adaptive Routing Using Context-Aware Agents for Networks on Chips. *4th International Design and Test Workshop (IDT).*

Nikaein, N., Laboid, H., & Bonnet, C. (2000). Distributed dynamic routing algorithm (ddr) for mobile ad hoc networks. *Proceedings of the MobiHOC 2000: First Annual Workshop on Mobile Ad Hoc Networking and Computing*.

OWL, W. (2010). *OWL Web Ontology Language Overview*. Retrieved April 21, 2010, from OWL Web Ontology Language Overview: http://www.w3.org/TR/owl-features/

Park, V., & Corson, M. (1997). A highly adaptive distributed routing algorithm for mobile wireless networks. *Proceedings of INFOCOM*.

Pei, G., Gerla, M., Hong, X., & Chiang, C. (1999). A wireless hierarchical routing protocol with group mobility. *Proceedings of Wireless Communications and Networking*.

Perkins, C., & Watson, T. (1994). Highly dynamic destination sequenced distance vector routing (DSDV) for mobile computers. *ACM SIGCOMM 94 Conference on Communications Architectures*.

Perkins, C., Belding-Royer, E., & Das, S. (2003, July). Ad hoc On-demand Distance Vector (AODV) routing. *IETF RFC 3561*.

Porekar, J. (2003). Random Networks.

Radhakrishnan, S., Rao, N., Racherla, G., Sekharan, C., & Batsell, S. (1999). DST–A routing protocol for ad hoc networks using distributed spanning trees. *IEEE Wireless Communications and Networking Conference*.

Raju, J., & Garcia-Luna-Aceves, J. (1999). A new approach to ondemand loop-free multipath routing. *Proceedings of the 8th Annual IEEE International Conference on Computer Communications and Networks (ICCCN)*, 522–527.

Rao, V. P., & Marandin, D. (2006). Adaptive Channel Mechanism for Zigbee (IEEE 802.15.4). *Journal of Communications Software and Systems (JCOMSS)*, 283-293.

RDF, W. (2010). *RDF - Semantic Web Standards*. Retrieved April 21, 2010, from RDF - Semantic Web Standards: http://www.w3.org/RDF/

Ruzzelli, A. (2008). What to expect when programming a WSN. *SenZation'08 - WSN Summer School.* Ljubljana.

Saeed, N., Abbod, M., & Al-Raweshidy, H. (2008). Intelligent MANET Routing System. *22nd International Conference on Advanced Information Networking and Applications - Workshops.*

Sato, T., & Mase, K. (2002). A scatternet operation protocol for Bluetooth ad hoc networks. *The 5th International Symposium on Wireless Personal Multimedia Communications*.

Schafersman, S. D. (1994, January). *Scientific Thinking and the Scientific Method.* Retrieved September 10, 2008, from An Introduction to Science: http://www.freeinquiry.com/intro-to-sci.html

Schmidt, A., Beigl, M., & Gellersen, H.-W. (1999). There is more to Context than Location. *Computers & Graphics Journal*, 893-902.

Shah, N., & Qian, D. (2009). Context-aware routing for peer-to-peer network on MANETs. *IEEE International Conference on Networking, Architecture, and Storage.* Beijing.

Shqeerat, K. A. (2008). Adaptive algorithm for increasing the efficiency ofDSR algorithm in Ad Hoc network. *5th International Multi-Conference on Systems, Signals and Devices*.

Sobeih, A., Chen, W.-P., Hou, J., Kung, L.-C., Li, N., Lim, H., et al. (2005). J-Sim: A Simulation and Emulation Environment for Wireless Sensor Networks. *IEEE Wireless Communications magazine*.

Soylu, A., De Causmaecker, P., Desmet, P., & Leuven, K. (2009). Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering. *Interdisciplinary Research on Technology Education and Communication (iTec)*, 992-1013.

Su, W., & Gerla, M. (1999). Ipv6 flow handoff in ad-hoc wireless networks using mobility prediction. *IEEE Global Communications Conference*, 271–275.

Sujatha, B., V.P, H., Namboodiri, D., & Sathyanarayana, D. (2008). Performance Analysis of PBANT (PBANT: Position Based ANT Colony Routing Algorithm for MANETs). *Proceedings of the 16th International Conference on Networks, ICON 2008*.

Tanenbaum, A. (1996). *Computer Networks - 3th Edition.* Prentice-Hall.

Toh, C. (1996). A novel distributed routing protocol to support ad-hoc mobile computing. *IEEE 15th Annual International*, 480–486.

Tretmans, J. (2001, January 25). An Overview of OSI Conformance Testing. University of Twente.

Varga, A., & Hornig, R. (2008). An overview of the OMNeT++ simulation environment. *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops.* Marseille: ICST.

Wang, K., Wu, M., Weifeng, L., Pengrui, X., & Shen, S. (2008). A Novel Location-Aided Routing Algorithm for MANETs. *Fifth International Conference on Information Technology: New Generations*.

Weingärtner, E., Lehn, H., & Weh, K. (2009). A performance comparison of recent network simulators. *Proceedings of the IEEE International Conference on Communications 2009 (ICC 2009).* Dresden: IEEE.

Woo, S.-C., & Singh, S. (2001). Scalable routing protocol for ad hoc networks. *Wireless Networks*, 513–529.

Xue, Y., & Li, B. (2001). A Location-aided Power-aware Routing Protocol in Mobile Ad Hoc Networks. *IEEE Globecom*, 2837--2841.

Zahary, A., & Ayesh, A. (2007). Analytical study to detect threshold number of efficient routes in multipath AODV extensions. *International Conference on Computer Engineering & Systems, 2007. ICCES '07*.

Zarei, M., Faez, K., & Nya, J. (2008). Modified Reverse AODV routing algorithm using route stability in mobile ad hoc networks. *Multitopic Conference, 2008. INMIC 2008. IEEE International*.

Zhao, F., Zhan, X., Yao, Y., & Yi, B. (2005). An Enhanced Reactive Dynamic Source Routing Algorithm Applied to Mobile Ad Hoc Networks. *Proceedings. 2005 International Conference on Wireless Communications, Networking and Mobile Computing*, 771 - 775.

ANNEX A. NETWORK TOPOLOGIES

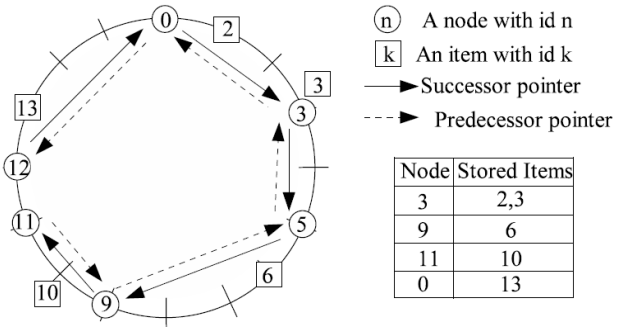
Networks can be either *physical* or *overlay*: *physical* means that networks work in the sense of their physical connections without taking any measures regarding efficiency; *overlay* means that the network has sensing abilities regarding its nodes, their roles and connections awareness. This research work will only focus on physical networks due to their low level application.

Networks can further be divided in two groups: *structured* and *non-structured*.

Structured overlay networks are stable networks that offer deterministic routing: for example O(*log N*) for N nodes. Nodes are established by some rules and possibly identified by a universal identifier; node's neighbours have sense awareness for fault tolerance; these networks are typically flat and sometimes referred as DHT. The main disadvantage of these networks is the fact that the *id* of a node does not have geographic awareness, which can lead to latency problems. Some examples of main DHT systems are presented here according to (El-Ansary & Haridi, 2005).

A.1. STRUCTURED NETWORKS

A **Chord** network assumes an overlay graph as a ring (Figure A.1: with an id space N = 16 and each node has 4 edges = $\log_2(16)$) with an identifier space of size *N* (*N* = number of nodes) (El-Ansary & Haridi, 2005). Each node has an identifier *u*, an identifier to the following node (*Succ*(*u*)), an identifier to the previous node (*Pred*(*u*)) and a list of fingers with $M = \log_2(N)$, this list is $F_u = \{(u, Succ(u + 2^{i-1}))\}$, $1 \le i \le M$, where the arithmetic is modulo *N*.



Node	Stored Items
3	2,3
9	6
11	10
0	13

FIGURE A.1 CHORD (EL-ANSARY & HARIDI, 2005)

The lookup process is part of how the id space is organized; insertion and querying depends on how successful it is to find the successor of an *id*. In normal situations a lookup needs $O(\log_2(N))$ hops.

There is a stabilization algorithm to organize the network. To enter the network, a node performs a lookup for its *id* and inserts itself between its successor *s* and the predecessor of *s*. To initialize its routing table it asks for *s*'s routing table and copies it, or, let *s* lookup each required edge of *n*. Finally it just needs to ask for the items with id less or equal to *n*. Nodes leave the network by transferring all items to the successor and informing both the predecessor and successor. The node failure has two major problems: loss of stored items and ring disconnection. Facing this problem, each node has a list of nodes that are next to it on the network. If a node detects one failure in a successor, it replaces it with the next entry in the successor list. To tackle this problem, each node stores their items at some node in its successor list, so, if they lose an item by a node failure, $\log_2(N) + 1$ nodes have to fail at the same time.

The **Pastry** tries to fix the issue of locality in DHTs (El-Ansary & Haridi, 2005). By hashing node's IP Numbers / Public Keys, nodes in the same region may need to communicate through another node that is in another region, which brings greater inefficiency because hashing does not take care of physical settings.

Like the Chord, Pastry assumes a circular space where each node has a list of $^{L}/_{2}$ successors, a list of $^{L}/_{2}$ predecessors (*leaf set*) and *M* close nodes (*neighbour set*) given by a metric like network delay, used only for maintaining locality properties. The routing table contains $\lceil \log_{2^b} N \rceil$ rows and $2^b - 1$ columns. *L*, *M* and *b* are network parameters.

To locate the closest node to an id *x*, a given node *n* checks if *x* is in the range of node ids in the leaf set. If so, it is forwarded to that node, otherwise, the lookup is forwarded to the node in the interval where *x* belongs to. The replication and fault tolerance is tackled by replicating items to *k* closest nodes in the leaf set. These copies can be used as cache to other lookups.

In **Kademlia** the identifier space is equal to the one used in Pastry, but the nodes are represented in a binary tree, where each node divides that tree in sub-trees and keeps at least one contact to each of those sub-trees (El-Ansary & Haridi, 2005). Kademlia does not have a predecessor nor a leaf set, it keeps, at most, *k* contacts for each sub-tree and for each k contacts in a sub-tree as a *k*-bucket.

The notion of distance is given by the bitwise exclusive or (XOR) of two identifiers. The lookup is performed in a concurrent and iterative manner. If an id belongs to a sub-tree, the query is forwarded to one of the nodes in the *k*-bucket. The lookup process is resolved in O(log(*N*)) hops. To maintain routing tables, Kademlia uses the lookup traffic. The reception of a message for a sub-tree is used to update the *k*-bucket of that sub-tree. Kademlia also updates the latency of nodes in *k*-buckets, with these maintenances this network has sense to delay and locality. For fault tolerance, Kademlia just depends on the number of *k* connections of the sub-trees.

Koorde is a DHT based on the DeBrujin graph (El-Ansary & Haridi, 2005). Koorde can resolve a query in at most $\log_2(N)$. To do this, when a node *n* wants to find an item *x* represented in binary as $d1d2..d_{\log2(N)}$, it takes the first bit and forwards the query to $E_n \circ d$. To manage nodes joining and leaving the network, Koorde, like Chord, has a mechanism for maintenance.

For fault tolerance, Koorde has to maintain an out-degree less than *log*(*N*) nodes, otherwise, a node will lose its contacts. This is an advantage over usual logarithmic DHTs.

The **Viceroy** is a DHT system based on the Butterfly network (El-Ansary & Haridi, 2005). Like Chord, it organizes the nodes in a circular identifier space, but also in levels from 1 to $log_2(N)$, where nodes also have *Butterfly pointers*, *up* and *down* (one up, instead of the upper level nodes and two down pointers) and pointers to successors and predecessors at the same level (Figure A.2).

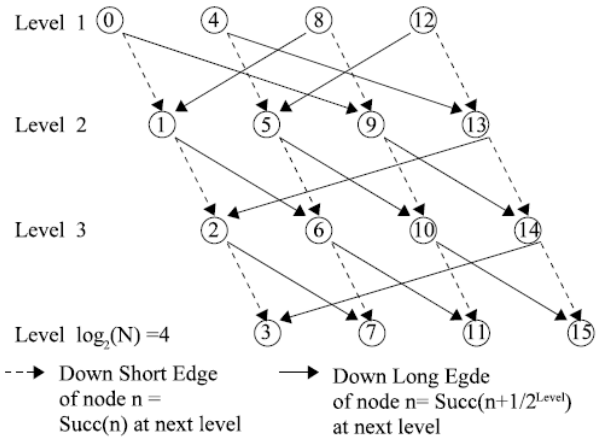


FIGURE A.2 DOWN POINTERS IN VICEROY (EL-ANSARY & HARIDI, 2005)

When a node wants to find an item, the query is forwarded by the nodes using the pointers, by using up and down pointers all levels can be reached in 2 * *log*(*N*) hops. To join the network, a node just needs to find its successor in the ring, then select a level based on the number of nodes, and then it can estimate its pointers. When a node disconnects, it informs the nodes pointed by its pointers, and its items are transferred to the successor.

The **CAN** protocol has its own identifier space and is based on a *d*-dimensional coordinate space (El-Ansary & Haridi, 2005). The hashing function of the identifiers are performed *d* times to get *d* coordinates and the coordinate space is dynamically portioned by all nodes. To get an item, the Cartesian straight line path is used, from source to destination. When a node wants to join the network, it sends a JOIN message and a zone is created by splitting an existing zone; this has a cost of O(*d*). When leaving, a node informs its neighbours and merge zones to valid zones. If there are no valid zones, its items are transferred to the neighbour with the smallest zone.

When a node detects a neighbour failure, it takes over its zone. A node can also detect other node failures by not receiving the periodical maintenance message. If two or more nodes want to take over a zone, they send each other their zone size and the node with the smallest zone wins the zone.

A **Scatternet** is a kind of network that deserves attention because it's a structured network created in the scope of Bluetooth (Figure A.3). This is a network topology, which is composed by two or more *piconets* (Sato & Mase, 2002).

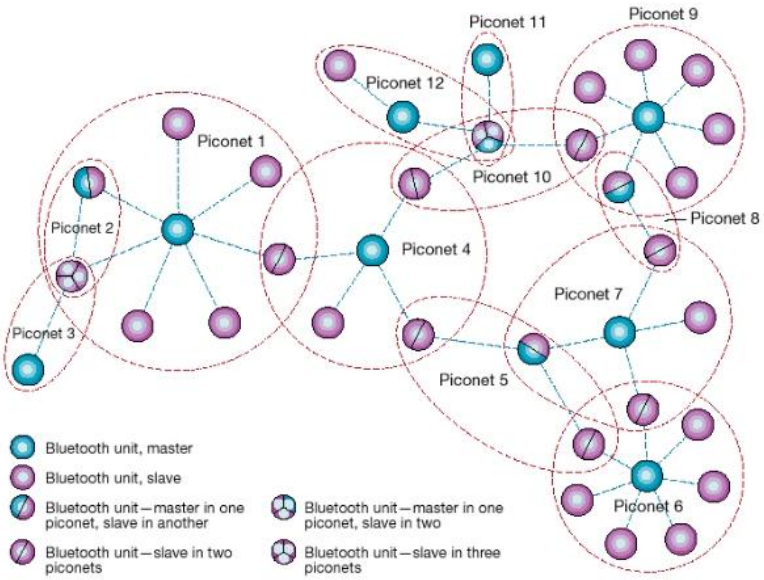


FIGURE A.3 BLUETOOTH SCATTERNET (SOURCE ACM)

A piconet is a group of two or more nodes synchronized where one node is the master and the others are slaves. In each piconet there can be up to seven active slaves at the same time. The master is responsible for selecting the correct frequency hopping to be used on the communications in the piconet, while it also communicates with the slaves through *Time Division Multiplex*. Another property is that if a node belongs to more than one piconet, it is called a *gateway*; there can only be a master node in a piconet.

A.2. NON-STRUCTURED NETWORKS

Non-structured networks are typically found on Wide Area Networks (WAN) (Tanenbaum, 1996), where the network connections are created in a random/irregular form; they are easy to establish, have a light organization and are fault tolerant, but, have some disadvantages in terms of routing and latency.

Some of these networks fit in a feature called *Small World Property*. This property tells us that a large network can have a small diameter or a small average path length. This property can be observed in society and nature (Barabási & Bonabeau, ScaleFree Networks, 2003) (Porekar, 2003), this is similar to what happens in chemicals where inside a living cell are at average three reactions away from each other; in network of scientific papers connected by citations; in the connections between Hollywood actors; in the cellular metabolic networks; in the protein-interaction network of cells; even in Internet and WWW can be found the small world property (!)…; despite these and other examples of social and natural

inadvertent organization, this is not meant to be a universal principle, but can be an opportunity to study.

According to (Porekar, 2003), **Random Networks** were presented by Erdõs and Reyni and represent the most simple network model that fits in the *Small World Property* because the average distance between two points are ln(*N*) and nodes follow a Poisson distribution to their degree (Porekar, 2003). These networks assume that all nodes are equal and with the same properties.

The **Scalefree Networks** (Barabási & Bonabeau, ScaleFree Networks, 2003) are networks that also fit the small world model and where the degree distribution follows the *power-law* (the probability that a node is connected to *k* other nodes is proportional to $1/k^n$). These networks do not assume that nodes may have different roles and different properties, so, in Scalefree networks there is the existence of *hubs*, which are nodes that tend, over time, to have more connections than others. This helps to explain the popularity of some nodes in certain networks, like *routers* in Internet or URLs in WWW (Figure A.4).

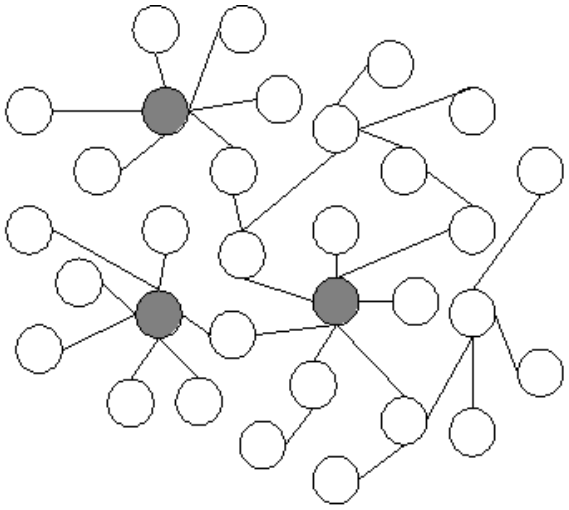


FIGURE A.4 A SCALEFREE NETWORK

Barabási and Albert proposed a model (BA Model) for creating and maintaining a Scalefree network that is called *Growth* and *Preferential Attachment* (Barabási & Albert, Emergence of scaling in random networks, 1999) (Figure A.5 source (Barabási & Bonabeau, ScaleFree Networks, 2003)). The Growth is defined by a network with a "*small number (m_0) of vertices, at every time step, we add a new vertex with m(<m_0) edges that link the new vertex to m different vertices already present in the system*" and Preferential Attachment is that node

connects to an existing node *i* with probability $\prod$, depends on the degree k_i of that node *i*, such as: $\prod(k_i) = \frac{k_i}{\sum_j k_j}$. After some steps t the network has $N = t + m_0$ nodes, *mt* edges and each node has *k* edges with a probability following the power law with exponent 3, this value is independent of *m*, the unique parameter of the model.

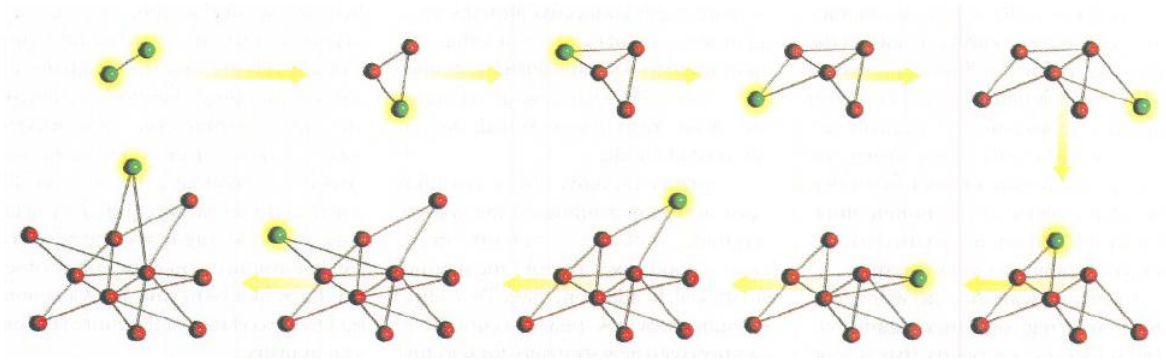


FIGURE A.5 GROWTH AND PREFERENTIAL ATTACHMENT

These networks are reliable in the presence of arbitrary failures; like in the Internet, although many routers continue malfunctioning, the network rarely suffers disruptions. This situation was simulated, and even when 80% of random routers fail, the others form small clusters which still makes it possible to communicate between two nodes. The problem of these networks is located in the hubs: by removing some major hubs the network can be divided in isolated clusters, or like in proteins, the deletion of some hubs can kill the organism, which does not happen with other nodes.

These networks may be used in Medicine on vaccine campaigns or in identifying hub molecules in some diseases, which can lead to new discoveries; in Business to understand the connections between companies and industries, avoiding cascading financial failures or in the propagation of a product in the society; and in Computer Science by creating networks resilient against failures or against virus attacks.

ANNEX B. ROUTING ALGORITHM CLASSES AND COMPARISONS

Nowadays are used two classes of routing algorithms are used: *link state* and *distance vector algorithms* (Abolhasan, 2003). In link state each node periodically floods the network to have a view of the network; in distance vectors each node *i* maintains a set of distances D_{ij}^{x} to a destination *x* and *j* ranges over the neighbours of node *i*. To select the shortest path to a destination, the node *i* selects a neighbour *k* to be the next hop for *x* in $D_{jk}^{x} = min_{j}\{D_{ij}^{x}\}$. The maintenance of these values is periodically updated to the network. Traditionally these algorithms do not scale to large networks because periodically route updates that cause network delays, power and bandwidth consumption. These algorithms can be separated in three major classes: global/proactive routing protocols; on-demand/reactive routing protocols; and hybrid routing protocols.

B.1. GLOBAL / PROACTIVE ROUTING PROTOCOLS

In these protocols the route to the destination is calculated in the start-up and then maintained by a periodic process. Following some algorithms of reference within this class of routing approach are presented:

The **Destination-Sequenced Distance-Vector** (DSDV) is a loop free routing algorithm and gives one shortest path to the destiny (Perkins & Watson, 1994). It uses two kinds of messages to routing table update in order to reduce the network overhead: the *full dump* and the *incremental*. The full dump is a message that contains all the routing information and the incremental is just the latest changes since the last full dump. This algorithm introduces too much overhead in the network in order O(N^2) for N nodes, and so, it does not scale to large networks.

The **Wireless routing protocol** (WRP) is also a loop free routing algorithm (Garcia-Luna-Aceves S. M., 1995). It is not suitable for devices with limited capacity of processing, because each node requires handling four routing tables and requires *hello* messages; this consumes a lot of bandwidth and if a node needs to sleep the algorithm will not work.

The **Global state routing** (GSR) is a protocol based on the Link State algorithm (Chen & Gerla, 1998) but it reduces the number of update messages only between intermediate nodes. To update routing tables each node periodically exchange messages with the neighbours, this reduces the number of control messages needed in the network, but if the network grows, the update messages will be larger and a large amount of bandwidth will be required.

The **Fisheye state routing** (FSR) is an improvement to GSR (Gerla, 2002). It reduces the message number by using the concept of fish eye area; it sends more update information to the nodes inside that area rather than the nodes outside. With the mobility of the nodes, node discovery will become less accurate, and so, is not suitable to handle scalability.

The **Source-tree adaptive routing** (STAR) is also an algorithm based in Link State algorithm (Garcia-Luna-Aceves & Spohn, 1999). Each node maintains a set of links containing the paths to destinations. This protocol reduces the overhead in the network when compared with the original link state, by using the LORA (or ORA) approach to exchange messages, and so, eliminating the periodic updating procedure. Due to its reduced overhead, this algorithm will scale to large networks, but to scale to large and mobile networks, this protocol will have a high overhead because each node must maintain a part of the network topology.

The **Distance routing effect algorithm for mobility** (DREAM) is an algorithm based on geographical coordinates through GPS stored in its routing table, known as location table (Basagni, Chlamtac, Syrotivk, & Woodward, 1998). This has an advantage because consumes less bandwidth, and so, is more scalable. Routing message update overhead is reduced by sending updating messages based on the mobility and distance of nodes.

The **Multimedia support in mobile wireless networks** (MMWN) is a routing protocol based on clustering hierarchy and the information stored in a dynamic distributed database (Kasera & Ramanathan, 1997). Each cluster has three kinds of nodes: *switches*, *endpoints* and a *location manager* (LM) which is responsible for the location management in the cluster. Only the LM perform location and updates, reducing dramatically the overhead of the network, but, due to its hierarchical structure, this location finding and updates are very complex, also, if there is any change in the LMs, this will also affect the management tree and introduce consistency problems.

The **Cluster-head gateway switch routing** (CGSR) is another hierarchical algorithm where nodes are clustered (Chiang, 1997). It is very similar to MMWN, but in CGSR there is no need to maintain the cluster hierarchy. To do this, there is an elected mobile node, called *cluster head*, responsible to manage the other nodes; it also controls the communication *medium* and inter-cluster communications. This is good because each node only needs to maintain the route to its cluster head, but, there are great overheads in maintaining the cluster membership because each node periodically broadcasts its cluster member table and the updates are made based in these broadcasts.

The **Hierarchical state routing** (HSR) is a protocol based on the Link State algorithm (Pei, Gerla, Hong, & Chiang, 1999), but, unlike others, it maintains a hierarchical addressing and a topological map. Similarly to MMWN, this protocol has also three kinds of nodes, but with other roles: *cluster-head* are the nodes responsibility for the local coordination, *gateways* are the nodes that lie in two clusters and, *internal* nodes that are the nodes inside each cluster. Each node can be identified by a unique ID based on its MAC address and a hierarchical ID (HID) which is a sequence of the MAC addresses from the top of the hierarchy to the source node. This protocol has the advantage because of separating the mobility management from the physical management. This protocol has less overhead compared to GSR and FSR, but introduces more overhead in the cluster formation and maintenance.

The **Optimised link state routing** (OLSR) is an algorithm based on the link state algorithm (Jacquet, Muhlethaler, Clausen, Laouiti, Qayyum, & Viennot, 2001). This algorithm reduces the size of each control message and the number of rebroadcasting nodes during the route update using multipoint relaying (MPR). In each topology update, each node selects a group of neighbours, called *multipoint relays*, to send the update information; the other nodes can read the packets but are not allowed to retransmit them. The routes to a destination are stored in the routing table and when a node wants to send a message to a destination, that route is available.

According to (Bellur, Ogier, & Templin, 2003), **Topology broadcast reverse path forwarding** (TBRPF) is another link-state algorithm that uses *reverse-path forwarding* (RPF) to change update packets in the reverse direction of the *spanning tree*. With this approach, it is possible to get a path to all destinations, applying a modified version of the Dijkstra's algorithm (Dijkstra, 1959). To minimize the overhead, each node sends only a part of its

source tree to the neighbours. To report changes in the network, each node send a periodic and differential hello messages; these messages only report the changes in the status of the neighbour nodes, and as result, the hello messages are smaller.

According to (Abolhasan, 2003), this kind of protocols does not scale well in large networks. The best ones are OLSR (because it chooses the neighbours to send the packets); and DREAM (because can change physical information rather than link state information). The common disadvantage in all of this protocols are in the behaviour presence of node mobility, that introduces unnecessary overhead to the network, and in the context of this work, this is a great issue to take care.

B.2. ON-DEMAND / REACTIVE ROUTING PROTOCOLS

These protocols calculate their routes only when required by the source, this process is usually called the *on-demand* route discovery. Some algorithms of this class are presented in the following paragraphs.

The **Dynamic source routing** (DSR) protocol requires the addresses of all nodes, from the source to the destination (Johnson, Maltz, & Jetcheva, 2002), so, is not a good choice to large and dynamic networks. This protocol is good for network with low mobility or a small number of nodes.

The **Ad hoc on-demand distance vector** (AODV) is a routing algorithm based on DSDV and DSR (Das, Perkins, & Royer, 2002); it uses periodic message exchange to maintain the connections and sequence numbering to avoid loops, like DSDV, and the route discovery is similar to DSR. The differences between AODV and DSR is that AODV only carries the address of the destination, while DSR carries all the routing information, leading to less overhead; another difference is in the replies, where DSR carries all the nodes of the route, while AODV only needs the destination address and the sequence number. This algorithm is very suitable to highly dynamic networks, but has some disadvantages in route discovery and link failure, because these events can bring extra bandwidth consume to the network.

The **Routing on-demand acyclic multi-path** (ROAM) is a protocol that uses internodal coordination and *diffusing computation*, which are directed acyclic sub graphs (Raju & Garcia-Luna-Aceves, 1999). This protocol eliminates the *count-to-infinity* problem stopping the flood when the search reaches the destination. There are router nodes which maintain entries to destinations and the packets flow thought them, reducing the storage and bandwidth needed; another advantage is the fact that when the distance between a router and the destination changes more than a threshold, it broadcasts update messages to its neighbours.

According to (Corson & Ephremides, 1995), **Light-weight mobile routing** (LMR) is a protocol that uses flood techniques to determine routes. This algorithm only maintains routing information about its neighbours and may have multiple routes to the destinations creating more reliability, but, when there is a route failure, it can introduce extra delays by determining the correct route.

The **Temporally ordered routing algorithm** (TORA) is an algorithm based on LMR (Park & Corson, 1997). This protocol has the advantage of a reduced number of control messages to the neighbours when the network changes; another vantage is that it supports multicasting. Similarly to LMR, TORA can produce invalid routes.

The **Associativity-based routing** (ABR) is another reactive algorithm based on route selection based on stability (Toh, 1996). Each node has an associativity *tick* to their neighbour, which is used to select the preferred route to the neighbour with lower tick. This cannot guaranty the shortest path, but there is no need to reconstruct routes and there will be more bandwidth available for communications. The disadvantages of this protocol are that is necessary a periodic maintenance of the ticks and so, nodes have to be always active conducting to extra power consumption. Another disadvantage is the fact that it does not maintain multiple routes or route cache.

The **Signal stability adaptive** (SSA) protocol is an improvement to the ABR protocol, but it uses signal strength and location stability instead of association tick (Dube, Rais, Wang, & Tripathi, 1997). Comparing to AODV and DSR, SSA intermediate nodes cannot reply to route requests send to a destination, and this can lead to a long time to route discovery. Another disadvantage is the fact that when a link fails there is no attempt to repair a route, instead a reconstruction of the entire route since the source is made.

According to (Aggelou & Tafazolli, 1999), **Relative distance micro-discovery ad hoc routing** (RDMAR) limits the route request packets to a certain number of hops. This technique can conserve a significant amount of bandwidth and battery power. A disadvantage of this protocol is if there is no previous communication between a source and a destination, route discovery will be performed by flooding.

The **Location-aided routing** (LAR) is a routing protocol based on flooding techniques with the notion of GPS coordinates (Ko & Vaidya, 1998). There are two schemes proposed to LAR, the first, calculates a boundary where the requests can reach; the second, stores the destination coordinates in the route request. These methods can control overhead and conserve bandwidth and determine the shortest path. The disadvantage is that each node requires GPS capabilities. In highly dynamic networks, this protocol behaves very similar to DSR and AODV.

According to (Günes, Sorges, & Bouazizi, 2002), **Ant-colony-based routing algorithm** (ARA) attempts to reduce the overhead in the network by simulating the food searching behaviour of the ants. When ants want to find food, they start at their nest leaving a trail of pheromones. With the pheromones, the other ants can follow the same path until it disappears. This algorithm is based on two phases, the first sends a Forward Ant (FANT) to the network, leaving pheromones in the nodes, as soon as once the ant gets in the destination a Backward Ant (BANT) is created and returns to the nest; this is the route discovery phase. The route maintenance phase is made when a packet flows through a node: it increments the pheromone value, otherwise, the pheromone value is decreasing until it expires. If a route fails, the nodes inform their neighbours of an alternate route; if they have a route, they will inform their neighbours by backtracking. If any node founds the route and the source node is reached a new route discovery is initiated. The FANT and BANT messages are a good approach because their small size introduces low bandwidth consumption, but the flooding procedure can bring problems when the network grows.

The **Flow oriented routing protocol** (FORP) is an algorithm that tries to predict a node failure due to mobility, by choosing an alternate path in this situations (Su & Gerla, 1999). When a node wants to communicate with other node and the route is not available, it broadcasts a Flow_REQ message; when a node receives the Flow_REQ, it calculates the Link Expiration Time (LET) with the previous hop given by GPS and then appends to the Flow_REQ and

rebroadcasts the value. When the Flow_REQ reaches the destination, that node takes the minimum LET in the Flow_REQ and sends a Flow_SETUP to the source. In the transmission each node appends their LET allowing the destination to predict when a node could fail or not. This protocol uses pure flooding techniques, which have problems in large scale networks; so is not suitable to this work.

The **Cluster-based routing protocol** (CBRP) is a hierarchical protocol where nodes are organized in clusters (Jiang, Ji, & Tay, 1999). These clusters have a cluster-head that controls communications. This protocol is good because has a small number of control messages but the cluster formation and maintenance creates overhead; sometimes routing loops can exist, due to inconsistent topology.

According to (Abolhasan, 2003), this class of routing algorithms have the same cost in the worst scenario due to the similar route discovery and maintenance procedures. This happens when a node does not have a route to a destination, what usually happens in the initial stage because there is any route available. The DSR, for instance, when a route expires, it floods the network searching for another route, in other case, the LAR or RDMR keep a route history to limit the search zone. The ABR or the SSR have another method to minimize the data transfer in the network, by select routes based on the stability, however, the ABR shows better results compared to SSR, and both perform better path selections than DSR.

This class is suitable for medium size protocols and can handle moderate mobility.

B.3. HYBRID ROUTING PROTOCOLS

This is a “special” kind of routing protocols because they combine proactive and reactive protocols. They can have a flat or hierarchical routing structure and use the other two approaches.

The **Zone routing protocol** (ZRP) is a protocol that defines two zones (Hass & Pearlman, 1999). The first zone is defined by a boundary, defined by a number of hops. Within this zone, the routes to a destination are immediately available; outside the zone, the algorithm

behaves like a reactive routing protocol. This algorithm can reduce the overhead and delays compared with reactive protocols. For large values of the boundary, this protocol behaves like a proactive algorithm, while for small, behaves like a reactive.

According to (Joa-Ng & Lu, 1999), **Zone-based hierarchical link state** (ZHLS) is a hierarchical protocol that defines two levels: node level topology and zone level topology. It uses the cluster-head and the location manager to coordinate the communications, what leaves them with no processing overhead and avoid bottlenecks and single points of failure; another advantage is that it reduces the communication overhead comparing with reactive protocols. When a route is required, the source broadcasts a zone-level location request; this reduces the overhead comparing with flooding techniques. While the destination node does not migrate, no location search is necessary. The disadvantage is that all nodes have to be prepared with the zone map in order to operate; this is bad when the network is highly dynamic.

The **Scalable location update routing protocol** (SLURP) is a routing protocol similar to ZHLS (Abolhasan, 2003), because it organizes nodes in zones (Woo & Singh, 2001). This algorithm reduces the overhead in routing information, eliminating the global discovery, so, each node has a *home zone* which is determined by a static mapping function (*f*(NodeMACAddress) → regionID). With this approach, any node can find the home zone; of a given node. Each node also maintains its home zone by unicasting a location update message to the home zone, when this message arrives, it is broadcasted to all the nodes in the zone, and then all nodes can unicast a location_discovery packet to the same zone nodes. When the location of the destination is found, the source can start sending information based on the most forward with fixed radius (MFR) geographical forwarding algorithm; when the information reaches the home zone of the destination it is sent by *source routing*. The home zones are the main disadvantage of this protocol.

In **Distributed spanning trees based routing protocol** (DST) a tree like network is used to group the nodes (Radhakrishnan, Rao, Racherla, Sekharan, & Batsell, 1999). These nodes can be *internal nodes* or *router node*s; the router nodes are responsible for the tree structure and internal nodes are the regular nodes. They can be in three kinds of state: *router*, *merge* and *configure*, depending on their roles. To find a route, this protocol proposes two approaches: the *hybrid tree-flooding* (HTF) and *distributed spanning tree shuttling* (DST). In

HTF the messages are passed by flooding the tree, and each node holds the packets for a period of time; this can be useful because network connectivity increases along the time. The DST sends the packets along the source to the leafs; when it reaches the leaf, it is sent up until a level called shuttling level and then send down to the tree or to the adjoining bridges. There are two main problems in this algorithm: the fact of using a tree approach leads to a single point of failure in the root node, and in HTF the holding time can bring delays to the network.

According to (Nikaein, Laboid, & Bonnet, 2000), **Distributed dynamic routing** (DDR) is an algorithm that is tree based, but that does not requires a root node; this can be done by beaconing neighbour nodes, and this will form is what is called a forest. The DDR consists of six phases: preferred neighbour election (the neighbour with more neighbours), forest construction (connection to the preferred neighbour), intra-tree clustering (creation of the intra-zone routing table), inter-tree clustering (creation of the routing table to the neighbouring zones), zone naming (zone ID assignation) and zone partitioning (non-overlapping zones); all these phases are performed based on the beaconing messages. The routing is made up in the DDR by a hybrid routing protocols, this protocol does not require static zone maps, but the choice of preferred neighbours can lead to bottlenecks and network delays; in large networks this can lead to packet drops.

As said in (Abolhasan, 2003), this class of algorithms have the potential to scale better than the reactive or proactive classes, because they attempt to reduce the message overheads by organizing nodes in order to organize the routing procedures. Like ZHLS, only the nodes more suitable can be used for route discovery, and then, creating collaboration between nodes, or in SLURP, the nodes work together in order to maintain the information of the zone. These procedures can potentially eliminate or reduce the number of flooding messages and they also attempt to eliminate bottlenecks or single points of failure in the network.

B.4. ROUTING ALGORITHMS CLASSES COMPARISON

As said in (Abolhasan, 2003), in global routing the flat addressing is easy to implement but is not suitable to large networks because it introduces overhead. To avoid this problem LBS (Location based services) can be used, like in DREAM or by using conditional updates besides periodical updates, like in STAR; the use of hierarchical schemes may create overhead due to location management. The flood based algorithms like DSR and AODV have stability problems in large networks due uncontrolled route discovery and route maintenance; again, the use of LBS systems can bring advantages like in the LAR protocol. The hybrid routing protocols have advantages in large networks when compared with hierarchical routing protocols because the location management is simplified. For example the ZRP protocol was designed to increase the scalability of mobile ad-hoc networks by maintaining the network connectivity for the routing zone and using an approach better than flooding outside that zone; it also can use other protocols for routing inside the routing zone, like LAR. In Table B.1 a schematic comparison between the three classes of protocols is presented.

Table B.1 Comparison between routing protocols

Routing class	Proactive	Reactive	Hybrid
Routing structure	Both flat and hierarchical	Mostly flat, except CBRP	Mostly hierarchical
Availability of route	Always available	Determined when needed	Depends on the location of the destination
Control traffic volume	Usually high, attempt at reduction is made. E.g., OLSR, TBRPF	Lower than Global routing and further improved using GPS. E.g., LAR	Mostly, lower than proactive and reactive
Periodic updates	Yes, However some may use conditional. E.g., STAR	Not required. However some nodes may require periodic beacons. E.g., ABR	Usually used inside each zone, or between gateways
Handling effects of mobility	Usually updates occur at fixed intervals. DREAM alters periodic updates based on mobility	ABR introduced LBQ. ROAM employs threshold updates. AODV uses local route discovery	Usually more than one path may be available. Single point of failures are reduced by working as a group
Storage requirements	High	Depends on the number of routes kept or required. Usually lower than proactive	Usually depends on the size of each cluster or zone may become as large as proactive protocols if clusters are big
Delay level	Small routes are predetermined	Higher than proactive	For local destinations small. Interzone may be as large as reactive protocols
Scalability level	Usually up to 100 nodes. OLSRand TBRPF may scale higher	Source routing protocols up to few hundred nodes. Point-to-point may scale higher. Also depends on the level of traffic and the levels of multihopping	Designed for up to 1000 or more nodes

ANNEX C. RWI NETWORK ARCHITECTURE – RTD ROADMAP

RWI Network Architecture RTD Roadmap	Before 2010 Today	2010-2015 Incremental	2015-2020 Incremental-Visionary	Beyond 2020 Visionary
	Networks of Things	Inter-networks of Things	Cognitive IoT	
Management	***Driver: « Flexible management »***			
	Convergence of frameworks	*Reference framework for autonomic management*	*Self-management & self-everything management frameworks*	
Communication and Routing	***Driver: « Maximum connectivity with minimum consumption »***			
	Network communication control (delay tolerant networks, storage networks)	*New communication schemes with network context awareness (e.g. device physical location)*	*Intelligent adaptive communication based on network condition (e.g. self-learning)*	
Interoperability	***Driver: « Seamless and adaptive interoperability & internetworking »***			
	Support for several disparate interoperability profiles	*Cross-interoperability with heterogeneous networks (e.g. internet)*	*Devices autonomously negotiate & learn communication methods*	
Trust & Security	***Driver: « Enhanced privacy, ethical & legal issues »***			
	Lightweight security mechanisms & protocols for 'simple' devices	*Usability, ID, Adaptive security, Security profiles selection based on security needs*	*Self-adaptive security framework based on perception of high-level intentions & goals*	

FIGURE C.1 RWI NETWORK ARCHITECTURE – RTD ROADMAP

ANNEX D. HARDWARE AND SOFTWARE CONSIDERATIONS

D.1. HARDWARE AND SOFTWARE USED

The proposed solution was implemented and tested in an Intel® Pentium® 4 2.40GHz with 494.6 MiB of memory and 18GiB of disk space. It was used the Linux distribution Ubuntu 9.10 (*karmic*) with the kernel 2.6.31-21-generic and GNOME 2.28.1, without Internet connection.

Although the pre-requisites needed for the execution and configuration of the network simulator are presented in the next chapter and the IDE used was the Anjuta IDE 2.28.0.0 for the implementation of the solution.

D.2. INSTALLATION AND CONFIGURATION OF THE NSNAM

In order to install and configure the nsnam in this Ubuntu release, a set of pre-requisites must be resolved first (*$>* represents the prompt):

- Minimal requirements for C++:

$> sudo apt-get install gcc g++ python

- Minimal requirements for Python:

$> sudo apt-get install gcc g++ python python-dev

- Mercurial:

$> sudo apt-get install mercurial

- Running python bindings from the ns-3 development tree:

$> sudo apt-get install bzr

- A GTK-based configuration system:

$> sudo apt-get install libgtk2.0-0 libgtk2.0-dev

- Debugging:

$> sudo apt-get install gdb valgrind

- Doxygen and related inline documentation:

$> sudo apt-get install doxygen graphviz imagemagick

$> sudo apt-get install texlive texlive-pdf texlive-latex-extra texlive-generic-extra texlive-generic-recommended

- Texinfo for ns-3 manual and tutorials:

$> sudo apt-get install texinfo dia texlive texlive-pdf texlive-latex-extra texlive-extra-utils texlive-generic-recommended texi2html

- The flex lexial analyser and bison parser generator for the Network Simulation Cradle (nsc):

$> sudo apt-get install flex bison

- Basic mobility visualization tests require goocanvas:

$> sudo apt-get install libgoocanvas-dev

- gcc-3.4 is needed for some Network Simulation Cradle (nsc) stacks:

$> sudo apt-get install g++-3.4 gcc-3.4

- To read pcap packet traces:

$> sudo apt-get install tcpdump

- Database support for statistics framework:

$> sudo apt-get install sqlite sqlite3 libsqlite3-dev

- Xml-based version of the config store:

$> sudo apt-get install libxml2 libxml2-dev

- Support for Gustavo's ns-3-pyviz visualizer:

$> sudo apt-get install python-pygraphviz python-kiwi python-pygoocanvas

- Support for utils/check-style.py style check program:

$> sudo apt-get install uncrustify

After this is possible to download the simulator using Mercurial:

$> cd

$> mkdir repos

$> cd repos

$> hg clone http://code.nsnam.org/ns-3-allinone

And then is possible to download the most common options with:

$> ./download.py -n ns-3-dev -r ns-3-dev-ref-traces

And the ~/repos/ns-3-allinone/ns-3-dev directory must have the building script for the simulator. To build it, just type:

$> ./build.py

And a confirmation message must be shown. In order to validate the installation a set of tests can be made, just by typing:

$> ./test.py