CBR Exploiting Broadcast Transmissions using Characteristic Based Communication

Suhaib Naseem

A dissertation submitted to the University of Dublin
in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

August 2013

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Suhaib Naseem: _____

August 30, 2013

PERMISSION TO LEND AND/OR COPY

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

Suhaib Naseem: _____

August 30, 2013

Acknowledgments

I would like to thank my project supervisor, Stefan Weber for being a tremendous mentor. He has been my source of enormous help and encouragement during my research work. Our enjoyable meetings over the last few months and his advice on the project and my career have been invaluable. To all my family for the support everyday during this period. I also thank all the Trinity guys, especially Guoxian Yang who gave me his valuable advice and directions about working with network protocols for ad hoc networks and their simulation techniques. To all my NDS classmates for making this an enjoyable study year. Finally I thank my God, my wife, for helping me to get through all the difficulties and giving me moral support and courage, with out all of them, my endeavor might not have been possible.

Suhaib Naseem

Abstract

In the world today there is an enormous leap in the development of mobile devices, which have grown hugely in capacity and popularity. The enterprise industry has a huge demand and application requirement for new services, in such space we need to envisage ad hoc networks that can genuinely be autonomous and do not rely on the user interaction for the service discovery. Traditional networks have relied upon using IP addresses for communication inside the networks and bind node features to these addresses for service discovery purpose. We find that this type of service discovery mechanism does not suit mobile ad hoc networks as the topology remains dynamic. We view that this IP addressing scheme can be replaced with characteristics or features of nodes. For this we introduce a characteristic based routing protocol called CBR. This protocol spreads the characteristics of nodes across the network through advertisement broadcasts that follows a stream like pattern, similar to the flow of a water stream. In this research project we will establish node communication based on these characteristics rather than the IP address. We choose OMNET++ simulation framework for the evaluation of our protocol. We approach our design through simulation techniques and prove that successful data delivery can be achieved by using node characteristics. We also demonstrate a two-way communication context with service instances broadcasting their device features.

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Applications 2

1.2.1 Service Discovery 2

1.2.2 Internet Connectivity 3

1.3 Research Problems 3

1.4 Contributions 4

1.5 Outline 5

2 Background **7**

2.1 Context 7

2.2 Wireless Communication 9

2.2.1 802.11 Wireless LANs 11

2.2.2 Medium Access Control 11

2.2.3 TDMA 11

2.2.4 Broadcasts 12

2.2.5 Mesh Networks 13

2.2.6 Vehicular Ad-hoc Networks VANETS 14

3 State Of Art **17**

3.1 Gradient based routing approach 17

3.1.1 Overivew 17

3.1.2 Basis of Gradient based routing 18

3.1.3 Gradient Broadcast (GRAB) 19

3.1.4 Directed Diffusion 20

3.2 Routing using potentials 20

3.3 Anycast Routing Protocols 20

3.3.1 Internet 21

3.4 Service Discovery Architectures . . . 22
3.4.1 DNS based service discovery . . . 23
3.4.2 Service Discovery . . . 23
3.4.3 Application-Layer . . . 23
3.4.4 Network-Layer . . . 24
3.5 Network Architectures . . . 25
3.6 Mobile Networks . . . 26
3.6.1 P2PNET . . . 26
3.6.2 DUMBONET . . . 27
3.7 Mobile Ad hoc Network (MANET) routing protocols . . . 28
3.8 Proactive Routing . . . 28
3.8.1 Destination-Sequenced Distance Vector (DSDV) . . . 29
3.8.2 Optimized Link State Routing (OLSR) . . . 30
3.8.3 Hierarchical State Routing (HSR) . . . 30
3.9 Reactive Routing . . . 31
3.9.1 Ad hoc On-demand Distance Vector (AODV) . . . 32
3.9.2 Dynamic Source Routing (DSR) . . . 33
3.10 Hybrid Routing . . . 34
3.10.1 Zone Routing Protocol (ZRP) . . . 35
3.11 Summary . . . 35

4 Design **37**
4.1 Overview . . . 37
4.2 Terminology . . . 39
4.3 Characteristic Based Communication . . . 40
4.4 Characteristic Flow . . . 42
4.4.1 An Example Scenario . . . 43
4.5 Characteristic Hierarchical Coding . . . 48
4.6 Data Forwarding . . . 49
4.7 Route Tracing . . . 50
4.8 Characteristic Updating . . . 50
4.9 Characteristic Table . . . 52
4.10 Weight Cost Calculation . . . 52

5 Implementation **55**
5.1 Overview . . . 55
5.1.1 MiXiM framework . . . 56

5.2 Data Structures . . . 57
5.3 Packets . . . 58
5.3.1 Request and Reply message fields . . . 59
5.3.2 Extra message fields . . . 59
5.4 Single Hop Neighbor Management . . . 60
5.5 CBR Application Layer . . . 61
5.6 CBR Network Layer . . . 61
5.7 CBR as a State Machine . . . 61

6 Evaluation **63**
6.1 Performance Measurements . . . 63
6.2 Generic Scenario . . . 64
6.3 Latency . . . 66
6.4 Success Rate . . . 68
6.5 Throughput . . . 68
6.6 Overhead . . . 69

7 Conclusions **71**
7.1 Future Work . . . 71

A Source Code **73**
A.1 Network Layer Module . . . 73
A.1.1 CbrNetwLayer.CC . . . 73
A.2 Application Layer Module . . . 79
A.2.1 CbrApplLayer.CC . . . 79
A.3 Network Control . . . 82
A.3.1 CbrSingleton.CC . . . 82
A.4 Messages . . . 87
A.4.1 CbrPacket . . . 87
A.5 Configuration . . . 88
A.5.1 omnet.ini . . . 88
A.5.2 CbrApplLayer.ned . . . 91
A.5.3 CbrNetwLayer.ned . . . 92
A.5.4 CharacteristicRoutingNetwork.ned . . . 92
A.5.5 Host80211.ned . . . 92
A.5.6 config.xml . . . 92

List of Figures

1-1 Service Discovery 2
1-2 Internet access inside wireless mesh network [15] 3

2-1 Vehicular Ad-hoc Networks [17] 14

3-1 Gradient Broadcast [22] 19
3-2 Service based topology for MANETs - S: Service provided by A, B and F, H, G: Service requestor. 21
3-3 Service discovery protocol architectures 23
3-4 DNS Service Lookup 23
3-5 Classification of routing protocols in MANETs 27

4-1 Color or Characteristic Propagation [18] 39
4-2 Flow 42
4-3 Propagation [6] 42
4-4 Characteristic Flow Diagram 43
4-5 Ad hoc stable network 45
4-6 Characteristics Hierarchical Scheme 48
4-7 Characteristic Heirarchy - Abstraction 49

5-1 Simulation model for CBR 55
5-2 MiXiM - Framework model 56
5-3 State diagram 62

6-1 Node mesh 16 64
6-2 Mobility 1mps 66
6-3 Mobility 10mps 67
6-4 Mobility 20mps 67
6-5 Success Rate 68
6-6 Throughput - Requests traffic (10/sec) for two characteristics 69

6-7 Overhead advertise in bit/sec 70

List of Tables

4.1 Firefight ad hoc network 44
4.2 Characteristics table of Node M 47
4.3 Characteristics Table Fields 52

5.1 Decider80211 57
5.2 Network Layer Recordings 60
5.3 Application Layer Recordings 60

6.1 Settings 65
6.2 Simulation parameters for our physical layer 65

Acronyms

MANET Mobile Ad hoc Network

CBR Characteristic Based Routing

AODV Ad hoc On-demand Distance Vector

OLSR Optimized Link State Routing

RFC Request for Comments

QoS Quality Of Service

IP Internet Protocol

MAC Medium Access Control

CBREQ Characteristic Request

CBREP Characteristic Reply

CBADV Characteristic Advertise Message

CBTRC Characteristic Trace

CBADVTIMER Characteristic Advertise Timer

GSM Global System for Mobile Communications

UMTS Universal Mobile Telecommunications System

Chapter 1

Introduction

1.1 Motivation

In todays dynamic network space where wireless communication devices have hugely gained capacity, speed and have become popular in the modern world. Just in the past decade devices like PDA's, music players, laptops, smart phones, game consoles e.t.c have gained considerable growth. We envisage new applications using the wireless networking capabilities of these devices which they incorporate, applications like location based services, contextual applications or even any content distribution service can take advantage of these mobile ad hoc networks. Mobile Ad hoc Network (MANET) routing protocols in ad hoc networks have already made some of these applications a reality where the communication is based on Internet Protocol (IP) addresses inside the networks, there is still a space for more flexibility in ad hoc networks which do not rely on IP address or identity for communication between the mobile nodes. Our visionary network forms an ad hoc and autonomic network of participating mobile nodes in which the delivery of information is intended towards certain characteristics or features of nodes rather than the node identity, this is because we see that the idenitity takes no meaning in a dynamic network topology. Therefore, we present a new characteristic based routing protocol which performs routing based on node features or characteristics. Characteristic Based Routing (CBR) semantics allows for the expression of an alternative network address to be called a characteristic. Our light weight routing protocol provides the basic communication primitive of resource or service discovery using a characteristic and does not rely on the IP layer for routing information.

Before looking into the research problems and contributions of this master proof,

we describe application scenarios for wireless ad hoc networks which can benefit from the results of this thesis.

1.2 Applications

For comparison as traditional networks have been designed and developed to provide point to point connectivity like the internet or voice telephony services, such type of networks have been the most widely used applications worldwide, like web content access through web servers or data or voice communication services all rely heavily on these traditional networks. We observe that the communication pattern of such applications exhibit certain features or characteristics.

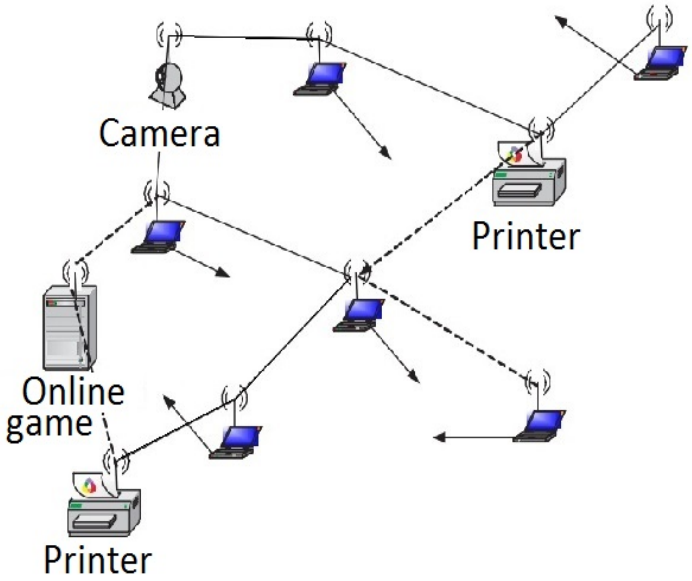

Figure 1-1: Service Discovery

1.2.1 Service Discovery

In order to support roaming mobile users with information services on the move, wireless ad hoc networks are used, therefore we envisage such networks that could provide services like print, games or information posts that the mobile users could access from anywhere on their mobile devices whenever they choose to access. Thus concluding that a particular server name or location providing a specific service becomes irrelevant to the mobile user, however the usage of the service is directly related to the proximity or coverage area of the mobile user.

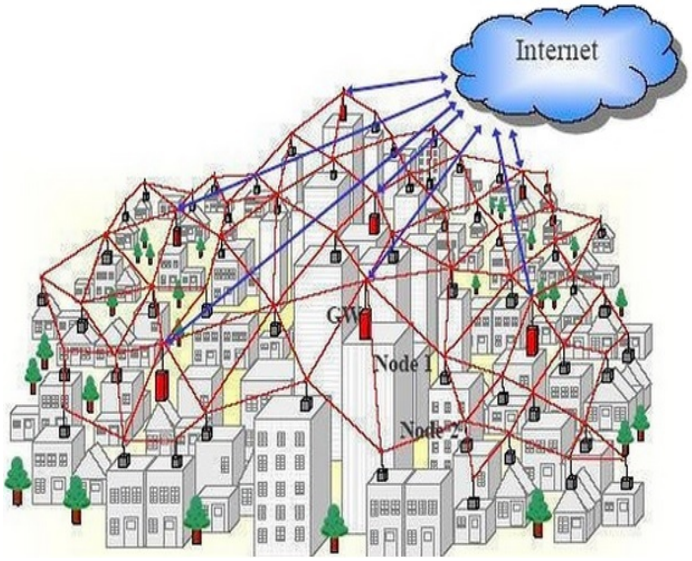

Figure 1-2: Internet access inside wireless mesh network [15]

1.2.2 Internet Connectivity

Mesh networks provide internet connectivity to mobile users. They are multihop wireless networks which are usually deployed on roof tops of houses and buildings in cities or urban central areas. In such type of network only a few number of houses or buildings have an internet connection over a dedicated fixed infrastructure like a dedicated DSL line and thus they can act as the internet gateway node and the rest of the houses or buildings act as relay nodes for providing internet connectivity. The mobile users who want internet access use such kind of an infrastructure for internet connectivity via the nodes or buildings that have got a fixed internet connection. In such type of wireless mesh networks, the routing protocols use anycast routing to route the packets to the gateway nodes to be sent out to the internet.

1.3 Research Problems

Wireless ad hoc networks have gathered a lot of attention of researchers recently because of their wide range of potential applications. These network's comprise of a number of wireless nodes which are installed either inside an application field or close to an application field without the help of any infrastructure that is centrally administered. However wireless ad hoc networks become challenging in many ways. Firstly the node mobility and wireless links adds high degree of dynamism in such networks and secondly the lack of any fixed infrastructure does not allow for a centralized so-

lution and thus we can never rely on a dedicated central server for the control and lastly we explore the broadcast primitive of the 802.11 standard technology inside the physical layer and analyse if a characteristic based routing protocol can increase the success rate of data delivery compared to other routing protocols inside MANETs which are based on addresses.

In this research topic, we look into possibilities and solutions which can tackle the below research problems with regard to the wireless ad hoc networks

- *How to apply the concept of the flow of water stream to routing inside MANETs?* The idea here is to route information or messages inside the network in analogy to the flow of a water stream. As we introduce the concept of characteristics which describe the features of nodes. The characteristics are spread throughout the network which simulate the flow of water stream where the characteristics diffuse in a potential or cost field called weight, just like the potential of a flowing water stream down the hill. These potential values are discrete at individual nodes. We study the effect of this discretization inside the network.

- *How to perform MANET routing in the presence of unreliable links?* This question makes us to think about how to best design the routing strategy and how should our control protocol and design technique cater best for our routing strategy, we define Characteristic Advertise Message (CBADV) messages in our control protocol to take care for route convergence.

- *How to implement service discovery in an ad hoc network?* We need to study the mechanisms of service discovery in a completely distributed environment, thus leading to another question of how could we discover a particular close service which broadcasts a high capacity and also in the best efficient manner when their could be multiple source service instances broadcasting or providing the same service.

1.4 Contributions

The contribution of this work is the design of a characteristics based routing protocol for loop free routing and its application to wireless ad hoc networks. The contributions detail is listed below:

- We propose a robust characteristics routing model which is based on the idea of a capacity or potential field. In this model, it is possible for successful delivery of information to endpoints which have certain features or characteristics even when the links are broken or the node endpoints move along the characteristics path. We thus prove that our new routing model is loop free and we demonstrate that our routing converges in random wireless networks.

- We analyse the comparison of capacity versus proximity based routing. We demonstrate that this type of routing is more robust in a highly dynamic network.

- We present our design for the service discovery protocol for MANETs which is characteristics based. Our new routing protocol joins service discovery and routing which minimizes the control overhead in comparison to employing two separate protocols. We show in our design how the characteristics based routing can distinguish services based on their potential to assist for distributing the requests to good services or to those services with less load or increased capacity.

- We analyse and show the benefits using our approach and present a proof of concept implementation.

1.5 Outline

The structure of this thesis is given below:

- **Chapter 2** In the second chapter, we give a background context and an overview of the wireless communication standards and discuss how the MAC network layer and broadcast nature is utilised inside networks for the purpose of service disovery like internet access in mesh networks.

- **Chapter 3** The third chapter discusses the state of the art approach to MANET routing protocols and different techniques applied for the purpose of establishing a two end communication context.

- **Chapter 4** The fourth chapter discusses the design approach of our characteristic based routing protocol and shows how it could benefit for service discovery and also demonstrates, the establishing of a service context.

- **Chapter 5** This chapter discusses the implementation of the characteristic based routing protocol and the simulation techniques applied using the MiXiM framework.
- **Chapter 6** This chapter will discuss the simulation results and the evaluation done for different performance metrics for our characteristic based routing protocol.
- **Chapter 7** This chapter concludes our research work and discusses our future improvements.

Chapter 2

Background

2.1 Context

As mobile ad hoc networks (MANETs) are networks without infrastructure in which hosts can provide new capabilities such as that of traditional network routers, these kind of networks place new challenges. The mobility also adds complexity to the hosts because of the dynamics of the network topology as it may change unpredictably and quickly over time. Such type of networks could be commissioned for a range of applications. For instance these networks could be deployed very easily in any terrain or disaster scenario like earthquakes. They could even be used in military hostile combat zones where the infrastructure never actually exists. We can say that MANETs are networks for events where spontaneity is valued. MANETs have further opened new research fields to be able to blend with traditional solutions in an environment without any central administration and control. Location based systems, context aware applications or simple applications like content distribution can take advantage of MANETs. Most popular deployments of fixed wireless infrastructure like Global System for Mobile Communications (GSM) or Universal Mobile Telecommunications System (UMTS) have leveraged the usage of such applications and have become a reality in todays world. However there is a growing demand of flexible ad hoc networks that do not rely on base stations that are fixed in mobile networks so as to facilitate communication between mobile nodes. There has been enormous growth in the recent years with small devices like smart phones, , small sensor devices such as iPod, smart TVs, variety of these are embedded with short range wireless networking functions in order to visualise new applications. They aid us to visualise autonomous ad hoc network systems to track the movement of people in campus buildings, corporate buildings or even big cities or town centres. Medium

access control, quality of service (QoS), routing protocols and security are still some issues in mobile ad hoc networks to be dealt with and a number of product solutions and further work is being carried out every year to tackle these issues. Medium access control protocols are designed to work in infrastructure mode which means they facilitate communication in networks through the presence of router nodes. The famous IEEE 802.11 wireless standard allows two hosts to communicate directly in ad hoc mode without the presence of a router node. In the last decade or so medium access control protocols have been designed more towards multiple hop wireless networks. Mobile ad hoc networks have another desired property to be able to self-organise where the main requirement is to permit hosts to enter or leave the network without any pre configuration in the network like the famous DHCP centralized servers. In the new development of medium access control protocols this self-organization function is added in a distributed manner rather than traditional central fashion, we can say that this type of network is decentralized where the network activity of discovering the topology and delivering messages is performed by the mobile nodes themselves i.e. the routing capability is incorporated into the mobile nodes. Mobile ad hoc networks can also be used for low cost communication as the setup cost of fixed wireless infrastructure becomes traditionally more expensive reflecting in high costs to the end users. Infrastructure less MANETs can use the local communication instead of large distance wireless communication so as to reduce the energy consumption of mobile nodes. In any country the mobile wireless networks are also controlled and regulated by the government or state over some licenced frequencies bands. Adhoc wireless networks however could be made operational using the unlicensed frequencies and overcome the regulation problem and allow people to be able to communicate so as to not be controlled by any government body. It is also seen in less populated areas or villages that the wireless service is less ubiquitous compared to urban areas or cities where there is large provisioning of fixed wireless services and wireless communication is omnipresent, therefore we see that the far reach poor areas or less income regions suffer and they lag behind with the most needed mobile services. Routing protocols are affected by the mobile conditions of mobile hosts or nodes. If we look into traditional networks the routing handover procedures are taken care off very well for node mobility, whereas due to lack of infrastructure, the hosts or mobile nodes must perform handover or routing inside the mobile ad hoc networks while tackling rapid topological changes, further the solution to take care of these rapid changing networks should be designed to provide optimum quality of service without any central administration. This thesis is mainly focused on the routing aspect of MANETs,

particularly characteristic based routing inside mobile ad hoc networks. In this thesis we will look into applications for wireless ad hoc networks that provide access to services. We will look into more detail further in the chapters where the design is based on anycast where data messages are directed towards their destination sources which announce certain features or characteristics (service discovery).

2.2 Wireless Communication

There has been a lot of work done exploring the characteristics of low power radio link in sensor networks and designing and implementing different mechanism for routing data in order to improve reliability. In my dissertation topic, I review related work about link characteristics and different approaches to improve performance like wireless broadcast advantage, multiple path routing e.g. gradient based routing, hop by hop retransmissions etc.

This section provides some overview of wireless communication and discusses the design principles and the wireless 802.11 standard technology, how it is widely used in the wireless telecommunication world today. Further in the next chapter we will discuss about the mechanism of gradient based routing in wireless networks.

Wireless communication has been around with us for over hundred years, since Guglielmo Marconi invention of long distance radio transmission and a radio telegraph system setup in 1897. Further in 1901, radio reception across the Atlantic Ocean had been successfully established. In these last hundred years, wireless communication has become one of one of the most vibrant areas in the field of communication today. In the last couple of decades there has been a tremendous increase in demand of tetherless connectivity for cellular telephony and in recent years a huge demand in wireless data applications. There have been a number of cellular systems designed and implemented.

One such system is the global system for mobile communication or widely known as GSM, it is a second generation cellular digital system. Second is the TDMA (Time-Division Multiple Access) cellular system developed in America, and the third is CDMA (Code Division Multiple Access). These systems were all developed for wireless telephony. Third generation cellular systems are developed to handle both data and voice. As some of the third generation systems have basically evolved from

the second generation voice systems, some other cellular systems are designed from scratch in order to take care of the specific characteristics of data. As the demand of wireless data applications requires higher data rates, these application have another two features that distinguish them from voice. Most data applications are highly bursty, as users quite often could remain inactive for a longer period of time, however they require very often high demands for shorter period of time. In contrast voice applications however have a fixed rate demand over longer period of time.

Voice applications have relatively tight latency requirement of the order of 100ms, whereas comparatively data applications have wide range of latency requirements. One data application example like the real time gaming application could even have more stricter delay requirements than voice applications, however many other data application such as file transfer applications have much less stricter latency requirement.

Beside the cellular telephony systems, their exist other kind of wireless systems like the AM radio, FM radio and TV broadcast systems. These systems behave in a similar fashion to the downlink part of cellular telephony networks; however the data rates, frequency and the sizes of the areas covered by each broadcasting site are very different. Another such systems are the wireless local area networks knows as LANs. These are designed for much higher data rates than cellular systems, but behave in a similar way to a single cell deployed in a cellular network. Such local area networks are designed to connect portable devices such as laptops into the local area network of an office building or similar such environment. Such local area network systems have the major function to provide portability, rather than mobility, there is less mobility expected in such systems. The major accepted standard for wireless LANs is the IEEE 802.11 family standard. There are other smaller scale wireless standards such as Bluetooth and the ultra-wideband (UWB) communication which reduces cabling in the office and makes it easy and simple for data transfers to occur between the office and hand-held devices. Furthermore there is another type of local area network called an ad hoc network. In this type of ad hoc network the data traffic flows through all the nodes alike rather than some central node (or base-station). This kind of network manages and organizes links between various node pairs and maintains routing table entries for each of these links. There is a lot of research going on for ad hoc networking to tackle the problem of distributed cooperation and relaying traffic among nodes, the research suggests that this problem could be tackled in the node physical layer.

2.2.1 802.11 Wireless LANs

802.11 WLANs are widely spread throughout a lot of network deployments these days, mainly because they are easy to implement. From the users perspective, they function and behave similar to a shared Ethernet LAN (Wired). 802.11 architecture is complicated because it becomes hard to control the medium and the challenge is more complex than that of the controlled wired Ethernet medium. 802.11 devices can not sense collisions, where as (CSMA/CD) based devices can sense collisions. As a result, 802.11 family standards provide more robust and scalable Medium Access Control (MAC) which works with minimized overhead. 802.11 based networks are flexible by design. 802.11 networks are flexible by design; they support three types of WLAN topologies:

1. Independent basic service sets (IBSSs)
2. Basic service sets (BSSs)
3. Extended service sets (ESSs)

Service sets are devices that are logicaly grouped. Wireless local area networks work by transmiting signals over a radio frequency channel. The transmiter before transmiting the signal attaches a service set identifier called the SSID to the broadcast signal, which is then used by the receiver to filter and listen to only those signals which match that particular SSID and ignore the others.

2.2.2 Medium Access Control

It is easy to detect collisions in a wired Ethernet medium, because if multiple stations are transmitting at the same time then the signal level on the wire increases, whereby this increase in signal level indicates to the transmitting stations that a collision has occurred. In the 802.11 wireless scheme, stations do not get this capability. It is therefore required for the 802.11 access mechanism to make extensive effort to avoid collisions. One such collision avoidance technique is knows as Time Division Multiple Access (TDMA).

2.2.3 TDMA

In the TDMA protocol, set of [N] stations share the same radio channel frequency, but each station uses the channel only in predetermined slots. Inside this protocols,

the TDMA frame consists of [N] slots, one for each station and the frames are continuous. Therefore the transmission bandwidth is [N] times the bandwidth that would be necessary to accommodate a single user or a mobile station. Importantly Time Division Multiple Access (TDMA) is combined with Time Division Duplex (TDD) where transmission and reception do not happen simultaneously, but at different time slots. This eliminates the need for expensive duplex filters. TDMA is simple to implement in a downlink i.e. base to mobile, it is done by multiplexing [N] user signals. However TDMA is more difficult during the uplink i.e. mobile to base, here the frequency signals have to be aligned in time coming from all the stations. For this fast power-up and power-off times are required so that signals from users do not interfere with those signals in the other time slots.

2.2.4 Broadcasts

Network communications most fundamental task is broadcasting. During broadcast a single node of the network is known as the source node which transmits a message to all other nodes inside the network. Further intermediate nodes inform the remote nodes via directed paths in the network. A radio network can be modeled as a graph whose nodes are stations or sensors that can both transmit and receive data. The nodes sends data packet or messages in synchronous rounds. In each and every round, a single node acts as either as a transmitter or a receiver. The transmitter node sends a message to all its out-neighbors, similarly a single node receives a message in a given round only if the particular node acts as a receiver and exactly one of its in-neighbors transmitted a message in this round. Also if at least two in-neighbors of a receiving node R transmit simultaneously in a given round then none of these messages are received by R in that particular round, hence we can deduce from this that a collision has occurred at R. Also in a radio network if nodes can differentiate collision from silence then we can easily say that collision detection is available. In a radio network a set of nodes i.e. stations or sensors are modeled as points of a plane. Each node N has a range rN which depends on its transmitter power and it can reach out to all the nodes at a distance which is at most rN from it. Here the collection of nodes that has ranges determine a directed graph on the set of these nodes which is called a geometric radio network (GRN). All of the node ranges are equal if their transmitter powers are equal and therefore we say that the geometric radio network is symmetric. Radio network model is applied to wireless networks only in a single frequency. The model of GRN is applied to wireless networks where the stations or nodes are present

in a flat region and without any large obstacles. Here, the signal of a transmitter is reached by receivers at the same distance in all directions, hence we can deduce that the set of receivers of a transmitter is a disc.

2.2.5 Mesh Networks

We can say today that the worlds largest mesh network is the internet. Information packets proceed in the internet by getting bounced from one router to the next router and so on until these data packets reach their destination and it all happens automatically. Therefore we can call the internet as a web or sometimes a cloud of connectivity since their exist billions of possible route or paths through which data packets can travel. Often these wireless mesh networks (WMNs) are a type of radio based network which require minimal infrastructure and configuration, also they can be built using low cost radio and computing platforms. Mesh networks can easily connect wirelessly entire cities using the inexpensive and existing technology. Traditional common network rely on small number of wired access points (AP) or often known as wireless hotspots in order to connect users, where as if we look into the wireless mesh networks, there are even hundreds of wireless mesh nodes that are spanned across that communicate or "talk" with one another to share the network connection across a large area. These types of networks have become an emerging technology and one day could make the dream of smoothly connected world a reality.

Mesh networks are basically multi-hop wireless networks that provide internet access to mobile users. These types of networks have been deployed on top of house rooftops in numerous cities in America and Europe. Only a few nodes among the buildings or houses have internet connection over fixed infrastructure like DSL cable lines and then the rest of the other nodes or houses simply act like relays for the internet connection. Mobile users using their devices (like laptops) take advantage of this mesh network to connect to the internet via nodes with internet connection. Typically in these mesh networks anycast routing is a natural fit to route data packets, since these data packets can be easily sent across through the internet via any node which acts as the gateway. The nodes pick up the quickest, safest route and this process is called dynamic routing. Nodes in a mesh network use the common wireless standard or WiFi also known as the 802.11a, b and g standard to communicate or talk with users wirelessly, but more importantly with each other wirelessly.

Nodes, in order to work in a larger network, are programmed with software that help them interact. Information travels from point A to point B through the network

by traversing wirelessly from one mesh node to the other. As compared to wired or fixed wireless networks, the biggest advantage for wireless networks is that they are genuinely wireless. Most wireless access points are still needed to be connected to the Internet using wires to broadcast their signal. Ethernet cables are laid on ceilings and walls and throughout public areas, for large wireless networks. Only one node is required to be wired to internet. This wired node shares its internet connection wirelessly with other nodes nearby. Similarly, these nodes share their internet connection wirelessly with other nodes nearby, and so on so forth. The more the nodes are, the further the connection can spread, that can create a wireless connection that can operate in a small office or a whole city. The advantages of wireless mesh networks are : Lower costs to set up a network, as fewer wires are required. The network becomes faster and bigger as more nodes are installed. They use the same Wifi standards that are already being used by most networks. They are useful where Ethernet wall connections cannot be used, for example in outdoor concerts, warehouses, transportation settings, etc. They are also useful for Non Line of Sight (NLoS) networks where wireless signals are irregularly blocked. For example, in parks, rides may, at times, block signals from a node. If there are more nodes around, the mesh network will automatically make changes to find a better signal. There is no need for a network administrator to integrate a new node into the existing network, as mesh networks can adjust the new node automatically attributing to self healing. Even if the nodes losing their signals or are blocked, the mesh networks can automatically find the fastest and most dependable path to send data. The wireless mesh networks configurations help the local network to run faster, as local packets need not to travel back and forth to server. Mesh nodes are easy to uninstall and install, which makes such network extremely adjustable and expandable.

2.2.6 Vehicular Ad-hoc Networks VANETS

Figure 2-1: Vehicular Ad-hoc Networks [17]

Vehicular Ad-Hoc Networks are a subset of mobile ad hoc networks, which supports data communications among close vehicles in proximity and the nearby fixed infrastructure also called the roadside entities. The basic idea behind all this is to aid the driver of the vehicle with useful information that he or she cannot obtain on their own. Also depending on the range of communications devices, vehicles or nodes in this type of ad hoc network communicate among themselves in type of short-range (vehicle-to-vehicle) or medium-range (vehicle-to-roadside). Some typical information could be about traffic jams ahead or about car accidents or even the weather conditions, VANETs are also used in guiding the driver to find a free car park space in any area of proximity i.e. the closest car park space available. In addition to this, major applications of VANETs include safety and real-time applications like real-time traffic jams and routing information, high speed tolling and many others. Some of the vehicular safety applications are collision, car accident, emergency and other safety warnings.

However their still remain several challenges in achieving scalability, high performance, robustness, fault tolerance and secure vehicular networking, some of them mentioned are as follows:

1. Availability, Scalability
2. PHY, MAC, Network Layer
3. Mobility, Traffic models
4. Security, Privacy
5. Cross-layer optimization techniques
6. Vehicle To Vehicle
7. Vehicle To Roadside

Chapter 3

State Of Art

The objective of my thesis is to investigate how successful data delivery can be achieved using characteristics of nodes for efficient routing inside MANETs. As we have come to know that wireless ad hoc networks are challenging in many ways like the wireless links and the mobility of the nodes increases the dynamism of wireless ad hoc networks and the infrastructure less property of these networks makes any centralised solution not applicable. Over here we will survey and discuss some routing approaches and then look into the most common protocols for MANETs in the following sections.

3.1 Gradient based routing approach

3.1.1 Overivew

In scientific terminology, it is laid out by researchers as; "Every physical event results in a natural information gradient in the proximity of the phenomenon" [3]

It is this natural information gradient which can be used to design efficient information driven routing protocols for ad hoc networks. If we take the example of "WSNs", they comprise of distributed sensors in an area of space to monitor the environmental or physical conditions and to collaborate with each other to pass information through the network to a main location. Some characteristics of the sensor nodes like for instance limited battery life, energy expensive wireless communication, loss of wireless communication, high risk of failure or malfunction and unstructured nature of the wireless sensor network "WSN" all of these make routing in the WSN a challenging problem. In these kind of networks, data of interest should be passed

to the application in an energy efficient and timely manner according to the protocols that do not waste much radio transmissions. It is of view that different kind of wireless applications may require different flavor of communication protocol or strategies in order to maximize the wireless network lifetime. A typical such application could be a data gathering application such as temperature or light sensing. In a data driven network model, an application could instruct each sensor node to sense its environment parameters at certain rate and for a specific period of time and then send matching data back to the sink. Here the sensor nodes are called sources and the sink is a node with high computing power and resources whose job is to analyze and process that data. Data can be transmitted from the sources to one or more sinks, hence we say that the network traffic is composed of several packets that flow around periodically between many sensors and one or more sink nodes.

Because of the recent progress in Wireless Sensor Networks (WSN) the interest has increased the possibility of the use of applications such as security surveillance, border protection, disaster management, etc. To attend to the environments that are unsupervised, unaccompanied or neglected, sensors become handy, which are supposed to be positioned remotely in large numbers that function independently.

3.1.2 Basis of Gradient based routing

Gradient concept inside routing protocols is used to describe a network layer augmentation where every node is associated with a potential value with respect to the sink. The sink being the root node. In such an overlay network the packets are routed by decreasing the potential value 'Q' at the next node inside the route path i.e. the packets are forwarded to those neighbouring intermediate nodes that contain a lower potential value with respect to the sink node. Such gradient based routing protocols employ a spread mechanism for collection of information of data inside mesh networks where the source node, spreads its gradient throughout the nodes inside the network. This flow creates a directed acyclic graph in which every participating node member has an associated value with respect to the sink also known as the root node. The sink is finally reached by following the direction of this acyclic graph. This flow process is also known as a "gradient descent". Such an idea or concept of tunnelling the data towards source node or nodes with higher capabilities or features is frequently employed inside ad hoc networks for pushing data across outside the network.

Following figure illustrates the gradient broadcast example towards a sink.

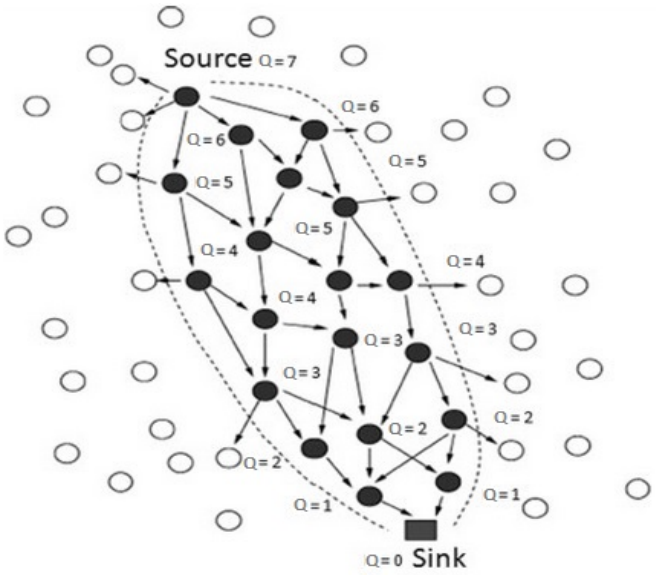

Figure 3-1: Gradient Broadcast [22]

3.1.3 Gradient Broadcast (GRAB)

GRAB is a gradient based routing protocol [4] that has a robust forwarding mechanism utilising efficiently multiple number of paths when descending the gradient. It builds and maintains an energy "cost" field, which gives every sensor the directed path to forward sensing data. Participating nodes in GRAB contain an estimation of this energy cost to forward the packet to a neighbouring node. This kind of cost gradient is initialised during the setup phase taking the measurement of Signal to Noise Ratio and eventually storing energy cost towards the sink at every node. Inside GRAB, data packets are forwarded over multiple paths adding to the reliability of the protocol and the interleaving of these paths provides fault tolerance along any single path. GRAB forms a band of nodes carrying the transmission from the source towards the sink, it controls the width of this band to increase the delivery ratio, this is controlled via a credit based value which is added to the energy cost as extra budget. A source node broadcasts a packet over the width of the band and towards the sink only if its energy cost is less than the total of the energy cost of the packet and the credit value from the source node. GRAB relies on the collective efforts of multiple nodes for data delivery, without the dependency on any individual ones.

3.1.4 Directed Diffusion

Directed diffusion [2] is data-centric i.e. all the communication is for some particular named data, thus this sensor data is named and it diffuses according to some pre-established gradients of interest towards the data sink. Here the entire directional diffusion based network is application aware. One feature of directed diffusion is its ability to aggregate data which is done according to the interests of the nodes, therefore we can say that it supports multiple destinations keeping in view that the main goal of this system is to deliver data to all of the interested nodes. In my work simulation we will provide the mathematical model as we carry out in this thesis in the next chapters.

3.2 Routing using potentials

For routing protocols it is very important to avoid congested areas in a network, therefore in 2003, A.Basu designed a routing algorithm [7] that was dynamic for the internet and was traffic aware. This was designed by actually modelling the routers queue length being the potential field, which was taken as the real input and used these potential values for this routing algorithm. In research topic, the author mentioned, described and evaluated the complete model for his potential based routing. In the tests and analysis for network routers, it was revealed that steepest gradient routing resulted in identifying routes via the routers that were less congested. Although this work very much focused on unicast routing in wired networks that are fixed, In my thesis I will utilise the source capacity for characteristics based routing in MANETs.

3.3 Anycast Routing Protocols

Anycast is a bit like making announcements of same network in different subnets inside a network so as to reduce the network hops necessary to reach that particular network, it uses a delivery mode in which a packet is addressed to one member within a group of hosts. We can say that anycast is used in the IPv6 6to4 transition protocol. In this particular section, we are going to discuss the research efforts carried out around this type of anycast routing protocols for the Internet and also for the mobile networks.

3.3.1 Internet

Inorder to make it easier to find a server among many, which are supporting a desired service, an Request for Comments (RFC) for the internet was proposed in 1993. This particular RFC proposed that multiple hosts could be assigned with an unicast IP address advertising this unicast address into the routing infrastructure from all the hosts which shared the same IP address. This scheme was later added into the addressing architecture of the IPv6 protocol. There are two main problems with IP anycast for not being widely deployed and used in todays internet. Mainly it is hard to deploy in the core of the internet at a large scale and also it does not scale well. A proposed GIA solution by Dina Katabi as a IP architecture that is scalable was tried, however in this proposal a change to the core internet routers was necessary and a main requirement which defeated the practical approach. Further many researchers looked into application layer solutions (proxy based approach like PIAS - deployed as a global anycast service) that could be adhered to scalability and which did not require much change to the routing and core internet infrastructure. Although it could be technically smooth and straightforward (but may be more expensive) for the ISPs to support a new IPvN protocol release, however they must have a choice and incentive to do so. It is quite evident that scalability and the ease of deployment are fundamental for the success of anycast routing protocol in the Internet, however it is less important for infrastructure less networks like mobile, ad hoc networks, or wireless sensor networks which is the main focus of this thesis. Here we mostly expect such type of networks to be less dense or smaller (may be hundred to thousand nodes as compared to perhaps millions of nodes in the internet) and also to be mainly used in local communications like within city or may be among a group of people. Such networks which function as self-organizing more likely will not rely on existing routing infrastructures (like the huge internet), making the problem easier for our protocol deployment.

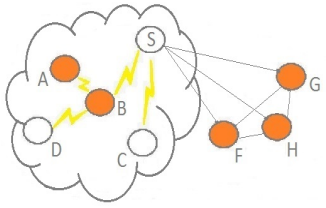

Figure 3-2: Service based topology for MANETs - S: Service provided by A, B and F, H, G: Service requestor.

3.4 Service Discovery Architectures

Service discovery architecture mainly depends on either having a directory or not having one. The directory is basically a store that stores information about services that are available inside the network, therefore the architecture for a service discovery protocol can be directory based or it can be directory less or it could even be a hybrid of both. MANETs have been setup using all such architectures, Following are the architectures that could be adopted for an approach to service discovery;

1. Service coordinator (directory based)
2. Distributed query based (directory less)
3. Hybrid of both

Service coordinator scheme is very similar to a directory based, which in fact is like a central directory and certain nodes in ad hoc network are dedicated service coordinators which act in the role of a directory agent like in SLP or the JINI lookup service, UDDI (Universal Description Discovery and Integration). These kind of service discovery approach fair better for networks that are infrastructure based and where the topology does not change frequently. They are not much suitable for MANETs where we see that the system topology is changing all the time due to node mobility. Directory based architecture is further classified into a central directory architecture and a directory based architecture.

On the other hand, distributed query based architecture is not centralized and is directory less. It has no service coordinator node. In this type of MANET the clients always flood the service discovery requests throughout the network neighbourhood. which results in an extra overhead due to flooding of requests and it consumes a lot of battery power, bandwidth and computational resources. Service location protocol and universal plug and play are a few examples. It is seen from surveys that directory less approach is more used inside MANETs than the directory based approach.

Hybrid architecture is a mix of both directory based and directory less architectures. In this architecture servers may wait for service query requests from clients or they might register their services with a directory agent when available, clients could then post service queries to directory agents if available or straight away to service providers by flooding.

	Directory-based	Directory-less
Overlay	• Service Rings [12] • Lanes [13]	• Allia [8]
No overlay	• Splendor [11]	• GSD [9] • Konark [10] • DSD [14]

Figure 3-3: Service discovery protocol architectures

3.4.1 DNS based service discovery

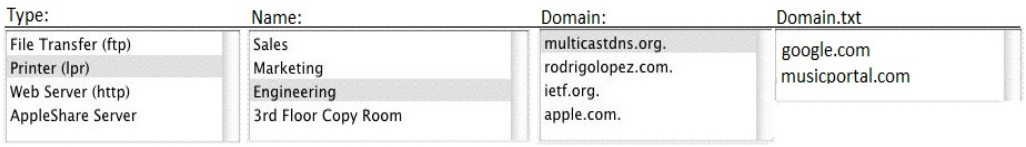

Figure 3-4: DNS Service Lookup

A DNS-SD is a traditional service discovery mechanism, where the nodes acting as a potential resource for a service like printer, ftp or some share service are named and organised into a structure, in order to assist the client for service lookup or discovery. When the client provides the type of the service and the domain, the DNS query mechanism gives back the list of named instances by searching from its database or other DNS lookups.

3.4.2 Service Discovery

In the following section we will look into the solutions for service discovery that work at the application layer and compare them with the solutions that work at the network layer which combine with the routing tasks as well.

3.4.3 Application-Layer

Just a couple of decades ago, in the late nineties, there started emerging a number of industry standards for service discovery. As service discovery requires common language to allow software agents to make use of each other's services without the need

for continuous user intervention, a lot of these standards incorporated XML as the common exchange format. Microsofts introduced XML based UPnP universal plug and play services, IETF put effort in bringing the SLP (service location protocol) and standardised the service discovery framework for the big internet. Researchers at Berkeley using the XML language also developed a framework for service discovery and put forward a secure architecture. XMPP, WS-Discovery, SAP (Session Announcement Protocol), SSDP (Simple Service Discovery Protocol) are a few more example in this area. Suns JINI framework for spontaneous distributed computing has also taken mobile devices to a new level and make possible for small devices to quickly discover services in proximity. Although these service discovery protocols or standards are not fit to work in mobile ad hoc networks. On the other hand a new service discovery and delivery protocol was designed and implemented for peer-to-peer networks, more importantly to deliver m-commerce oriented software services, this protocol was called Konark. It used a tree-based structure for storing services and its purpose was mainly for registration and advertisement of services on a particular mobile device, and also the discovery of services on other devices in the environment. It used XML based approach for the service description, but the service delivery itself was SOAP based. Since Konark design took the application layer approach, it still assumed an underlying routing substrate.

3.4.4 Network-Layer

If we look into the popular Bluetooth technology, its specification has got a definition for service discovery protocol (SDP) in order to be able to help for the discovery of services a device has to offer. This came into being from the early Intentional Naming System (INIS) that was proposed in 1999 at MIT by W.A.Winoto. In their approach the particular service requests coming from the clients were directly routed towards the matching service instances even without doing a proxy or intermediate lookups to find out the particular service instance address. They proposed a resolver network which contained a list of dedicated INS resolvers in order to properly route all the service requests. Their work focused more on the design of a naming architecture. This thesis work tends more towards the routing of service queries and selection of a specific service instance with a characteristic. Also INS was developed for wired networks where as we are going to take the approach of the wireless channel using its broadcast nature. Koodli formulated in 2002 in an RFC the proposal of adding the service discovery process inside the MANET routing protocols themselves. His

idea was to bundle the service information directly inside the route request messages which was further incorporated into the on demand ad hoc routing protocols like the DSR and Ad hoc On-demand Distance Vector (AODV). There was a draw back in this design that it introduced flooding inside the network by each and every service request. The next step taken was to incorporate a novel service discovery mechanism inside the mobile ad hoc networks. For this a more dynamic approach of creating a virtual backbone was introduced which assured that all the host nodes become a part of it or they are located at least at one hop distance from the backbone. However in this model it is not possible to make the service selection effective and it shows discrimination as it becomes hard to distinguish when there are multiple service instances available which are of the same type.

3.5 Network Architectures

We aim to come up with a new architecture for mobile ad hoc networks, the main idea and plan will be to figure out how to transmit heterogeneous traffic over radio wireless links efficiently.

1. Applications: tactical ad hoc networks, wireless Internet access, sensor networks.

2. Rquirements: low-bandwidth, high-definition video, voice, data.

3. Target: communication-on-the-move in dynamic network space, which is reliable (QoS)

There is lot of literature that has come up with proposals and solutions for the architectural fix for the internet protocol which we have mentioned above including routing, naming/addressing, security, quality of service (QoS) are some of them, all of them tend to choose of improving the current ISP infrastructures. The solution lies at the application layer which act as overlays, but this solution adds a significant stretch to the path length and is not found to be good for wireless ad hoc networks. Another approach of cross layer design is a good one and is more efficient. In the cross layer design approach there are misconceptions that in this design we need to get rid of protocol layers and to integrate all of them, rather in this design we take a more holistic view of wireless networking. With the cross layer architecture we can use information across from the application layer then at the routing layer to form a strictly layered architecture which fades away from the path length stretch. This

type of architecture does preserve the integrity of the IP architecture simultaneously also allowing for the particularities of MANETs which we require. There is also a sensor network architecture that abstracts the communication to a single hop rather than multiple hop as observed with the IP architecture and this sensor network architecture deals nicely with the addressing schemes and also with the diversity of different semantics. This is a trait of Publish-subscribe system which is a paradigm that totally decouples communication as the destination address is not required and also the components involved in the communication process may not be online simultaneously. Publish-subscribe architectures have been widely studied and applied in wired networks, but their deployment on mobile ad hoc networks still presents a lot of challenges. Publish-subscribe architectures like TIB / RENDEZVOUS have processes that could subscribe to only those network messages that contain information on specific subjects while the other processes publish such network messages. Therefore, we can ascribe our characteristic based routing protocol similar to publish-subscribe system, where service providers use a characteristic potential field to subscribe to any clients subscribe requests. We are going to take the advantage of the broadcast nature of wireless network where usually such publish-subscribe systems are built like the overlays over IP. We can figure out in the later development that how such internet architecture can get more evolved and deployable.

3.6 Mobile Networks

It has been described by Vincent D. Park and Joseph P.Macker in their paper [1] of how to extend unicast routing techniques like distance vector routing and link state for anycast routing for data packet delivery. Similarly, J. Wang in his paper [5] proposed techniques of how to extend the DSR and AODV protocols in order to support anycast delivery for MANETs. All these protocols use the same routing strategy i.e. to route data packets to the nearest member within a group over the shortest route.

3.6.1 P2PNET

P2PNET is their to support the mobile group users for information demand and communications likely for any disaster management, it is used as a good disaster management solution for emergency communication and it is also based on a serverless p2p based MANET network which is their to support for a temporary group

communication. Its real world usage is for example in the army battle field, rescue teams in any catastrophic disasters and also in mobile learning groups. It has the flexibility for optional nodes which can act as satellite gateways for the possibility of other nodes in the network to connect to the internet when the nodes are available, here communication with the satellite are not part of managing for the routes in the MANET nodes, P2PNET communications system is based on the walkie talkie like ad hoc communications model.

3.6.2 DUMBONET

DUMBONET is another emergency communications system which uses a VPN (virtual private network) to hide the heterogeneity of the network and also uses direct satellite links to setup connection between the command headquarters and the diverse and far reaching disaster sites, thus exploiting the satellite communications to tackle the partitioning of mobile ad hoc networks. It is analyzed in a test simulation environment which connects remote site via geostationary satellite like IPSTAR, the basis of the test representing a VPN where all the devices or nodes use the same private subnet and Optimized Link State Routing (OLSR) routing protocol for routing the data traffic among devices or nodes. OLSR is good in this case because using this protocol finds no difference between the satellite links and terrestrial connections, therefore the simulation achieved successful data transmissions between the disaster areas and the remote sites. Further proposals are made for the OLSR routing protocol to extend with link characteristic awareness which is a good add on.

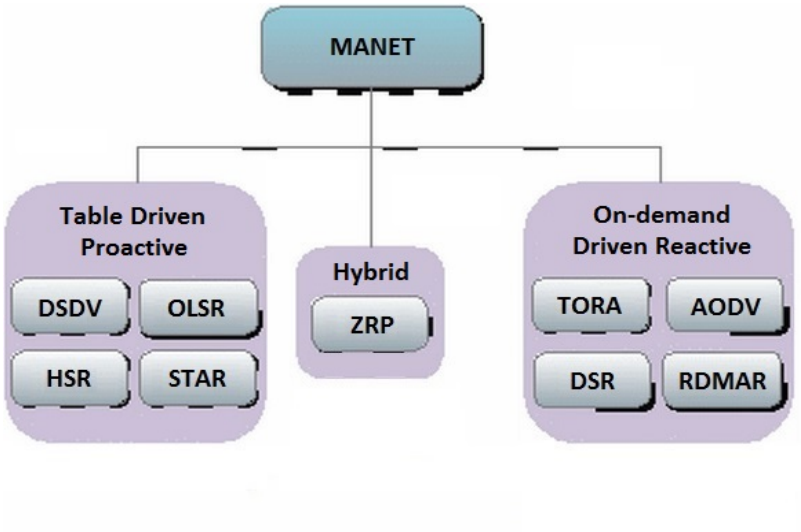

Figure 3-5: Classification of routing protocols in MANETs

3.7 MANET routing protocols

MANET routing protocols show that the mobility of nodes becomes a constraint and it yields a network that behaves dynamic in a sense that the links among the nodes remains temporary i.e. if we consider a particular node desiring to send a data packet to some other node which lies outside the sending nodes radio range, then it becomes a separate requirement to use some intermediate node which can act as a multi-hop link to that particular sink node as the final destination. Hence all the nodes in a MANET should have the same routing capabilities inorder to determine which particular node is the actual next node in the path or route to reach the final destination node. In reality this routing task is not a trivial one because of the nature of temporary links seen in these type of dynamic networks and majority of traditional routing protocols do not fair better as a solution. Some good efforts in this regard have brought up several solutions and new concepts which were adapted from the earlier traditional protocols. We will investigate further in the list of these protocols and look into a different routing technique which is based on characteristics or properties of a node.

3.8 Proactive Routing

Proactive routing is precomputed routing. In this type of routing each and every node is bound for maintaining the global network topology information by periodically exchanging and updating routing information which is stored in the form of tables in order to maintain consistent and accurate network state information. It therefore maintains paths or routes to all the destinations in order to proactively respond to any kind of network topological changes. This routing information is usually flooded inside the whole network. Thus whenever a particular node wants to send information packets to a destination, it executes an appropriate path finding algorithm on the topological information it maintains. These routing protocols are seen as an extension to the wired network routing protocols. Traditionally it has been analysed that proactive type routing is not suitable for mobile ad hoc network routing, this is because the dynamics of such networks follows constant updates which adds to a significant increase in the overhead. A number of solutions are presented and discussed next.

3.8.1 Destination-Sequenced Distance Vector (DSDV)

DSDV protocol is derived from the traditional routing information protocol also known as RIP, it is actually one of the first protocols that was designed for the mobile ad hoc wireless networks. This protocol belongs to the distance vector family of routing protocols and it is based on an extended version of the distributed Bellman-Ford algorithm in which every node stores a routing information table that stores the shortest distance and also stores the first node in the shortest path to all other nodes inside the network. In DSDV protocol a sequence number field is also added into the routing information table which differentiates between the routes that are stale and to those that are fresh. It also incorporates a mechanism of table updates with incrementing sequence number tags in order to prevent the loops and also for the faster convergence, therefore we can say that routes or full paths to all the destinations are constantly available at each and every node inside the network at all times. The mechanism of table updates are of two types, incremental updates and full dumps. Here the later includes all the routing information that is available while the former includes just the information that is changed since the last full dump. DSDV is not found suitable for mobile networks that contain large number of nodes and where the mobility is high among the nodes, this is because in this case the signalling cost or amount becomes high in the order of O(n2) where n is the number of nodes in the mobile network. If we look into the computation of the DSDV routing algorithm, here the nodes inside the network maintain table which contains the next hop and the distance to each destination node, all this information is periodically exchanged and sent to the neighbouring nodes, when the node receives a distance table from a particular neighbouring node, this node then calculates the shortest path or route to all the other nodes in the network and updates its own table. In the beginning of DSDV routing algorithm, any particular node will know only the distance to its neighbors, subsequently this information spreads node by node across the network until the routing algorithm converges and the routing information tables are fully prepared and reach complete state.

Advantages and Disadvantages

It makes the availability of routes to all destinations at all times making the delay in route setup phase minimum. However as updates are necessary to maintain an up-to-date view of the routes, the updates get propagated through out the network, which can lead to a control overhead or even worse does not scale very well especially

when links are broken i.e. it can choke the available bandwidth during high mobility, therefore, it does not scale very well.

3.8.2 OLSR

OLSR is part of the link state family of routing protocols and is a proactive routing protocol. It follows an efficient packet forwarding strategy called multipoint relaying which tackles the overhead issue of proactive routing protocols. The strategy reduces the size of the signalling messages and also the number of retransmissions in a broadcast. In this protocol, each node in the network maintains the network topology by periodic broadcasting the pure link states. Multipoint relaying employs an efficient link state packet forwarding scheme and the protocol optimizes the link state routing protocol. This optimization is performed in two ways, firstly by reducing the control packet's size and then secondly by minimizing the number of links that are utilized in forwarding the link state packets. The computation of the routing algorithm employs the shortest path algorithm to the received topology information in order to determine the routes or paths to all the destinations inside the network. Inside the network which uses OLSR, each node will select a set of neighbours as their multipoint relays in such a way that these multipoint relays must cover all its two hop neighbours. This is because to allow the multipoint relays only to retransmit messages when receiving broadcast messages from a node. Their are two main processes or methods inside the OSLR protocol regarding the signalling which are the link sense and the link state distribution. Here the link sense method utilises the local exchange of HELLO type messages for the nodes to gather information about the links which lie at most at a distance of two hops. After obtaining this information the protocol's topology control messages are then broadcasted inside the entire network which then takes advantage of the multipoint relaying topology.

Advantages and Disadvantages

It has an added advantage of having a low cost connection setup and also a reduced control overhead.

3.8.3 Hierarchical State Routing (HSR)

HSR is a multilevel clustering protocol which performs the logical partitioning of mobile nodes. First network is divided into clusters and then the protocol chooses a cluster-head by applying a cluster-based algorithm. The cluster-heads together can

also form other clusters. This link information is broadcasted between the nodes of a physical cluster. This information is summarized and sent to nearby cluster-heads via gateway. They exchange their link information along with the summarized lower level cluster information between themselves and so on. A node at each level floods the information it obtained, after the algorithm has done its job.

Advantages and Disadvantages

This protocol reduces the size of the routing table by using the hierarchical information. Although the reduction of size of the routing table is good, but the control overhead involved in exchanging packets that hold information about the levels of hierarchy makes it less affordable in the context of ad hoc wireless network.

3.9 Reactive Routing

Reactive routing is also known as on-demand routing i.e. routes prepared on demand, therefore, protocols that fall into this category never have to maintain the routing path information of the entire network topology, this type of routing minimizes the extra overhead that is seen inherent inside the proactive routing protocols, this is because reactive routing protocols have to maintain information for active routes only. Since they gather the necessary path when it is required, routes are obtained for the nodes that require to send data packets to a specific destination node by a connection establishment process and thus do not exchange any routing information periodically. Inorder for a node to send data to another node inside the network, that node first broadcasts a routing request to gather and setup the particular route. Once this route is discovered and setup finally then the protocol performs a bit of route maintenance for keeping the active routes updated and also follow or react to the routes that have broken because of the link outage observed due to the node mobility. These procedures are carried out by the protocol to save network bandwidth and also to avoid any unnecessary calculation of the unused and stale routes, but however the process of sending out the route request first adds to the route acquisition latency i.e. even before sending out any packet to a particular destination node. If we look into the proactive protocols, the nodes inside this category do not get affected by the route acquisition latency, this is because, in proactive routing protocols, they gather and maintain all the distance paths to all the possible destinations all the time by employing the HELLO type packets or broadcasts. A number of reactive routing

protocols also differ from each other in the way they do the route discovery and route maintenance, we will look into these next.

3.9.1 AODV

AODV has been developed as a result of the IETF (International Engineering Task Force) best effort inside the MANET group and is now standardized since the earlier presentation of RFC 3561. This routing protocol utilizes an on-demand approach in finding out the routes or paths inside the network.

It's important main tasks are:

1. Only to broadcast route finding 'RouteRequest' messages when necessary.
2. Differentiate between the neighbourhood identification also knows as local connectivity management from the general topology maintenance.
3. Pass on the network change information inside the local connectivity management to the neighbouring nodes which are mobile and more likely to need this change information for path identification.

AODV establishes the routes or paths only when it is required by any source node inside the network to transmit the data packets, for transmitting those data packets it uses a destination sequence numbering scheme to find out the latest and updated path. AODV employs three processes to perform its main functions or tasks, one being the route discovery process, second is the route maintenance process and finally the local connectivity management process. The node initiating the issuance of data starts with the route discovery process. It first of all starts broadcasting the RouteRequest packet or message to the neighbours in order for building up the correct path or route to the destination node, this RouteRequest comprises of the source identifier (SrcID), destination identfier (DestID), source sequence number (SrcSeqNum), the destination sequence number (DestSeqNum) and the broadcast identifier (BcastID). A node which receives the RouteRequest message sends back a RouteReply message or packet only if it knows the route to the destination node which is set inside the original RouteRequest message. Duplicate RouteRequest messages are identified and filtered with the inclusion of a unique BROADCAST ID (or the BcastID-SrcID pair) which is present inside the message. As the RouteRequest message passes on inside the network and disseminates, simultaneously a reverse direction path towards the originating source node is also built and setup, this is to facilitate the sending back of

the RouteReply message as a unicast towards the source node. Also while the unicast is happening, the route to the destination is then stored inside the intermediate nodes which have the forward path built and setup. AODV becomes a hop by hop routing protocol, in the sense that the active routing information is stored inside the intermediary nodes in the shape of destination next hop pairs. If a node moves away or if the active link or route is broken due to mobility, then the route maintenance mode kicks in and starts discovering another route to the node that has moved. The local connectivity management process detects any link outages by using the LLACKs also knows as link layer acknowledgement packets. If the local connectivity management of a node finds out that a link is broken, then it will send out an unwarranted RouteReply message to all the active upstream neighbouring nodes. If including the source node or any node along the path direction or route wants to build the route again to the destination, it will start sending out a RouteRequest message inorder to setup a new route. The function of the local connectivity management process of any node is to maintain the list of neighbours. Inorder to buildup and maintain this list, each node must send out a HELLO packet to the neighbours within a set HELLO packet interval. Network changes are detected by the local connectivity management process in a node if a number of consecutive HELLO message are missed or failed to be received which is usually set to two by default. This process in the node can also be employed to ensure the bidirectional connectivity among the neighbouring nodes.

Advantages and Disadvantages

Advantage for this protocol is that the connection setup delay is less, the disadvantage is that their becomes a possibility of stale and unfresh routes when the source holds an old sequence number while some intermediate node holds a higher sequence number thus leading to inconsistent routes, also it consumes unnecessary bandwidth because of frequent beaconing.

3.9.2 Dynamic Source Routing (DSR)

Dynamic source routing protocol or DSR is an on-demand routing protocol which is related to the AODV protocol. It is designed to control the bandwidth used by the control packets inside the ad hoc wireless networks by getting rid of the usual periodic table update messages utilized inside the table driven approach. With respect to the AODV protocol, the main difference with the DSR protocol is the source routing approach whereas AODV uses the hop by hop routing approach. This means

that in the DSR protocol, the route is stored inside the actual data packets and not like in AODV, where the route is stored and distributed inside the intermediary nodes. Another difference between DSR and the other on-demand routing protocols is that this protocol is a beacon-less protocol and it does not require the transmissions of frequent and periodic HELLO packet or beacon which are commonly used by the node inside the network to inform its neighbours of its own presence. Hence the difference follows to bring some changes to the route discovery and the route maintenance process inside DSR. Inside DSR when it broadcasts RouteRequest for the route discovery process, then during progression of the message inside the network, the node- ids that processed the message is stored inside the header of the message, this is because when it reaches the destination node, it should be sent back to the source node. To do this the hop sequence is piggybacked inside a new RouteRequest message which is then bounced back to the source node. We can observe that for data transmission inside DSR unidirectional links are perfectly compatible, duplicate RouteRequests are easily detectable using a request-id field. Inside the protocol when the RouteReply reaches the source node, it extracts the hop sequence and packages this hop sequence into the header of the data packets sent out to the destination. This important and explicit routing information present inside the data packets and the RouteReplies is then cached by the intermediary nodes. Also, more promiscuous approaches could be utilized to permit other nodes which lie in the transmission range to also cache the same routing information.

Advantages and Disadvantages

It has a reduced control overhead, although it has a higher delay during connection setup. Its disadvantage is that its performance degrades quickly when mobility increases and its routing overhead are proportional to the path length.

3.10 Hybrid Routing

Hybrid routing is classified as the third routing approach inside MANETs. It works by allocating geographic zones to nodes inside the network i.e. the nodes which are located at a particular distance from a node are said to be inside the routing zone of the concerned node. These protocols use a table-driven approach for nodes when working within the routing zone and employ an on demand routing approach for nodes outside this zone. ZRP is another hybrid routing protocol.

3.10.1 Zone Routing Protocol (ZRP)

ZRP is a hybrid routing protocol which uses both the reactive and proactive routing protocol approach, it combines the advantages of both these types of routing protocols. This protocol works on the concept of zones and each node inside the network has a different definition of its routing zone and the zones of neighbouring nodes overlap. Every node's zone has a radius R defined which is calculated in hops and is therefore called a limited zone which means it includes nodes in the zone which are at most R-HOP from that node. ZRP takes the proactive routing approach when working with the nodes inside the R-HOP neighbourhood and it takes the reactive approach for the nodes outside this R-HOP zone. It names its proactive routing approach as intra-zone routing protocol (IARP) and names inter-zone routing protocol (IERP) for the reactive routing approach. Nodes inside the routing zone periodically broadcast route update packets thus trying to maintain routes to all the nodes lying inside the particular zone. Doing so it adds an extra control overhead as the routing zone becomes larger.

Advantages and Disadvantages

ZRP demonstrates a reduced control overhead compared to the reactive and proactive approaches which is a good thing. However as the zone radius becomes larger then the ZRP performance gets impacted.

3.11 Summary

In this chapter, we have gone through a survey of the current state of the art concepts for routing inside MANETs and discussed some techniques involved in gradient based routing protocols. Also, we gave an introduction to the service discovery approach inside ad hoc networks at different layers. In the next chapter we discuss the design of our characteristics based routing protocol.

Chapter 4

Design

The design of our CBR routing protocol for MANETs involves with the task to achieve the successful delivery of information to nodes or endpoints having a strong gradient to specified characteristics and to tackle the frequent and evolving network topology. In different scenarios our protocol works by spreading characteristic advertisement packets which finally gets decimated inside the ad hoc network. The spread of these characteristics advertisement packets simulate the flow of a coloured water stream from a higher gradient or potential towards a low gradient. In this chapter we will do the analysis and design of our routing protocol and discuss the approach.

4.1 Overview

In our design for characteristic based routing, we will take the approach of features of nodes rather than the identities of nodes as the basis for routing of the data flow. If we look into the internet protocol or IP, it provides identity based routing at the network layer inside the OSI architecture model, our aim is to substitute this network layer addressing scheme with our own characteristic addressing terminology. In fixed or wired networks each and every node is given an IP address which hints the location of a network node or the subnet that the particular node is attached to, however in MANETs this kind of addressing scheme becomes illogical due to the dynamic topology where the location of nodes changes frequently, therefore we can deduce that such IP based end to end communication is not very well suited for MANETs rather we can argue that it is the properties or the characteristics of the nodes which are of interest for a number of applications, such as a rapid intervention team member wanting to access temperature data from a thermal imaging camera, similarly among the applications and services that are accessed on mobile devices, it is the data services

that are without doubt the most demanded services for MANETs, hence we see that it is all about service discovery rather than the node discovery. Our assumption to this approach is that a node at least has one characteristic or knows some neighbour that has one, similarly a node can have multiple characteristics and there can also be multiple sources of the same characteristic inside the network.

In the simple case where a node has one characteristic, then this node spreads out its characteristic information through an advertisement packet to the neighbouring nodes which take this characteristic advertisement packet, inherit the characteristic and then propagate or broadcast it further to their neighbouring nodes. In our approach, this kind of a flow is called a characteristic flow. Thus intermediary nodes from their knowledge can answer the question "Who is the next strong gradient for a specified characteristic X?".

Similar to the analogy of a coloured water stream flow, different flows of a particular characteristic can join together at an intermediate node and then continue to form a new flow. We can also deduce that the gradient is strongest at the source of a characteristic and it starts diminishing as it moves further away. In our approach such kind of a characteristic flow will eventually reach a stable state. Following are the three important points to our design approach:

1. The warm-up/start-up time must be kept minimum where all characteristic flows reach a stable state as this becomes a requirement for MANET protocols always. This is because we do not want the application to be held for a long time where the bandwidth is completely taken up by the packets in the early start-up phase for the characteristic flows. Since ad hoc networks should always deal with frequent changes in topology as mobile nodes tend to wander around, changing their network location and link status on a regular basis, meaning nodes could join or leave the network or could be turned off. Our CBR routing protocol must minimize the time required to converge after such topology changes. A low convergence time is more critical in ad hoc networks because temporary routing loops can result in packets being transmitted in circles, further consuming valuable bandwidth.

2. In the start-up phase each characteristic takes the same steps as discussed above for a flow inside the network as the distribution of characteristics inside the network becomes important for the case where a node fails, however the failure

will have less effect on the service discovery aspect.

3. A particular node upon receiving multiple characteristics will fuse them together before further broadcasting takes place. The analogy can be given of two coloured water streams flowing, one is a red colour stream and the other is a blue colour stream, they eventually meet together at some distance and blend together to form a purple coloured stream and the water flow continues or propagates further for the distribution. We say in this analogy that the density of colour is lowered as the purple coloured stream flows further downstream, thus at some far reached node the colour will become lighter purplish.

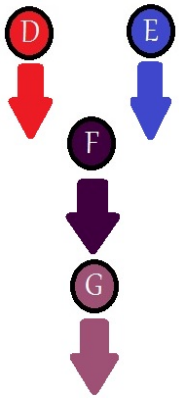

Figure 4-1: Color or Characteristic Propagation [18]

In the figure above, lets elaborate this analogy of a coloured water stream with an example. If a node G wants to send a request to a node for a blue characteristic. It can analyse that node F knows something about blue since its colour is inherited purple from nodes D and E and thus the request will eventually reach node E via node F.

4.2 Terminology

Characteristic

This represents the actual inherent features of the mobile devices which are passed in the CBADV packets inside the ad hoc network.

Characteristic Instance

This refers to the characteristic object, that is realised to denote an active feature of a mobile device.

Local Characteristic Table

This refers to all the characteristic instances that are associated to a particular node and stored inside the local table, which can also contain characteristic instances arriving from other intermediate nodes.

Characteristic Propagation

This refers to the process of characteristic spread inside the mobile ad hoc network.

Weight

This refers to the gradient strength or potential of a particular characteristic instance.

Tracing

This refers to the entries added into the local characteristic table, inorder to assist for the reverse forwarding of Characteristic Reply (CBREP) packets to those nodes that originaly transmitted the Characteristic Request (CBREQ). The entries are added to the path which the CBREQ follows inside the network.

4.3 Characteristic Based Communication

Our routing protocol is different to other routing protocols in the sense that it does not involve routing based on any hop count, distance calculation or sensing the delay in the network, but rather it measures the routing in the size of a weighted metric or in other words computing the capacity of resources when choosing our service providers. We use our characteristic based communication for the purpose of resource lookup or service discovery. Inside this type of characteristic based routing the route is always discovered or setup by a resource potential or like a weighted metric and it fuses along the characteristic flow in order to successfully establish the route to assist for the service discovery. We mention about the node weights as a potential value which represents the capacity of the particular node with a characteristic. Thus each node which has a characteristic also stores the potential value to represent its weight

or capacity. Inside the network the spread of the characteristic as mentioned before simulates the flow of water down in a stream. The direction of the water flow is always from a high potential towards a low potential and also this potential is reduced along the flow similar is the case for our characteristic flow inside the network. Also a number of other flows might merge together at one intersection point to form into a new stream and then continue to propagate further. Therefore we can deduce from it that the higher potential a characteristic contains, then it has the tendency to reach further during the propagation, conversely if a node is closer to a characteristic source then it is likely to sense a high potential. We use potential or capacity as one or another. With the top down encoding of our characteristics which is formed by the underlying features of ad hoc devices or service providers, a node could be represented with the grouping of characteristics to form a set which describes the services like for example internet access could become a characteristic set which represent the internet service; like dial-up, wifi or optical broadband service to name a few, for high capacity services like the metropolitan ethernet or gigabit ethernet could be used for medium to large business services. In our characteristics propagation inside the network, each characteristic shows a different form of weight change depending upon the network environment they run through. This can be explained for instance, a service offering some print service could be accessible in a large area when the form of weight change is minimum for the Quality Of Service (QoS) per unit of area, where as similarly a service of an internet connection could form a higher weight change for the QoS per unit of area, this means its range is less compared to range for the print service, this is much similar to saying that the internet connection running a video stream for a route that is formed over multiple hops does not give the same throughput as the route over a single hop, thus we say that the bandwidth is reduced in a multiple hop route path. In our design we drive the data forwarding by the capacity or a weight gradient, where by eliminating the need for a distance vector or a hop count gradient for the identification of the route instead we use this weight gradient to balance out with the distance from the actual potential of a characteristic.

4.4 Characteristic Flow

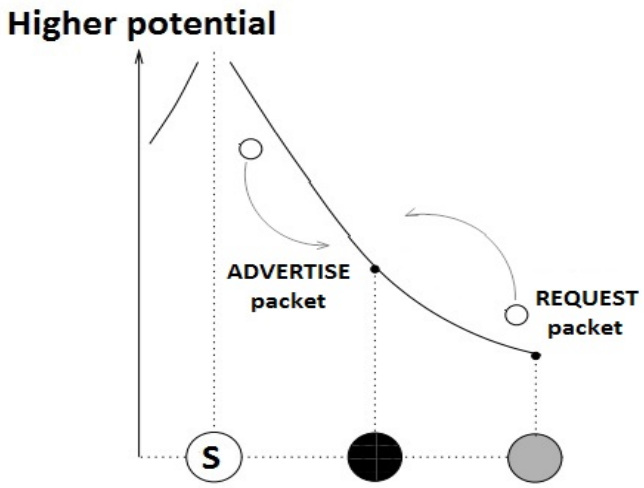

Figure 4-2: Flow

A node inside the network that has a characteristic will at recurring intervals broadcast its characteristic information outwards to its one hop neighbours, the neighbours then inherit these characteristics and then spread them out more further via broadcasts and the process continues, therefore in our characteristic based routing protocol every node has the knowledge about the characteristics of its one hop neighbours. On each single hop during the propagation, the weight gradient gets reduced by applying the weight cost function. The characteristic information always flows in the direction of a low gradient. i.e. it starts at the source which posses higher weight or capacity towards lesser capacities inside the adhoc network. Propagation is controlled by a cost function. Similar to Uisce [6], we define a constant "W_DIFF" which controls the characteristics flow in our ad hoc network.

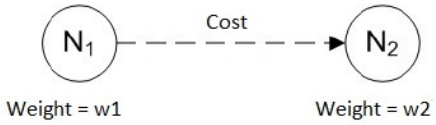

Figure 4-3: Propagation [6]

$$\left\{ \begin{array}{c} w1 - cost \geq w2 + w_d iff \\ cost + w_d iff > 0 \end{array} \right\}, \tag{4.1}$$

The figure 4-3 illustrates the calculation of our weight cost during the propagation of a characteristic from node N1 to N2. This figure illustrates that the sum of weight cost plus the weight diff constant should always be greater than zero, however the weight diff constant value can never be zero but it can be infinitely small, therefore we can extract from this illustration that a node which contains a characteristic with weight W can only accept incoming characteristic packets if the characteristic is of the same type and the receiving node has the weight no lower than W - W_DIFF. Now there could be multiple sources of a characteristic inside the network and these sources tend to meet at some intermediate node, therefore the merged characteristic is calculated based on a weight fusion function. This function in our implementation would generally be a MATH.max method.

$$max\{x_1, x_2\} = \begin{cases} x_1, if\ x_1 > x_2 \\ x_2, otherwise \end{cases} \quad (4.2)$$

Therefore the crossed over characteristic of the same type merging at an intermediate node will actually be worked upon by applying the max technique in order to fuse and merge them over the selected characteristic flow or route. To summarise further our weight cost function is applied to all the characteristics before they are broadcasted in the advertise packets.

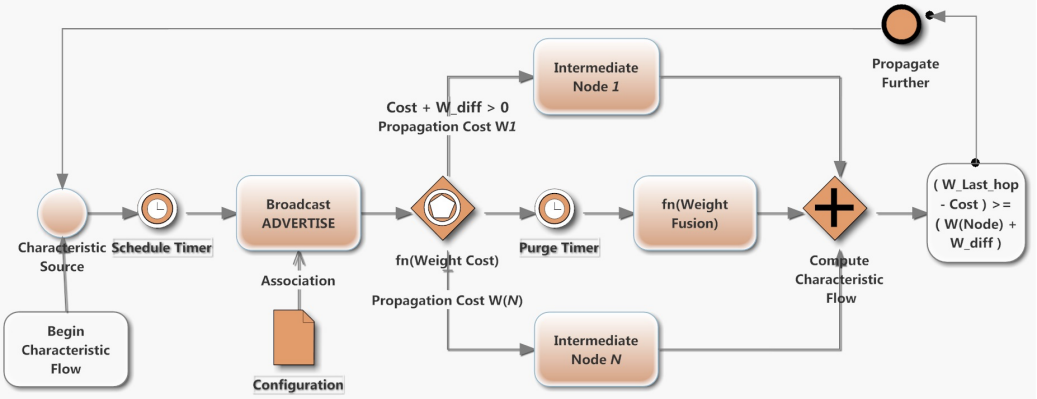

Figure 4-4: Characteristic Flow Diagram

4.4.1 An Example Scenario

We will look at a building fire emergency as an example scenario to describe the workings of our protocol. In the building fire situation, their are fifteen fire fighters who have a wearable computer attached to them all the time and it has the latest

networking capabilities attached to it, like GSM, Bluetooth, GPRS, GPS, UMTS and 802.11 standards technology. Now the computer that each firefighter uses should be able to seamlessly change over with the most suitable technology or service automatically as the communication in such a scenario becomes a mission critical one i.e. the safety of precious human life is utmost. Their can be a situation where a fire fighter detects a dangerous substance and requires to send the still photo or a quick voice message to some remote experts over the internet for labeling or even to the group commander inside the fire engine. One fire fighter could want to stream the incident video in real-time to experts for the rapid decision and response and to control the evolving situation. Another communication could be to stream a thermal camera video to other colleagues that do not have a thermal camera. In all these situations the underlying technology protocols will be processed over CBR. Following table shows the ad hoc network which has reached a stable state for our scenario.

Table 4.1: Firefight ad hoc network

Characteristics	Nodes
Internet Access	FE-1, FE-2
Eutelsat 15Mbps	FE-1
3G-Link 2Mbps	FE-2
GPS-Service	16
Camera	20, 14
Thermal	20
Night-Vision	14

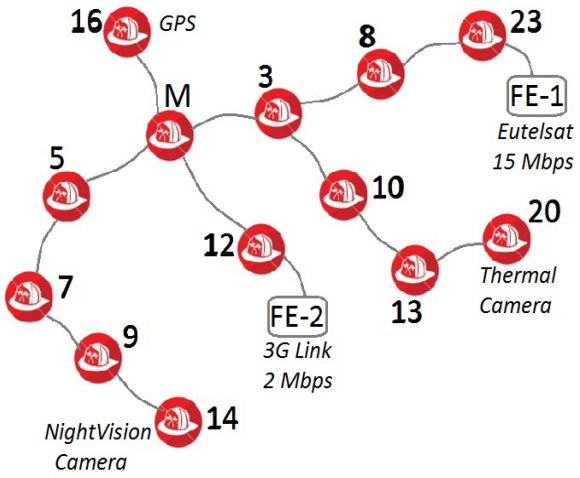

Figure 4-5: Ad hoc stable network

Nodes inside our ad hoc network do the periodic characteristic broadcast and the characteristic flow starts propagating inside the ad hoc network. We go through this characteristic propagation and for node M to perform the following operations:

1. Node M wants to have the thermal camera view as it does not have one.
2. Node M wants to send a short voice message (only few seconds), for the labeling of the dangerous substance found at the incident location.
3. Node M wants to join the video conference with the area commander.

CBR Broadcast attempt in the following chronological order:

- FE-2 Characteristic broadcast; CBR protocol at node FE-2 sends down to the lower layer, CBADV packet (characteristic=3G-Link 2Mbps, seq=1, hop-count=0, node=FE-2)
- Node 20 Characteristic broadcast; CBR protocol at node 20 sends down to the lower layer, CBADV packet (characteristic=Thermal, seq=1, hopcount=0, node=20)
- Node 14 Characteristic broadcast; CBR protocol at node 14 sends down to the lower layer, CBADV packet (characteristic=Night-Vision, seq=1, hopcount=0, node=14)
- FE-1 Characteristic broadcast; CBR protocol at node FE-1 sends down to the lower layer, CBADV packet (characteristic=Eutelsat 15Mbps, seq=1, hop-count=0, node=FE-1)

- Node 16 Characteristic broadcast; CBR protocol at node 16 sends down to the lower layer, CBADV packet (characteristic=GPS-Service, seq=1, hopcount=0, node=16)

- Node 23 recieves CBADV packet (characteristic=Eutelsat 15Mbps, seq=1, hopcount=1, node=FE-1) and it inherits it inside its own characteristic table and then spread it further to its neighbours.

- Node 8 recieves CBADV packet (characteristic=Eutelsat 15Mbps, seq=1, hopcount=2, node=23) and it inherits it inside its own characteristic table and then spread it further to its neighbours.

- Node 12 recieves CBADV packet (characteristic=3G-Link 2Mbps, seq=1, hopcount=1, node=FE-2) and it inherits it inside its own characteristic table and then spread it further to its neighbours.

- Node 13 recieves CBADV packet (characteristic=Thermal, seq=1, hopcount=1, node=20) and it inherits it inside its own characteristic table and then spread it further to its neighbours.

- Node 10 recieves CBADV packet (characteristic=Thermal, seq=1, hopcount=2, node=13) and it inherits it inside its own characteristic table and then spread it further to its neighbours.

- Node 3 recieves CBADV packets (characteristic=Eutelsat 15Mbps, seq=1, hopcount=3, node=8), (characteristic=Thermal, seq=1, hopcount=3, node=10) and it inherits it inside its own characteristic table and then spread it further to its neighbours.

- Node 9 recieves CBADV packet (characteristic=Night-Vision, seq=1, hopcount=1, node=14) and it inherits it inside its own characteristic table and then spread it further to its neighbours.

- Node 7 recieves CBADV packet (characteristic=Night-Vision, seq=1, hopcount=2, node=9) and it inherits it inside its own characteristic table and then spread it further to its neighbours.

- Node 5 recieves CBADV packet (characteristic=Night-Vision, seq=1, hopcount=3, node=7) and it inherits it inside its own characteristic table and then spread it further to its neighbours.

- Node M recieves CBADV packets (characteristic=3G-Link 2Mbps, seq=1, hopcount=2, node=12), (characteristic=Eutelsat 15Mbps, seq=1, hopcount=4, node=3), (characteristic=Thermal, seq=1, hopcount=4, node=3), (characteristic=Night-Vision, seq=1, hopcount=4, node=5), (characteristic=GPS-Service, seq=1, hopcount=1, node=16) and it inherits it inside its own characteristic table and then waits for any characteristic update.

The shape of node M characteristic table is given below:

Table 4.2: Characteristics table of Node M

Characteristics	Distance In Hops	Next-Hop Node
Eutelsat 15Mbps	4	3
3G-Link 2Mbps	2	12
Thermal	4	3
Night-Vision	4	5
GPS Service	1	16

Now, if we assume that the network becomes stable and reaches a static state i.e. weights no more fluctuate. Then if node M wants to use his video lan client (VLC) software to hook up to the RTSP stream provided by node 20 thermal camera. The application layer of node M can locally resolve the thermal camera node i.e. rtsp://thermalcamera:554 which is present inside the fire fight ad hoc network, CBR will then support the forwarding of the requests to the resource using characteristic instance 'Thermal'. The RTSP OPTION and PLAY request is packaged into the CBR packet and directed towards the strong gradient source via nodes 3,10 and 13. Upon arrival at the source, the CBR protocol layer unpacks the RTSP OPTION and PLAY requests and passes it the upper application layer at node 20, which sends back the response to the requestor node M, following the characteristic flow downstream via the CBR trace knowledge i.e 13,10,3. This request and response forms a service context and handshake is established i.e. two way communication link is made and the streaming of the video can begin.

Similarly, if node M wants to send a few seconds of voip data, node M network layer can use the voip facility via a low bandwidth link. As the size of the message consumes low bandwidth, our CBR protocol will choose characteristic instance '3G-Link 2Mbps' and therefore the voice over ip protocol message will be packaged in the CBR packets and forwarded to FE-2 via node 12, which then gets unpacked in the upperlayers of FE-2 and sent out to the voip endpoint.

However the setup of video conferencing with the area commander requires high-bandwidth, therefore CBR in this case will communicate over the characteristic instance 'Eutelsat 15Mbps' which is a highbandwidth link and the CBR packets will relay back and forth the traffic data i.e. VOICE(RTP) fragments over the highbandwidth link using the CBR protocol.

The CBR protocol builds up the characteristic heuristics for message types like VIDEO, VOIP, IM, EMAIL, THERMAL e.t.c and will eventually know which characteristic instance to choose or prefer for data delivery based on their weights.

4.5 Characteristic Hierarchical Coding

We use a characteristic hierarchical coding model to form a coding scheme with quality hierarchy, in this model the lowest layers inside the hierarchy have less detailed information for intelligibility but the succeeding number of layers contribute to the increasing quality inside the ad hoc formation.

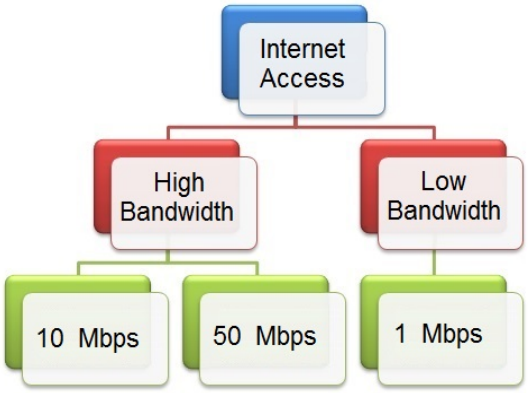

Figure 4-6: Characteristics Hierarchical Scheme

We see that in networks, packets pass on data from one node to the other and similarly these packets can be marked according to their importance for intelligibility towards the characteristic inside the network, this knowledge can be useful in determining which packets could be dropped or delayed or given priority. This hierarchical coding model becomes ideal to deal with transmission over a number of links with different bandwidths. In a non-hierarchical coding scheme either the complete traffic passed through a lowest bandwidth link degrades the quality of the service or to cause the low bandwidth link to face congestion where the requestors become affected and loose intelligibility for the service. Thus with the choice of a good hierarchical coding

scheme, request packets could be filtered to preserve the intelligibility of characteristic instances inside the network.

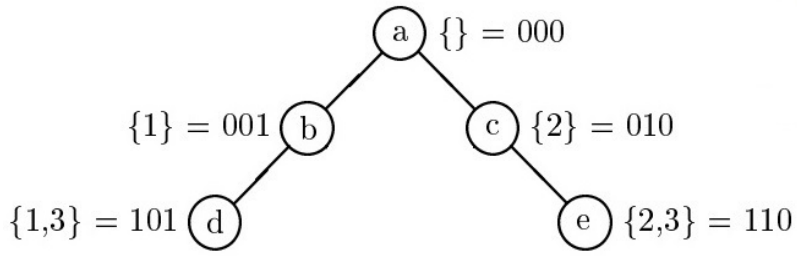

Figure 4-7: Characteristic Heirarchy - Abstraction

$$\left| \begin{array}{cc} a : InternetAccess & b : LowBandwidth \quad c : HighBandwidth \\ d : 1Mbps & e : 10Mbps \end{array} \right| \tag{4.3}$$

We see above in the figure which illustrates how a top down hierarchical encoding for a node is computed and which is actually the union of the characteristics for all its ancestors and for itself. This parent child relationship forms a tree structure where we can see that the node "e" encoding 110 is formed by the union of node "c" and node "a" characteristics. Such type of a hierarchical encoding scheme simplifies operations on the characteristics. We define such operations as a union set function which brings along a number of characteristic flows towards an intermediate node and then elevate this characteristic to a higher level at some remote node along the propagation path. This means that the characteristics reaches an abstraction after the union or fusing of a number of characteristic flows at some intermediate point during the flow. Therefore we can say that the characteristics abstraction becomes less detailed along the propagation path.

4.6 Data Forwarding

As we have the CBADV packets inside our routing protocol for the spread of the characteristic. We also have CBREQ, CBREP packets to assist sending of data to the characteristic source and receive response. The flow of these data packets is followed in reverse to our weight gradient definition as mention in the last section. We define this logic in our implementation where the data packets will always be propagated from the nodes with lower weight to the nodes that contain higher weight i.e. upstream. Data forwarding works by broadcasting to close proximity neighbours

i.e. one hop neighbours. So when sending data to the source of a characteristic, the sender has no knowledge about the location or proximity, rather it relies on the one hop neighbours whom he thinks might have the information for the actual source of the characteristic, thus the one hop intermediate neighbour upon receiving this data packet continues to follow the same process that the last sender performed and so this process continues until our data packet reaches the specified characteristic source.

Data forwarding involves the data packets that carry the information to the source, this completes one end communication in ad hoc networking, however sometimes there becomes a need for different applications to have two end communications i.e. the source also wants to send a reply message back to the requestor, when it receives a request. This puts an additional constraint to maintain the communication at both the sender and receiver nodes. Inside our characteristic based routing we maintain the two end communication by introducing Characteristic Trace (CBTRC) packets.

This two end communication forms a different context to the one end communication path, where the context is established whenever the first data packet arrives at the characteristic source node.

4.7 Route Tracing

We define CBTRC entry for reverse forwarding of the replies back to the requestor. When the CBREQ is sent from the requestor node to a characteristic source, the protocol during its upstream propagation through the intermediary nodes, creates reverse path trace entries in the local characteristics table of the intermediate nodes. The entries are associated with an expiry timer, when ever the node is traversed by the CBREQ, the trace entry expiry timer is reset to the current time plus the reverse path expiry timeout. This way the trace characteristic entry remains fresh. Thus when the CBREQ reaches the source, it can send back the CBREP back via the same trace route that the request came from.

4.8 Characteristic Updating

Inside our characteristic based routing protocol, there exist a node which contain at least one characteristic which is considered active with high gradient value and associated to the particular node, from this node the characteristic spreads to the other

nodes, which further stores the received characteristic information inside their local characteristics table. Our protocol works by configuring and starting a few internal timers, it has an internal Characteristic Advertise Timer (CBADVTIMER) message which helps the protocol working in the network layer to initiate periodically the CBADV packets from those nodes that want to broadcast from their active internal characteristics set and are ready. This CBADV packet is then broadcasted to all the neighbours. This spread is similar to a HELLO packet inside the OLSR proactive approach, it moves forward and gets distributed like in most other gossiping protocols, which are commonly defined by probabilistic flooding, Upon receiving the first advertise packet, a node will forward this packet with a fixed probability P to the neighbouring nodes that lie within the communication range via the known broadcast primitive of the MAC layer. The forwarding mechanism also checks for the CBADV packets propagated weight cost and makes sure that it is greater than the local stored weight value for the broadcast characteristic. Also inside the forwarding mechanism the nodes delay the forwarding of the CBADV packet for a random time which is between 0 and fDelay. We illustrate this in our algorithm as given below.

```
Var: c, backoff
Map: characteristic_table
# On receive characteristic advertise message AM
if AM.CHARC ∉ characteristic_table then
    if AM.FORWARD_WEIGHT_COST ≫ WEIGHT then
        # Received characteristic AM for first time
        add {AM.CHARC} to characteristic_table
        print AM
        if randomDelay(1.0) ≤ c then
            wait(randomDelay(backoff))
            broadcast AM to all one hop neighbours
        end
    end
    else
        discard AM
    end
```

Algorithm 1: ADVERTISE(c) - Advertise Packet Algorithm

4.9 Characteristic Table

Our node's characteristics table is made up of the following fields as shown below:

Table 4.3: Characteristics Table Fields

Name	Type
Characteristic	string
NextWeightAddress	string
NextWeight	long
Weight	long
SequenceNumber	long
HopCount	integer
ReachStable	boolean

This table stores all information about the propagating characteristic flows ocurring inside the ad hoc network. The lookup key is the characteristic field, which is used to search and update entries inside the table when CBADV arrives. All the characteristic table entries have an associated timer which is kept for the characteristic staleness check to ensure that the node proximity is correct and it remains fresh. Hence the characteristic stable field is set, in the case where a node disappears as detected by the expiry timer event, then its value is reset as it senses that for that particular characteristic their are no more incoming flows.

4.10 Weight Cost Calculation

We apply our simple weight cost function that simulates the weight lost in the direction to the characteristic flow. This is deduced by applying a gradient function to calculate the minimum of the function for multiple variables.

$$cost(w) = w + \frac{(w^1 + const)}{1!} + \frac{(w^2 + const)}{2!} + \frac{(w^3 + const)}{3!}... \tag{4.4}$$

$$sinw = \sum_{n=0}^{\infty} \frac{1}{(2n+1)!}(w^{2n+1} + const) \tag{4.5}$$

The cost term belongs to the Maclaurin series expansion where all the terms are nonnegative integer powers of the weight cost variable. W_DIFF is computed using the sine function as below:

$$\begin{vmatrix} const = 0.5 \\ wDiff \approx sin(w * \pi/\alpha) \\ \alpha = 50000 \end{vmatrix} \tag{4.6}$$

We tune the parameters to adjust the gradient function, these are the values that we use for our evaluation.

Chapter 5

Implementation

This chapter discusses our implementation part, using the OMNET++ [23] network simulation library and the usage of opensource MiXiM, INETMANET frameworks. We concentrate on the network layer part which is the focus area of our characteristic based routing protocol. We show the setting-up of our network model to collect our simulation results.

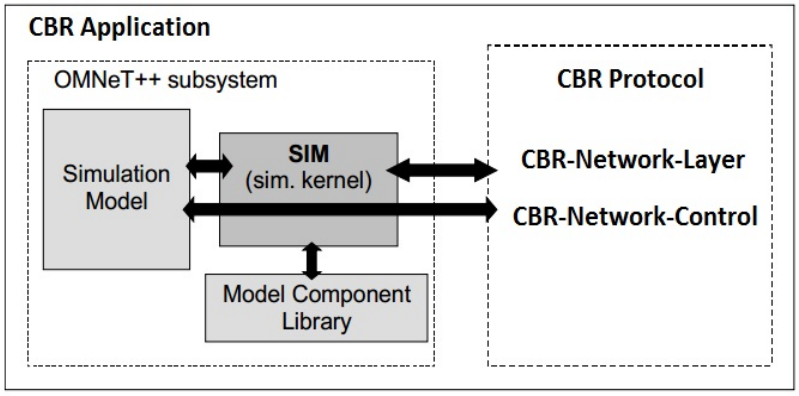

Figure 5-1: Simulation model for CBR

5.1 Overview

OMNET++ uses the concept of modules that work in layers, and they can communicate via sending and receiving messages. For our CBR module, we do the mapping of our implementation system to these layers of communicating modules. The components that we implement, and the model structure, we develop is managed through a network description (NED) file. Our CBR modules are developed in C++. We test our implementation by supplying the simulation engine with our test configuration

ini file. After successfully building the program, we link our code to the omnet++ simulation kernel and execute the simulation. We have defined a number of scalar and vector recorders inside our program to capture our test metrics for latency, throughput, overhead and success rate.

5.1.1 MiXiM framework

MiXiM framework provides the modeling of the physical, network and application layer. We can set the physical parameters like sensitivity and radio channel frequencies for our simulation. Following figure illustrates the MiXiM model where the devices can be configured to provide an interface to access the medium.

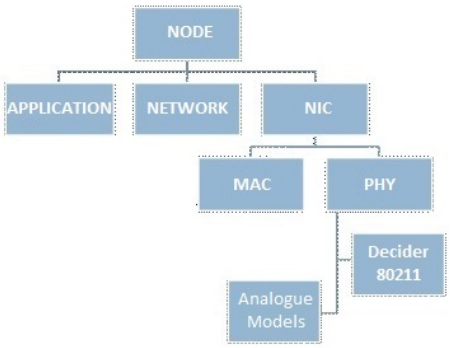

Figure 5-2: MiXiM - Framework model

The important aspect of setting up the network simulation for our protocol is the medium access control and the network layer. This is the area where our protocol efficiency will show effect. MAC layer provides the channel access function that will support a number of nodes to be able to communicate inside the network. This configuration setting will allow the nodes to remain connected by sharing the same physical medium. As their are wide variety of protocols for wireless communication, we carried out simulation with three MAC protocol implementations

1. CSMA/CA
2. LMAC (Lightweight Medium Access Protocol)
3. IEEE80211

In the end, we carried out our final evaluation based on the standard IEEE80211 MAC protocol with the following decider settings.

Table 5.1: Decider80211

Parameter Name	Parameter Value
Threshold	0.12589254117942

Our network layer component is the nitty gritty of our protocol and it is this layer that is responsible for propagating requests to the characteristics source nodes. We view that the network control, error control are part of this network layer implementation.

5.2 Data Structures

Following are the typedef's for storing the characteristic weights and next hop addresses, it is this data structure that forms our routing table that we look upon in forwarding the characteristic requests to strong sources.

```
struct localCharacteristicEntry {
        std::string nextWeightAddress;
        double nextWeight;
        double weight;
        int hopCount;
        unsigned long seqNumber;
};
typedef std::map<std::string,
struct localCharacteristicEntry>
mapCharacteristicEntries;

typedef std::map<std::string,
mapCharacteristicEntries_>
localCharacteristicsTable;
```

We define a CBADVTIMER message that manages the sending of CBADV messages from nodes that inherit the characteristic instance. Each node will then associate an entry timer to keep the local characteristics table entries fresh.

```
/* Timer for individual characteristic entries inside
 * the local table. */
class CBR_CharacteristicLinkTimer : public CBR_Timer
```

```
{
  public:
    CBR_CharacteristicLinkTimer(CBRSingleton* controller)
        : CBR_Timer(controller) {}

    CBR_CharacteristicLinkTimer(CBRSingleton* controller,
        mapCharacteristicEntries* entry)
        : CBR_Timer(controller) { singleEntry = entry; }

    CBR_CharacteristicLinkTimer():CBR_Timer() {}
    void expire();
};
```

5.3 Packets

Omnet++ provides a flexible way of modelling protocol packets for the necessary communication among the nodes. We use this provided flexibility of message definition to create our CBR packets. The message c++ files are automatically generated via reflection inside omnet++.

```
class LAddress::L3Type extends void;
packet CbrPkt
{
        LAddress::L3Type destAddr;
        LAddress::L3Type srcAddr;
        int              hopCount;
        unsigned long    seqNum;
        string characteristic;
        double weightOnPropagate;
        double weight;
        string data;
}
packet CbrReqPkt extends CbrPkt
{
        LAddress::L3Type reqSrcAddr;
        long transactionId;
```

```
        simtime_t messageTime;
}
packet CbrRepPkt extends CbrPkt
{
        LAddress::L3Type reqSrcAddr;
        long transactionId;
        string payload;
        simtime_t messageTime;
}
packet CbrTracePkt extends CbrPkt
{
        LAddress::L3Type reqSrcAddr;
        long transactionId;
        simtime_t origTime;
        simtime_t messageTime;
}
```

5.3.1 Request and Reply message fields

In our implementation, the communication context is one way, so we only require to persist the address from the origination point only. i.e. when a CBREQ is generated at the application layer from any node, this originating address is only relevant for our one way communication. When the packet arrives at the characteristic source, then it knows where to send back the reply, i.e. to which origination point. This is set inside the 'ReqSrcAddr' message field. The hop count resembles the life span of the packet and it defaults to 4 hops. After this, the packet acquires an ascended hierarchical code from the characteristic set and starts from fresh. Sequence number identifies a unique message for a certain advertise characteristic originating from a node, when it arrives at the intermediary node, this sequence number is compared to assist for handling and keeping uptodate with the topology changes.

5.3.2 Extra message fields

In our message definition, we add extra fields for our scalar and vector recording purpose. These include the transactionId, origTime, messageTime and data field. TransactionId is used to collate the request and response packets arriving from the

source node, we match our requests with reply based on this identifier, which initialises it self from a time value and we can do a less than check to confirm that the transactionId is increasing with each node request generated at the application layer.

$$(Tid_n)_{n=1}^{\infty} \tag{5.1}$$

We use the orig and message time field for the latency metric, where as the data field is simply an optional place holder for sending back a protocol message state back to the requestor inside the reply packet coming from the characteristic source node.

Statistics

Table 5.2: Network Layer Recordings

Type	Name
Scalar	Netw-Control-Overhead
Scalar	Control-Data-Bits

Table 5.3: Application Layer Recordings

Type	Name
Scalar	Total-Requests-Received
Scalar	Total-Requests-Sent
Scalar	Control-Overhead
Scalar	Aggregate-Latency
Scalar	First-Packet-Sent-Time
Scalar	Last-Packet-Receive-Time
Scalar	Total-Data-Bits
StdDev	Test-Stat
Vector	Latencies-Raw

5.4 Single Hop Neighbor Management

The network layer also keeps track of the level 2 addresses of the one hop neighbours, inorder to propagate the characteristic advertisement and for the request and reply packets. For this it has a second routing table to manage the node level 2 addresses. For this we define a typedef as below.

```
typedef std::map<LAddress::L3Type, LAddress::L2Type> RoutingTable;
```

5.5 CBR Application Layer

This layer is responsible for sending out CBREQ packets to characteristic sources. This layer first waits for the service application above to send a service message destined to a particular characteristic source for initiating the path for the service discovery. When the sending node's application layer receives a CBREP from the characteristic source, it passes the reply message to the upper service layer which creates a bind for the communication endpoint and establishes a service context.

5.6 CBR Network Layer

This layer starts with the broadcast of CBADV packets inside the ad hoc network and builds the characteristic spread. It is the source nodes which announce a strong gradient weight. Since this layer deals with our network behaviour, it is this layer which adds to the control overhead.

5.7 CBR as a State Machine

Inside characteristic based routing protocol the application layer and the network layer component modules gets initialised in the beginning by reading the configuration parameters supplied in the omnet.ini file.

Once these modules are set and initialised, the protocol moves from the INIT state to the IDLE state, then it remains in an IDLE until the schedule timer for the ADVERTISE characteristic broadcast expires, and the protocol starts the warm-up phase of the ad hoc network, where the nodes are trying to refresh their characteristic instances list from the receiving separate broadcast messages. When the node is ready to send an ADVERTISE broadcast message, the state is shifted to ADVERTISE state. Similarly when a neighbouring node receives this broadcast, it moves to RECEIVE-MESSAGE state, the RECEIVE-MESSAGE state is set when ever a message of type CBADV, CBREQ, CBREP is received from the lower layer. The node decides if the request is destined to a next hop address, in that case the message is forwarded or

relayed to the next hop address, the node then shifts to RELAY-MESSAGE state. If the CBREQ arrives at the characteristic source node then the node delivers this message to the application layer and the CBR protocol prepares a CBREP packet destined for the requestor and delivers it to the lower layer. The node arrives back to the IDLE state.

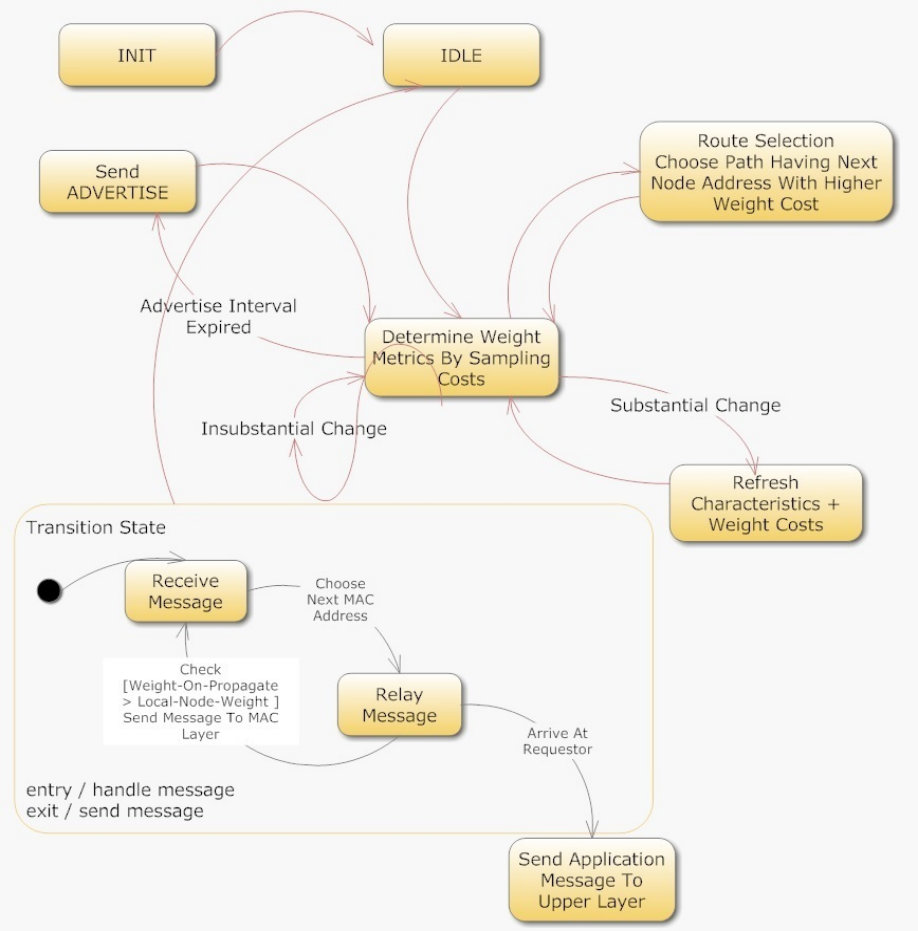

Figure 5-3: State diagram

An intermediate node will buildup the weight metric and always choose the higher gradient weight for forwarding the CBREQ to the next hop.

Chapter 6

Evaluation

In this chapter, we evaluate the performance metrics of our characteristics based routing protocol and present a comparative study of the results we have gathered in our test analysis and inspect these with the other routing protocols like AODV and OLSR. The tests are performed using the MiXiM modeling framework which is perfect for testing mobile ad-hoc networks as it concentrates on the lower layers of the protocol stack, we use the OMNET++ simulation engine for running our tests. We use the INETMANET framework for testing the performance of the AODV and OLSR routing protocols. In our evaluation, we managed to look into the latency, throughput, control overhead and the success rate metrics. We analyze our simulation results by calculating the mean for each performance metric.

6.1 Performance Measurements

1. We measure our latency and define it as the time taken by a CBRRequest packet from the network layer of the requestor node until the packet arrives at network layer of the characteristic source node. We then add this metric to our recording vector for calculating the mean latency for the test duration.

2. We measure the control overhead by adding up the byte size of all the CBADV packets received by each node in the entire duration of our simulation test run.

$$adv_overhead = \sum_{M=1}^{n} adv_overheadM \tag{6.1}$$

Where

$$adv_overheadM \tag{6.2}$$

is the number of advertise packets generated by node M.

3. We gather our throughput metric by calculating the amount of data that is transferred and then divide it by the data transfer time which is actually measured by recording the time interval of the first request packet that is sent from any requestor node and the last request packet which is received at the characteristic source node during the entire simulation duration.

4. We measure the success rate by dividing the count of successful delivery of request packets for a characteristic source by the total number of request attempt by all the nodes during the test run.

6.2 Generic Scenario

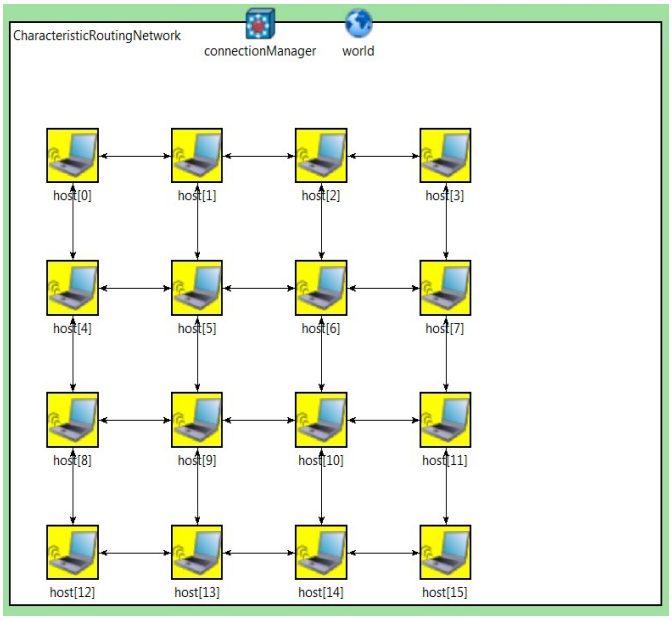

Figure 6-1: Node mesh 16

Our test network is composed of N nodes where N = 10 is set to simulate a low-density network and N = 100 is set to simulate a high-density network. Nodes are placed in an 800 by 800 square meter field, the nodes start off in a square box i.e. they are initialised to correct positions in the square area and then they start moving with constant speed (1mps, 5mps, 10mps, 15mps, 20mps). All the simulation test run were completed under 120 seconds in Express mode. We use the constant speed mobility type as one of the topology property of our network because it impacts the evolution

of our topology. If we increase, the speed parameter, it will give us a more dynamic network topology. The node wait time is a default to 5 seconds, higher this number, more likely the network topology shifts to static. We choose our speed depending on the range of our radio module during our simulation. We show our result statistics in average metrics.

Table 6.1: Settings

Parameter	Value
CBADV Interval	Random [1-9] sec
CBREQ Frequency	10/sec
Theta	0.5
Mobility-Type	ConstSpeedMobility,MassMobility
Nodes	10,16,20,25,30,40,50,60,70,80,90,100

Table 6.2: Simulation parameters for our physical layer

Module	Parameter	Value
connectionManager	alpha	4
connectionManager	sat	-120dBm
connectionManager	pMax	110.11mW
phy	maxTXPower	110.11mW
phy	sensitivity	-119.5dBm
mac	bitrate	2E+6bps
radio	berTableFile	per_table_80211g_Trivellato.dat
wlan	typename	Ieee80211Nic

To test the feasibility and correctness of our characteristics based routing protocol, we tested it in two different mobility settings, "Mass_Mobility and Constant_Speed_Mobility". We also in the start used "Stationary_Mobility" where the nodes are standalone and do not move. Each node was configured to use the IEEE802.11 MAC and the radius of the range for our radio module were set to 100 metres inside our physical model. We setup a test environment using the INETMANET framework for analyzing the latency and successrate for the provided implementation of the AODV and OLSR routing protocols. We updated the UDPBasicApp provided in the framework to collect the latency and success rate metric, exactly like in our test setup inside

MiXiM where a number of nodes were configured to be characteristic source nodes and other were setup as requestor nodes sending out request packets to a number of characteristics. In all our simulation test runs, the traffic model used was UDP packet flows, with 100 byte payloads and the traffic sources were configured to send out 10 requests per second. We used the available INETMANET framework for evaluating the latency and the successrate using AODV and OLSR protocol.

6.3 Latency

In our simulation we found the latency to be acceptable. As the design of our protocol is proactive, we see from the latency graphs that AODV adds extra latency because it works in reactive mode, meaning it adds extra delay in finding new routes.

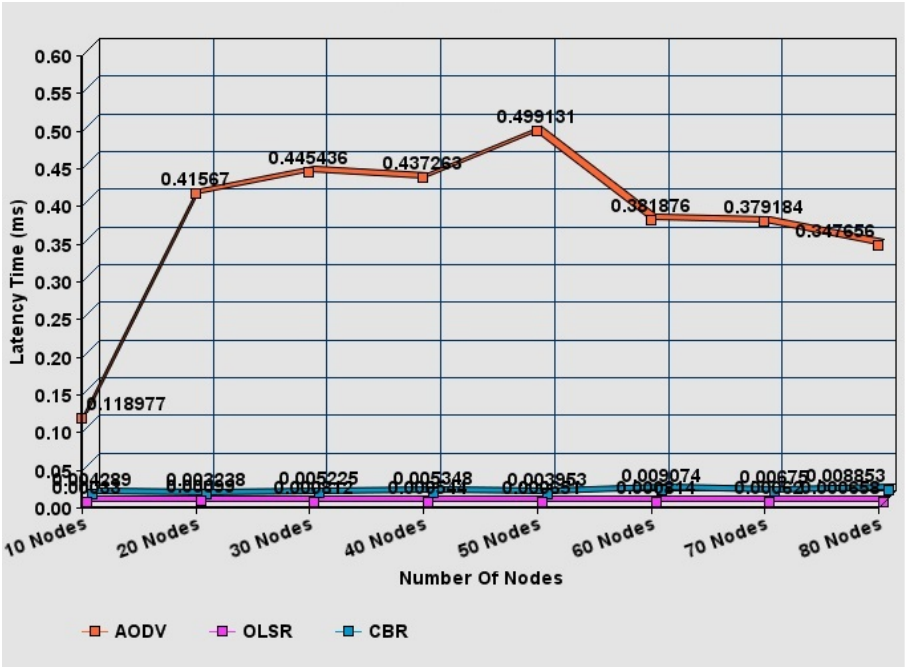

Figure 6-2: Mobility 1mps

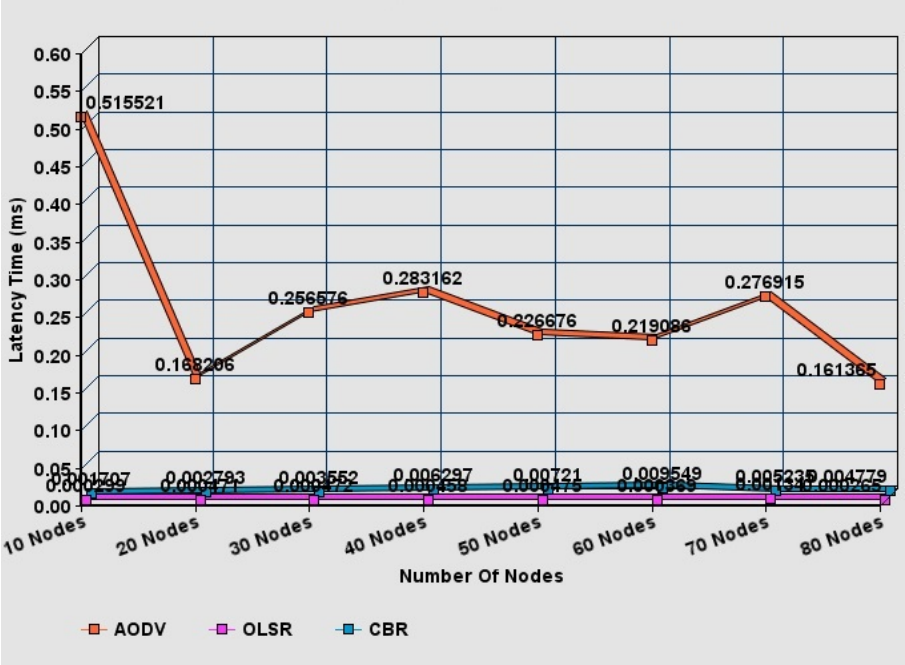

Figure 6-3: Mobility 10mps

We analyse, that the latency drops, as the mobility speed increases for AODV. We found OLSR faired better in latency.

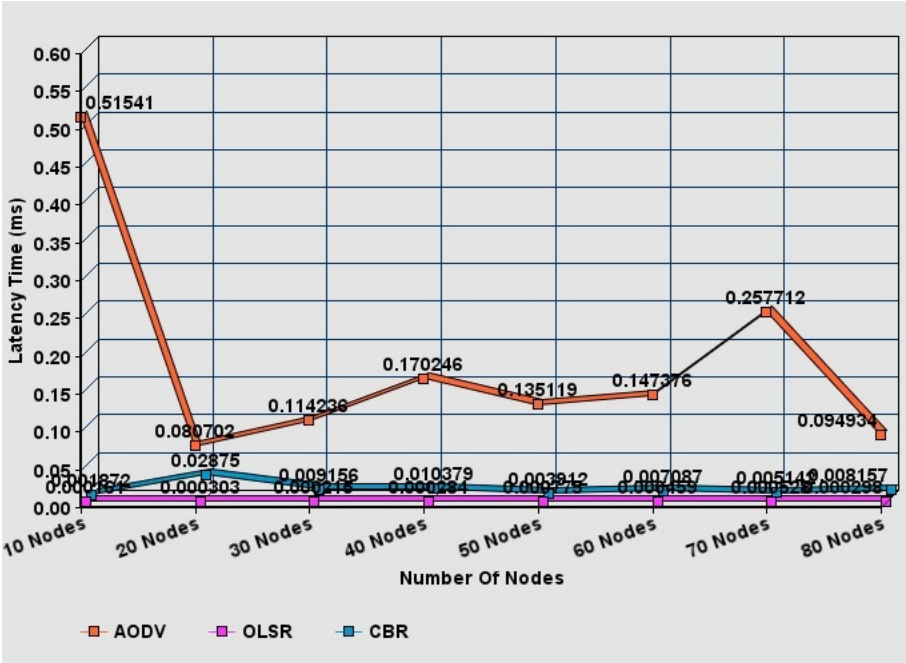

Figure 6-4: Mobility 20mps

6.4 Success Rate

The success rate metric tells us about the efficiency of our routing protocol. We measure it by accumulating the count of the CBREQ received at the characteristic source originating from the application layer of different requestor nodes. From our test, we managed to see around 65 percent average success rate which is better to the OLSR metric of 44 percent, but behind the AODV success rate. Issue of lower success rate to AODV is found to be because of packets being dropped in the simulation as no alternate route could be found. This was investigated to be because our network did not reach a stable state where the weight values kept fluctuating with the increase in mobility.

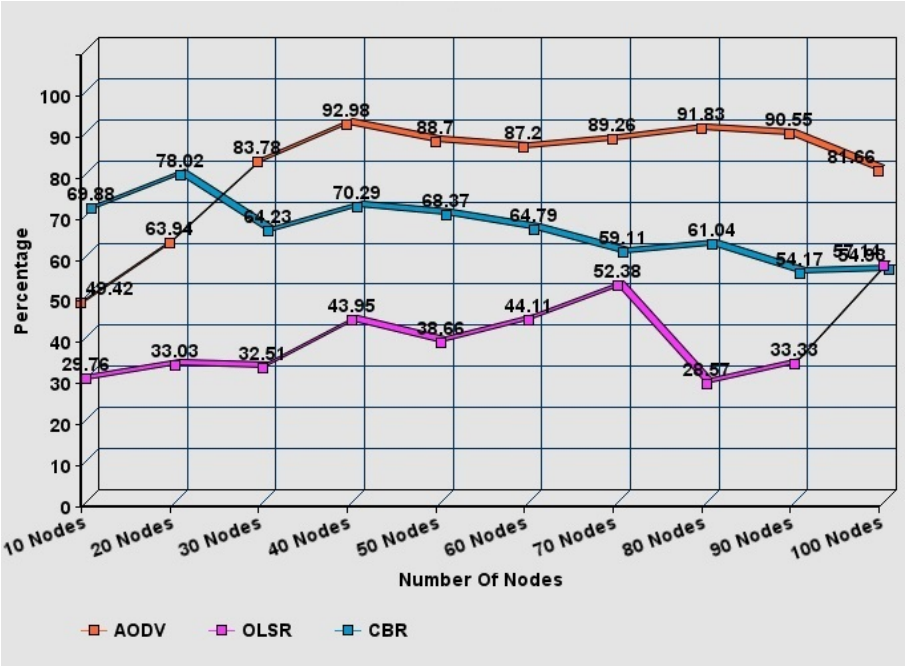

Figure 6-5: Success Rate

6.5 Throughput

This metric will effect the performance of applications, therefore, we need to look for a high throughput, as a high metric value becomes a quality trait for communication networks and routing protocols. We setup our simulation test to analyze how the varying mobility settings and the characteristic route length will effect the throughput value. The characteristic route path is at maximum inside high density i.e. when N = 100. Our request packet was a constant block size meaning the size of the packets

data and header part remained constant and did not change during the simulation run. The test was carried out with 5 iterations in express mode using the omnet++ simulation engine.

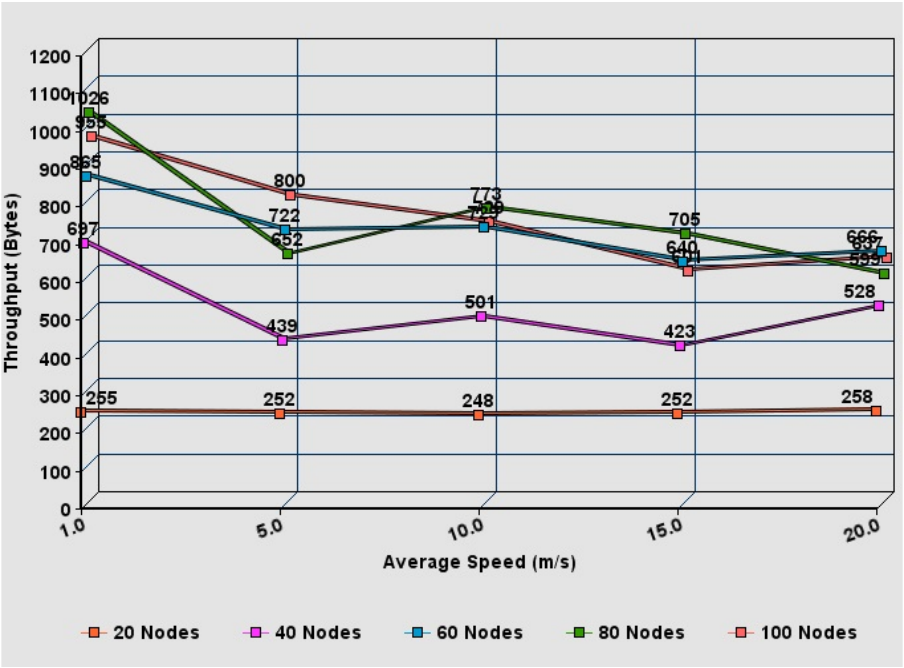

Figure 6-6: Throughput - Requests traffic (10/sec) for two characteristics

The above graph shows that the throughput in our characteristics based routing protocol increases as the node density increases. We can see that the throughput value tends to converge as we shift to the higher node density areas.

6.6 Overhead

We studied different grid settings for analyzing the advertise routing overhead, we worked out the test plan for low density N = 20 and high density network N = 100. We did not manage to achieve a horizontal line. However, in our test we did achieve near to perfect connectivity for low density N = 20 and N = 40, the higher the density the routing overhead gets increased inside high mobility. Following figure illustrates our findings.

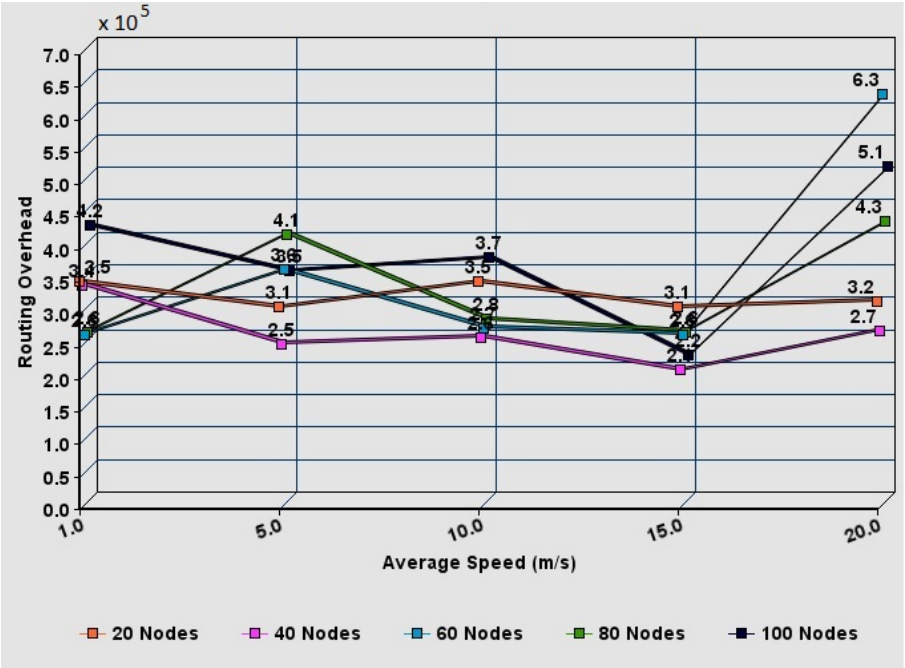

Figure 6-7: Overhead advertise in bit/sec

Chapter 7

Conclusions

In this master proof, we have studied the issue of delivering and routing data traffic based on node features or characteristics rather than the assigned IP addresses. We have given a complete view of why this approach is feasible for MANETs where their becomes a huge demand in enterprise for setting up urgent communication network in situations of war conflict and emergency disaster relief operations due to some major incident. We have shown how our routing protocol seamlessly and efficiently provides service discovery and assistance for communication to nodes that announce specific features or characteristics. We have tested in simulation the scalability of the protocol in the presence of different mobility settings and route failures and we have achieved acceptable latency level for the data delivery in MANETs. We have in our design shown that nodes in close proximity are chosen than the distant nodes unless we sense routes with high potential over the distant path. Therefore, we can conclude that our protocol integrates the characteristics routing correctly with service discovery.

7.1 Future Work

Following are the areas that we are aware to improve further in the development of this protocol.

1. Research into the design of hybrid heuristics using simulation technique in a highly heterogeneous network space with unique services.

2. A truth based mechanism for ensuring that the nodes are not declaring a false weight cost during the broadcast, in order to protect from some malicious attacker.

3. Reducing the algorithmic complexity to self adjust when a low bandwidth is sensed and when using battery power reserve.

4. Bundling of multiple characteristic instances.

5. Adding a session control mechanism when the nodes battery power has gone below a threshold level or has wandered off too far. Inorder to keep track of the communication, when the node reappears.

Appendix A

Source Code

This appendix, provides with the source code of our proof of concept characteristics based routing protocol which is implemented using OMNET++ v4.3 [19] network simulator and using the MiXiM v2.3 [20] framework.

A.1 Network Layer Module

A.1.1 CbrNetwLayer.CC

```
#include "CbrNetwLayer.h"
Define_Module(CbrNetwLayer);
/*--------methods----------*/
CbrNetwLayer::CbrNetwLayer() :
        BaseModule(), boredTime(), maxTtl(0), na(), dataIn(-1), dataOut(-1),
         startCharacteristicAdvertisement(NULL), runningSeqNumber(0),
         cntrlOverhead(0), ohNumberOfControlBits(0), routingTable() {
    cbrSingleton = cbrSingleton->getInstance();
}
CbrNetwLayer::~CbrNetwLayer() {
    cancelAndDelete(startCharacteristicAdvertisement);
    delete cbrSingleton;
}
void CbrNetwLayer::scheduleCharacteristicAdvertisement() {
    if (startCharacteristicAdvertisement->isScheduled()) {
        cancelEvent(startCharacteristicAdvertisement);
    }
    scheduleAt(simTime() + (rand() % (9 - 1 + 1) + 1),
            startCharacteristicAdvertisement);
}
void CbrNetwLayer::broadcastAdvertisement() {
    printLog("Broadcasting advertisement packet.");
    cbrpkt_ptr_t helloAdvertisement = new cbrpkt_t("ADVERTISE", ADVERTISE);
    helloAdvertisement->setBitLength(10 * 8);
    helloAdvertisement->setDestAddr(LAddress::L3BROADCAST);
    helloAdvertisement->setSrcAddr(na);
    helloAdvertisement->setSeqNum(runningSeqNumber++);
    helloAdvertisement->setHopCount(1);
    NetwToMacControlInfo::setControlInfo(helloAdvertisement,
            LAddress::L2BROADCAST);
    const_cast<cModule*>(getNode())->bubble(
            "Advertise Message #" + runningSeqNumber);
    sendDown(helloAdvertisement);
}
void CbrNetwLayer::sendDown(cPacket* pkt) {
    send(pkt, dataOut);
}
/* Method takes care of forwarding the ADVERTISE~ment
 * messages to single-hop nodes during a
 * characteristics broadcast */
void CbrNetwLayer::handleAdvertise(cMessage* msg, cbrpkt_ptr_t pkt,
        double rssi) {
    /* Who advertise this message */
    const LAddress::L3Type& srcNodeAddress = pkt->getSrcAddr();
```

```
/* We already know ourself, i.e. it matches our NA (Node Address) */
if (srcNodeAddress == na) {
    std::stringstream printSrcNodeAddress(std::stringstream::out);
    printSrcNodeAddress << "PrintLog::HandleAdvertise - SrcNodeAddress: "
            << srcNodeAddress << " NodeAddress: " << na;
    printLog(printSrcNodeAddress.str());
    delete pkt;
    return;
}
/* Have we got information about him already? Start building the routing table. */
if (routingTable.count(srcNodeAddress) == 0) {
    /* If not then add him to our routing table with the mac address of the previous hop */
    MacToNetwControlInfo* cInfo =
            static_cast<MacToNetwControlInfo*>(pkt->getControlInfo());
    const LAddress::L2Type& prevHop = cInfo->getLastHopMac();
    routingTable[srcNodeAddress] = prevHop;
    cbrSingleton->getInstance()->insertIntoAddressTable(srcNodeAddress,
            prevHop);
    /*std::stringstream osBuff(std::stringstream::out);
     osBuff << "Got advertise packet from host[" << (srcNodeAddress - 1) << "]-"
     << srcNodeAddress << "  prevHop: " << routingTable[srcNodeAddress];
     const_cast<cModule*>(getNode())->bubble(osBuff.str().c_str());
     printLog(osBuff.str());*/
    /* Now also schedule characteristic advertisement */
    scheduleCharacteristicAdvertisement();
    ohNumberOfControlBits += pkt->getBitLength();
    delete pkt;
    return;
} else {
    std::stringstream thisNodeAddress;
    thisNodeAddress << na;
    std::stringstream nextNodeAddress;
    nextNodeAddress << pkt->getSrcAddr();
    std::string pktCharacteristic = pkt->getCharacteristic();
    /* Make sure the source node is eliminated from the characteristic
     propagation stream flow */
    bool isConfiguredAsSourceNode =
            cbrSingleton->getInstance()->isConfiguredAsSourceNode(
                    thisNodeAddress.str(), pktCharacteristic);
    /* Make sure the incomming packet has the propagation weight larger than the
    one stored for the node inside the local table */
    double localWeightStored =
            cbrSingleton->getInstance()->getLocalNodeStoredCharacteristicWeight(
                    thisNodeAddress.str(), pktCharacteristic);
    int localTotalRoutes =
            cbrSingleton->getInstance()->getTotalAddressRoutes();
    /* Looks like a characteristic advertisement packet has arrived from some
     neighbour, deal with it and propagate it further */
    if (!isConfiguredAsSourceNode /*&& localTotalRoutes != (numHosts-1)*/
    && pkt->getWeightOnPropagate() > localWeightStored
            && pkt->getWeight() > 0 && pktCharacteristic.size() > 0
            /*&& pkt->getHopCount() <= maxTtl*/) {
        cbrSingleton->getInstance()->insertAdvCharacteristic(
                thisNodeAddress.str(), pktCharacteristic,
                nextNodeAddress.str(), pkt->getWeight(),
                pkt->getWeightOnPropagate(), pkt->getHopCount(),
                pkt->getSeqNum());
        cbrSingleton->getInstance()->printLocalCharacteristicsTable();
        cbrpkt_ptr_t propagateAdvertisement = new cbrpkt_t("ADVERTISE",
                ADVERTISE);
        propagateAdvertisement->setBitLength(10 * 8);
        /* Gather the weight cost from last hop address */
        double weightPropagationCost = pkt->getWeightOnPropagate();
        /* Calculate the weight flow for the next propagation */
        double nextWeightPropagationCost =
                cbrSingleton->getInstance()->prepareWeightCost(rssi,
                        weightPropagationCost);
        int timeToLive = pkt->getHopCount();
        propagateAdvertisement->setDestAddr(LAddress::L3BROADCAST);
        propagateAdvertisement->setSrcAddr(na);
        propagateAdvertisement->setHopCount(++timeToLive);
        propagateAdvertisement->setWeight(weightPropagationCost);
        propagateAdvertisement->setWeightOnPropagate(
                nextWeightPropagationCost);
        propagateAdvertisement->setSeqNum(pkt->getSeqNum());
        propagateAdvertisement->setCharacteristic(
                pktCharacteristic.c_str());
        NetwToMacControlInfo::setControlInfo(propagateAdvertisement,
                LAddress::L2BROADCAST);
        const_cast<cModule*>(getNode())->bubble(
                "Propagating characteristic advertise message #"
                        + propagateAdvertisement->getSeqNum());
        sendDown(propagateAdvertisement);
        if (controlBegin) {
            cntrlOverhead = simTime() - cntrlOverhead;
            testCntrlOh.collect(SIMTIME_DBL(cntrlOverhead));
        }
    }
    /* Now also schedule characteristic advertisement */
    scheduleCharacteristicAdvertisement();
    ohNumberOfControlBits += pkt->getBitLength();
    delete pkt;
```

```
    }
}
/* Timer method which takes care of sending out the
 * ADVERTISE~ment messages for a broadcast */
void CbrNetwLayer::handleAdvertiseCharacteristicsTimerMessage(cMessage* msg,
        double rssi) {
    int localTotalRoutes = cbrSingleton->getInstance()->getTotalAddressRoutes();
    nodeConfCharacteristics_ ncc =
            cbrSingleton->getInstance()->getConfCharacteristics();
    int iterator = 0;
    if (ncc.size() > 0) {
        for (std::map<std::string, characteristicEntries_>::iterator it =
                ncc.begin(); it != ncc.end(); ++it) {
            std::string address = it->first;
            characteristicEntries_ entries = it->second;
            std::stringstream myaddress;
            myaddress << na;
            if (address
                    == myaddress.str() /*&& localTotalRoutes != (numHosts-1)*/) {
                if (entries.size() > 0) {
                    for (std::map<std::string, double>::iterator it1 =
                            entries.begin(); it1 != entries.end(); ++it1) {
                        std::string characteristic = it1->first;
                        if (it1->second > 0L) {
                            /* Now also start characteristic advertisement to neighbours */
                            double weightCharacteristic = it1->second;
                            /* Apply Cost Function */
                            double weightPropagationCost =
                                    cbrSingleton->getInstance()->prepareWeightCost(
                                            rssi, weightCharacteristic);
                            cbrpkt_ptr_t characteristicAdvertisement =
                                    new cbrpkt_t("ADVERTISE", ADVERTISE);
                            characteristicAdvertisement->setBitLength(10 * 8);
                            characteristicAdvertisement->setDestAddr(
                                    LAddress::L3BROADCAST);
                            characteristicAdvertisement->setSrcAddr(na);
                            characteristicAdvertisement->setSeqNum(
                                    runningSeqNumber++);
                            characteristicAdvertisement->setHopCount(1);
                            characteristicAdvertisement->setWeightOnPropagate(
                                    weightPropagationCost);
                            characteristicAdvertisement->setWeight(
                                    weightCharacteristic);
                            characteristicAdvertisement->setCharacteristic(
                                    characteristic.c_str());
                            NetwToMacControlInfo::setControlInfo(
                                    characteristicAdvertisement,
                                    LAddress::L2BROADCAST);
                            const_cast<cModule*>(getNode())->bubble(
                                    "Characteristic Advertise Message #"
                                            + runningSeqNumber);
                            sendDown(characteristicAdvertisement);
                        }
                    }
                }
            }
        }
    }
    /* Now also schedule characteristic advertisement */
    scheduleCharacteristicAdvertisement();
}
/* Method takes care of forwarding the CBRRequests
 * to the strong characteristic source. */
void CbrNetwLayer::handleServiceRequest(cMessage* msg) {
    if (dynamic_cast<ApplPkt*>(msg) != NULL) {
        ApplPkt* requestPacket = static_cast<ApplPkt*>(msg);
        if (requestPacket != NULL
                && (requestPacket->getData()
                        && requestPacket->getData()[0] != '\0')) {
            std::ostringstream logMessage;
            logMessage << "Received service request message for "
                    << requestPacket->getData() << " Source: "
                    << requestPacket->getSrcAddr() << " Dest: "
                    << requestPacket->getDestAddr() << " transaction number "
                    << cbrSingleton->getInstance()->LongToString(
                            requestPacket->getTransactionId()) << endl;
            std::stringstream nodeAddress;
            nodeAddress << na;
            bool reachedSourceNode =
                    cbrSingleton->getInstance()->isConfiguredAsSourceNode(
                            nodeAddress.str(), requestPacket->getData());
            if (reachedSourceNode) {
                std::cout << "Do nothing, source node "
                        << requestPacket->getSrcAddr() << " and host is " << na
                        << endl;
                /* Now send the received message to the upper layer also as we
                 have reached the source node */
                /* ApplPkt* rupmsg = new ApplPkt("REQUEST_CHARACTERISTIC",
                                              REQUEST_CHARACTERISTIC);
                 rupmsg->setBitLength((100 * 8));
                 rupmsg->setData(requestPacket->getData());
                 rupmsg->setSrcAddr(requestPacket->getSrcAddr() );
                 rupmsg->setDestAddr(requestPacket->getDestAddr());
                 rupmsg->setTransactionId(requestPacket->getTransactionId());
```

```
                rupmsg->setSentTime(msg->getCreationTime());
                NetwControlInfo::setControlInfo(rupmsg, requestPacket->getDestAddr());
                send(rupmsg, findGate("upperLayerOut"));*/
            } else {
                if (nodeAddress.str().size() > 0) {
                    std::string nextNodeAddress =
                            cbrSingleton->getInstance()->getNextNodeAddress(
                                    nodeAddress.str(),
                                    requestPacket->getData());
                    if (nextNodeAddress.size() > 0) {
                        cbrreqpkt_t* fwd = new cbrreqpkt_t(
                                "REQUEST_CHARACTERISTIC",
                                REQUEST_CHARACTERISTIC);
                        LAddress::L3Type nextHopL3Address = LAddress::L3Type(
                                atol(nextNodeAddress.c_str()));
                        LAddress::L2Type nextHopL2Address =
                                routingTable[nextHopL3Address];
                        fwd->setSrcAddr(na);
                        fwd->setDestAddr(nextHopL3Address);
                        fwd->setReqSrcAddr(requestPacket->getSrcAddr());
                        fwd->setTransactionId(
                                requestPacket->getTransactionId());
                        fwd->setCharacteristic(requestPacket->getData());
                        fwd->setMessageTime(msg->getCreationTime());
                        std::string rroutePath;
                        std::stringstream requestSourceAddress;
                        rroutePath.append("REQUEST_CHARACTERISTIC_");
                        rroutePath.append(requestPacket->getData());
                        rroutePath.append("_");
                        rroutePath.append(
                                cbrSingleton->getInstance()->LongToString(
                                        requestPacket->getTransactionId()));
                        rroutePath.append(" REQUESTOR ");
                        rroutePath.append(" >> ");
                        requestSourceAddress << requestPacket->getSrcAddr();
                        rroutePath.append(requestSourceAddress.str());
                        rroutePath.append(" >> ");
                        rroutePath.append(nextNodeAddress);
                        fwd->setData(rroutePath.c_str());
                        /*std::cout << "GOT REQUEST from " << requestPacket->getSrcAddr()
                         << " and arrive, host is " << na
                         << " nextNodeAddr is " << nextNodeAddress
                         << " FWD_DEST_ADDR_L3 is " << nextHopL3Address
                         << " FWD_DEST_ADDR_L2 is " << nextHopL2Address
                         << endl;*/
                        /* Make sure to use rolling sequence number from this node
                         * when characteristic based routing begins for the request packet
                         * coming from the upper layer */
                        fwd->setSeqNum(runningSeqNumber++);
                        fwd->setHopCount(1);
                        NetwToMacControlInfo::setControlInfo(fwd,
                                nextHopL2Address);
                        sendDown(fwd);
                    }
                }
            }
        }
        delete msg;
    } else if (dynamic_cast<CbrReqPkt*>(msg) != NULL) {
        cbrreqpkt_t* forwardedPacket = static_cast<cbrreqpkt_t*>(msg);
        std::stringstream nodeAddress;
        nodeAddress << na;
        if (forwardedPacket != NULL) {
            std::ostringstream logMessage;
            logMessage
                    << "2) Received forwarded service request message for characteristic "
                    << forwardedPacket->getCharacteristic()
                    << " source address " << forwardedPacket->getSrcAddr()
                    << " dest address " << forwardedPacket->getDestAddr()
                    << " transaction number "
                    << forwardedPacket->getTransactionId() << endl;
            bool reachedSourceNode =
                    cbrSingleton->getInstance()->isConfiguredAsSourceNode(
                            nodeAddress.str(),
                            forwardedPacket->getCharacteristic());
            if (reachedSourceNode) {
                /* Display now the request message path vector */
                std::ostringstream displayPathVector;
                displayPathVector << forwardedPacket->getData();
                printLog(displayPathVector.str());
                /* As we have reached the source node for the
                                 * characteristic request, prepare a reply */
                cbrreppkt_t* replyBack = new cbrreppkt_t("REPLY_CHARACTERISTIC",
                        REPLY_CHARACTERISTIC);
                LAddress::L3Type lastHopL3Address =
                        forwardedPacket->getSrcAddr();
                LAddress::L2Type lastHopL2Address =
                        routingTable[lastHopL3Address];
                replyBack->setSrcAddr(na);
                replyBack->setDestAddr(lastHopL3Address);
                replyBack->setReqSrcAddr(forwardedPacket->getReqSrcAddr());
                replyBack->setTransactionId(
                        forwardedPacket->getTransactionId());
```

```
            replyBack->setCharacteristic(
                    forwardedPacket->getCharacteristic());
            replyBack->setMessageTime(forwardedPacket->getMessageTime());
            /* Make sure to use rolling sequence number from this node for this reply packet */
            replyBack->setSeqNum(runningSeqNumber++);
            replyBack->setHopCount(1);
            LAddress::L2Type macAddressReached =
                    cbrSingleton->getInstance()->getMacAddressForNode(na);
            std::ostringstream nodeMacAddress1;
            nodeMacAddress1 << macAddressReached;
            std::string nodeMacAddress2;
            nodeMacAddress2.append(nodeMacAddress1.str());
            std::string payload;
            payload.append(forwardedPacket->getCharacteristic());
            payload.append(
                    " service can be reached and the service MAC address is ");
            payload.append(nodeMacAddress2.c_str());
            replyBack->setPayload(payload.c_str());
            /* Send back reply message and set the reply route */
            std::string rproutePath;
            std::stringstream sourceNode;
            sourceNode << na;
            std::stringstream forwardNode;
            forwardNode << lastHopL3Address;
            rproutePath.append("REPLY_CHARACTERISTIC_");
            rproutePath.append(forwardedPacket->getCharacteristic());
            rproutePath.append("_");
            rproutePath.append(
                    cbrSingleton->getInstance()->LongToString(
                            forwardedPacket->getTransactionId()));
            rproutePath.append("   SOURCE ");
            rproutePath.append(" << ");
            rproutePath.append(sourceNode.str());
            rproutePath.append(" << ");
            rproutePath.append(forwardNode.str());
            replyBack->setData(rproutePath.c_str());
            NetwToMacControlInfo::setControlInfo(replyBack,
                    lastHopL2Address);
            // sendDown(replyBack);
            /* Now send the received message to the upper layer also as
             * we have reached the source node */
            ApplPkt* rupmsg = new ApplPkt("REQUEST_CHARACTERISTIC",
                    REQUEST_CHARACTERISTIC);
            rupmsg->setBitLength((100 * 8));
            rupmsg->setData(forwardedPacket->getCharacteristic());
            rupmsg->setSrcAddr(forwardedPacket->getSrcAddr());
            rupmsg->setDestAddr(forwardedPacket->getDestAddr());
            rupmsg->setTransactionId(forwardedPacket->getTransactionId());
            rupmsg->setSentTime(forwardedPacket->getMessageTime());
            NetwControlInfo::setControlInfo(rupmsg,
                    forwardedPacket->getDestAddr());
            /* Send it back to the application layer above */
            send(rupmsg, findGate("upperLayerOut"));
        } else {
            if (nodeAddress.str().size() > 0) {
                std::string nextNodeAddress =
                        cbrSingleton->getInstance()->getNextNodeAddress(
                                nodeAddress.str(),
                                forwardedPacket->getCharacteristic());
                if (nextNodeAddress.size() > 0) {
                    cbrreqpkt_t* fwd = new cbrreqpkt_t(
                            "REQUEST_CHARACTERISTIC",
                            REQUEST_CHARACTERISTIC);
                    LAddress::L3Type nextHopL3Address = LAddress::L3Type(
                            atol(nextNodeAddress.c_str()));
                    LAddress::L2Type nextHopL2Address =
                            routingTable[nextHopL3Address];
                    fwd->setSrcAddr(na);
                    fwd->setDestAddr(nextHopL3Address);
                    fwd->setReqSrcAddr(forwardedPacket->getReqSrcAddr());
                    fwd->setTransactionId(
                            forwardedPacket->getTransactionId());
                    fwd->setCharacteristic(
                            forwardedPacket->getCharacteristic());
                    fwd->setSeqNum(forwardedPacket->getSeqNum());
                    fwd->setHopCount(forwardedPacket->getHopCount() + 1);
                    fwd->setMessageTime(forwardedPacket->getMessageTime());
                    /* Set the flowing route path */
                    std::ostringstream routePath;
                    routePath << forwardedPacket->getData();
                    std::string routePath1;
                    routePath1.append(routePath.str());
                    routePath1.append(" >> " + nextNodeAddress);
                    fwd->setData(routePath1.c_str());
                    NetwToMacControlInfo::setControlInfo(fwd,
                            nextHopL2Address);
                    sendDown(fwd);
                }
            }
        }
        delete forwardedPacket;
    }
```

```
    }
}
/* Method takes care of forwarding the service
 * reply message in a reverse forward path.
 */
void CbrNetwLayer::handleServiceReply(cMessage* msg) {
    if (dynamic_cast<CbrRepPkt*>(msg) != NULL) {
        cbrreppkt_t* replyPacket = static_cast<cbrreppkt_t*>(msg);
        std::stringstream nodeAddress;
        nodeAddress << na;
        if (replyPacket != NULL) {
            if (na == replyPacket->getReqSrcAddr()) {
                /* Display now the reply message path vector */
                std::ostringstream displayPathVector;
                displayPathVector << replyPacket->getData();
                printLog(displayPathVector.str());
                // create and send a new message
                ApplPkt* rmsg = new ApplPkt("REPLY_CHARACTERISTIC",
                        REPLY_CHARACTERISTIC);
                rmsg->setBitLength((100 * 8));
                rmsg->setData(replyPacket->getCharacteristic());
                rmsg->setSrcAddr(replyPacket->getSrcAddr());
                rmsg->setDestAddr(replyPacket->getDestAddr());
                rmsg->setTransactionId(replyPacket->getTransactionId());
                rmsg->setSentTime(replyPacket->getMessageTime());
                NetwControlInfo::setControlInfo(rmsg,
                        replyPacket->getDestAddr());
                /* Send it back to the application layer above */
                send(rmsg, findGate("upperLayerOut"));
            } else {
                /* Get the next hop relaying address from the path vector */
                std::string forwardingNodeAddress =
                        cbrSingleton->getInstance()->getForwardingNodeAddress(
                                nodeAddress.str(),
                                replyPacket->getCharacteristic());
                if (!forwardingNodeAddress.empty()) {
                    /* As we have reached the source node for the characteristic
                     request, prepare a reply */
                    cbrreppkt_t* relayNextTo = new cbrreppkt_t(
                            "REPLY_CHARACTERISTIC", REPLY_CHARACTERISTIC);
                    LAddress::L3Type relayToL3Address = LAddress::L3Type(
                            atol(forwardingNodeAddress.c_str()));
                    LAddress::L2Type relayToL2Address =
                            routingTable[relayToL3Address];
                    relayNextTo->setSrcAddr(na);
                    relayNextTo->setDestAddr(relayToL3Address);
                    relayNextTo->setReqSrcAddr(replyPacket->getReqSrcAddr());
                    relayNextTo->setTransactionId(
                            replyPacket->getTransactionId());
                    relayNextTo->setCharacteristic(
                            replyPacket->getCharacteristic());
                    /* Make sure to use rolling sequence number from this
                     node for this reply packet */
                    relayNextTo->setSeqNum(replyPacket->getSeqNum());
                    relayNextTo->setHopCount(relayNextTo->getHopCount() + 1);
                    relayNextTo->setPayload(replyPacket->getPayload());
                    relayNextTo->setMessageTime(replyPacket->getMessageTime());
                    /* Set the reverse route path */
                    std::ostringstream revRoutePath;
                    std::stringstream relayNode;
                    relayNode << relayToL3Address;
                    revRoutePath << replyPacket->getData();
                    std::string routePath2;
                    routePath2.append(revRoutePath.str());
                    routePath2.append(" << " + relayNode.str());
                    relayNextTo->setData(routePath2.c_str());
                    NetwToMacControlInfo::setControlInfo(relayNextTo,
                            relayToL2Address);
                    sendDown(relayNextTo);
                }
            }
        }
        delete replyPacket;
    }
}
void CbrNetwLayer::printLog(std::string logStatement) {
    EV<< logStatement << endl;
    std::cout << logStatement << endl;
}
/* Gather our metric inside for the control overhead */
void CbrNetwLayer::finish() {
    recordScalar("Netw-Control-Overhead", testCntrlOh.getMean());
    recordScalar("Control-Data-Bits", ohNumberOfControlBits);
}
/* Method is called during our simulation run and our
 * network protocol module gets initialized */
void CbrNetwLayer::initialize(int stage) {
    if (stage == 0) {
        dataOut = findGate("lowerLayerOut");
        dataIn = findGate("lowerLayerIn");
        /* FindModule<>::findHost(this)->getDisplayString().
```

```
        setTagArg("i", 0, "device/accesspoint");*/
        numHosts = par("numHosts");
        std::cout << "Number of hosts configured " << numHosts << endl;
        na = LAddress::L3Type(par("na").longValue());
        maxTtl = par("maxTtl").longValue();
        boredTime = par("boredTime").doubleValue();
        /* Read this nodes main characteristics from omnetpp.ini */
        cbrSingleton->getInstance()->populateConfCharacteristics(
                par("na").str(), par("nodeCharacteristics"));
        cbrSingleton->getInstance()->printConfCharacteristics();
        cntrlOverhead = simTime();
    } else if (stage == 1) {
        startCharacteristicAdvertisement = new cMessage(
                "ADVERTISE_CHARACTERISTICS_TIMER",
                ADVERTISE_CHARACTERISTICS_TIMER);
        broadcastAdvertisement();
    }
}
/* Method which is called for processing all incoming
 * and outgoing traffic (Adevrtise, CBRRequest, CBRReply)
 * packets inside our network protocol layer
 *
 * This is where all the characteristic based routing
 * occurs. */
void CbrNetwLayer::handleMessage(cMessage* msg) {
    /* Ignore RSSI */
    double rssi = 0.0;
    switch (msg->getKind()) {
    case ADVERTISE:
        handleAdvertise(msg, static_cast<cbrpkt_ptr_t>(msg), rssi);
        break;
    case ADVERTISE_CHARACTERISTICS_TIMER:
        handleAdvertiseCharacteristicsTimerMessage(msg, rssi);
        break;
    case REQUEST_CHARACTERISTIC:
        handleServiceRequest(msg);
        break;
    case REPLY_CHARACTERISTIC:
        handleServiceReply(msg);
        break;
    case BaseMacLayer::PACKET_DROPPED:
        printLog("Packet dropped by MAC layer.");
        delete msg;
        break;
    default:
        error("unknown packet type of packet %s", msg->getName());
        break;
    }
}
```

A.2 Application Layer Module

A.2.1 CbrApplLayer.CC

```
#include "CbrApplLayer.h"
#include "BaseMacLayer.h"
#include "NetwControlInfo.h"
#include "ApplPkt_m.h"
Define_Module(CbrApplLayer);
/* Method is called when our simulation starts
 * Parameters are initialized from our omnet.ini
 * conf file */
void CbrApplLayer::initialize(int stage) {
    BaseModule::initialize(stage);
    if (stage == 0) {
        /* Begin by connecting the gates
           to allow messages exchange */
        dataOut = findGate("lowerLayerOut");
        dataIn = findGate("lowerLayerIn");
        ctrlOut = findGate("lowerControlOut");
        ctrlIn = findGate("lowerControlIn");
        // Retrieve parameters
        debug = par("debug").boolValue();
        stats = par("stats").boolValue();
        trace = par("trace").boolValue();
        isTransmitting = false;
        nbPackets = par("nbPackets");
        trafficParam = par("trafficParam").doubleValue();
        nodeAddr = LAddress::L3Type(par("nodeAddr").longValue());
        dstAddr = LAddress::L3Type(par("dstAddr").longValue());
        flood = par("flood").boolValue();
        PAYLOAD_SIZE = par("payloadSize"); // data field size
        PAYLOAD_SIZE = PAYLOAD_SIZE * 8; // convert to bits
        /* Configure internal state variables and objects */
        numberTotalReqSent = 0;
        numberTotalReqReceived = 0;
```

```
        controlOverhead = simTime();
        nbPacketsReceived = 0;
        remainingPackets = nbPackets;
        INITIAL_DELAY = 5; // initial delay before sending first packet
        /* Start timer if needed */
        if (nodeAddr != 0 && remainingPackets > 0) {
            delayTimer = new cMessage("app-delay-timer");
            /* we add a small shift to avoid systematic collisions */
            scheduleAt(simTime() + INITIAL_DELAY + uniform(0, 0.001),
                    delayTimer);
        } else {
            delayTimer = 0;
        }
        if (stats) {
            /* we should collect statistics */
            int nbNodes = getNode()->size();
            for (int i = 0; i < nbNodes; i++) {
                std::ostringstream oss;
                oss << "latency";
                oss << i;
                cStdDev aLatency(oss.str().c_str());
                latencies.push_back(aLatency);
            }
        }
        if (trace) {
            /* record all packet arrivals */
            latenciesRaw.setName("rawLatencies");
            oneEndLatenciesRaw.setName("OneEndRawLatencies");
        }
        scheduleDelayTime();
    }
}
CbrApplLayer::~CbrApplLayer() {
    if (delayTimer)
        cancelAndDelete(delayTimer);
}
/* Method called to tidy things up and record all
 * our required statistics for results */
void CbrApplLayer::finish() {
    if (stats) {
        recordScalar("Total-Requests-Received", numberTotalReqReceived);
        recordScalar("Total-Requests-Sent", numberTotalReqSent);
        recordScalar("Packets-Received", nbPacketsReceived);
        recordScalar("Aggregate-Latency", testStat.getMean());
        recordScalar("Control-Overhead", testControl.getMean());
        for (unsigned int i = 0; i < latencies.size(); i++) {
            cStdDev aLatency = latencies[i];
            aLatency.record();
        }
        /* Throughput Time */
        recordScalar("First-Packet-Sent-Time", firstCbrPacketTime);
        recordScalar("Last-Packet-Receive-Time", lastCbrPacketTime);
        recordScalar("Total-Data-Bits", tpTotalNumberOfBits);
    }
}
/* Method to manage the delay timer for sending out the
 * CBRRequests */
void CbrApplLayer::scheduleDelayTime() {
    if (delayTimer->isScheduled()) {
        cancelEvent(delayTimer);
    }
    scheduleAt(simTime() + (rand()%(9-1 + 1) + 1), delayTimer);
}
/* Method which is called when a CBRRequest packet arrives into
 * our source node's application layer, where other nodes
 * sending out the CBRRequest packets for a characteristic,
 * It generates all our statistics for the latency, throughput,
 * overhead and successrate.
 *
 * Also gather relevant statistics for the CBRReply packet
 * arriving from the strong characteristic source node */
void CbrApplLayer::handleMessage(cMessage * msg) {
    debugEV << "In CbrApplLayer::handleMessage" << endl;
    if (msg == delayTimer) {
        if (debug) {
            debugEV << "  processing application timer." << endl;
        }
        std::stringstream thisNodeAddress;
        thisNodeAddress << nodeAddr;
        std::string strNodeAddress;
        strNodeAddress = thisNodeAddress.str();
        if (!isTransmitting && flood ) {
            /*const long double sysTime = time(0);
              const long double sysTimeMS = sysTime*1000;*/
            // create and send a new message
            ApplPkt* msg = new ApplPkt("REQUEST_CHARACTERISTIC",
                    REQUEST_CHARACTERISTIC);
            msg->setBitLength(PAYLOAD_SIZE);
            /* Now send out random characteristic request */
            int randCharacteristic = rand()%(2-1 + 1) + 1;
            if (randCharacteristic == 1)
                msg->setData("AAAABBAA");
            else if (randCharacteristic == 2)
                msg->setData("AAAABBBB");
```

```
            else if (randCharacteristic == 3)
                msg->setData("AABBBBAA");
            msg->setSrcAddr(nodeAddr);
            msg->setDestAddr(LAddress::L3BROADCAST);
            msg->setTransactionId(
                    ((transactionid--) + atol(strNodeAddress.c_str())));
            /* Broadcast it to neighbours */
            NetwControlInfo::setControlInfo(msg, LAddress::L3BROADCAST);
            if (debug) {
                debugEV << " sending down new data message to"
                        << " MAC layer for radio transmission."
                        << endl;
            }
            send(msg, dataOut);
            numberTotalReqSent++;
            tpTotalNumberOfBits += msg->getBitLength();
            if (!controlBegin) {
                controlOverhead = simTime() - controlOverhead;
                testControl.collect(SIMTIME_DBL(controlOverhead));
                controlBegin = true;
            }
            if (!sentFirstCbrPacketTime) {
                firstCbrPacketTime = simTime();
                sentFirstCbrPacketTime = true;
            }
            /* update internal state */
            remainingPackets--;
        }
        /* reschedule timer if appropriate */
        if (remainingPackets > 0) {
            if (!flood && !delayTimer->isScheduled()) {
                scheduleAt(simTime() + exponential(trafficParam) + 0.001,
                        delayTimer);
            }
        } else {
            cancelAndDelete(delayTimer);
            delayTimer = 0;
        }
    } else if (msg->getArrivalGateId() == dataIn) {
        if (msg->getKind() == REPLY_CHARACTERISTIC) {
            /* We received a data message from someone else !*/
            ApplPkt* m = dynamic_cast<ApplPkt*>(msg);
            if (debug)
                debugEV << "I (" << nodeAddr << ") received a message from node "
                                << m->getSrcAddr() << " of size "
                                << m->getBitLength() << "." << endl;
            nbPacketsReceived++;
            if (stats) {
                simtime_t theLatency = msg->getArrivalTime() - m->getSentTime();
                latencies[m->getSrcAddr()].collect(SIMTIME_DBL(theLatency));
                testStat.collect(SIMTIME_DBL(theLatency));
            }
            if (trace) {
                simtime_t theLatency = msg->getArrivalTime() - m->getSentTime();
                latenciesRaw.record(SIMTIME_DBL(theLatency));
            }
            delete msg;
        } else if (msg->getKind() == REQUEST_CHARACTERISTIC) {
            /* we received a data message from someone else ! */
            ApplPkt* m = dynamic_cast<ApplPkt*>(msg);
            numberTotalReqReceived++;
            if (trace) {
                simtime_t theLatency = msg->getArrivalTime() - m->getSentTime();
                oneEndLatenciesRaw.record(SIMTIME_DBL(theLatency));
            }
            tpTotalNumberOfBits += m->getBitLength();
            lastCbrPacketTime = simTime();
            delete msg;
        }
    } else if (msg->getArrivalGateId() == ctrlIn) {
        debugEV << "Received a control message." << endl;
        /* msg announces end of transmission. */
        if (msg->getKind() == BaseMacLayer::TX_OVER) {
            isTransmitting = false;
            if (remainingPackets > 0 && flood && !delayTimer->isScheduled()) {
                scheduleAt(simTime() + 0.001 * 001 + uniform(0, 0.001 * 0.001),
                        delayTimer);
            }
        }
        delete msg;
    } else {
        if (debug) {
            ApplPkt* m = static_cast<ApplPkt*>(msg);
            debugEV << "I (" << nodeAddr << ") received a message from node "
                            << (static_cast<ApplPkt*>(msg))->getSrcAddr()
                            << " of size " << m->getBitLength() << "." << endl;
        }
        delete msg;
    }
    scheduleDelayTime();
}
```

A.3 Network Control

A.3.1 CbrSingleton.CC

```
#ifndef __CBRSINGLETON_H__
#define __CBRSINGLETON_H__
using namespace std;
/* Local Characteristics Table Structure */
struct localCharacteristicEntry {
        std::string nextWeightAddress;
        double nextWeight;
        double weight;
        int hopCount;
        unsigned long seqNumber;
        localCharacteristicEntry(std::string p1="", double p2=0, double p3=0,
                        int p4=0, unsigned long p5=0):nextWeightAddress(p1),
                nextWeight(p2),
                weight(p3),
                hopCount(p4),
                seqNumber(p5){}
};
/* Map for storing the characteristics packets received for a particular node */
typedef std::map<std::string, struct localCharacteristicEntry> mapCharacteristicEntries_;
typedef std::map<std::string, mapCharacteristicEntries_> localCharacteristicsMap_;
typedef std::map<std::string, double> characteristicEntries_;
typedef std::map<std::string, characteristicEntries_> nodeConfCharacteristics_;
typedef std::map<LAddress::L3Type, LAddress::L2Type> addressTable_;
class CBRSingleton
{
private:
    nodeConfCharacteristics_ ncCharac_;
    localCharacteristicsMap_ localCharacteristicsTable_;
    addressTable_ macAddressTable_;
    friend class CBR_CharacteristicLinkTimer;
    static bool instanceFlag;
    static CBRSingleton *single;
    CBRSingleton()
    {
        //private constructor
    }
public:
    double CONST_COST_GRADIENT = 0.5;
    static CBRSingleton* getInstance();
    void method();
    void populateConfCharacteristics(std::string nodeAddress, const char *nodeCharacs);
    nodeConfCharacteristics_ getConfCharacteristics();
    void printConfCharacteristics();
    double getNodeConfCharacteristicWeight(std::string nodeAddress, std::string characteristic);
    double getLocalNodeStoredCharacteristicWeight(std::string localNodeAddress,
                std::string characteristic);
    bool isConfiguredAsSourceNode(std::string nodeAddress, std::string characteristic);
    std::string getNextNodeAddress(std::string nodeAddress, std::string characteristic);
    std::string getPrintStringForDouble(double weight);
    std::string getPrintStringForString(std::string str);
    std::string getForwardingNodeAddress(std::string nodeAddress, std::string characteristic);
    double prepareWeightCost(double rssiFeed, double characteristicWeight);
    std::string LongToString(long value);
    void insertIntoAddressTable(LAddress::L3Type level3Address, LAddress::L2Type level2Address);
    LAddress::L2Type getMacAddressForNode(LAddress::L3Type level3Address);
    void insertAdvCharacteristic(std::string nodeAddress, std::string characteristic,
         std::string nextWeightAddress, double nextWeight, double weight, int hopCount,
                 long seqNumber);
    void printLocalCharacteristicsTable();
    int getTotalAddressRoutes();
    double gradientCost(double input);
    /* Fast inverse square root function */
    float inverseSquareRoot( float number ) {
            long i;
            float x2, y;
            const float threehalfs = 1.5F;
            x2 = number * 0.5F;
            y  = number;
            i  = * ( long * ) &y;
            i  = 0x5f3759df - ( i >> 1 );
            y  = * ( float * ) &i;
            y  = y * ( threehalfs - ( x2 * y * y ) );
            return y;
    }
    ~CBRSingleton()
    {
        instanceFlag = false;
    }
};
bool CBRSingleton::instanceFlag = false;
CBRSingleton* CBRSingleton::single = NULL;
/*
 * Make sure to carry out protocol logic
 * in a singleton instance. Keep tidy.
 */
CBRSingleton* CBRSingleton::getInstance()
```

```
{
    if(! instanceFlag)
    {
        single = new CBRSingleton();
        instanceFlag = true;
        return single;
    }
    else
    {
        return single;
    }
}
void CBRSingleton::method()
{
    cout << "Method of the singleton class" << endl;
}
/*
 * MAC address routing table, populate method
 */
void CBRSingleton::insertIntoAddressTable(LAddress::L3Type level3Address,
                LAddress::L2Type level2Address) {
    macAddressTable_.insert(std::pair<LAddress::L3Type, LAddress::L2Type>
                        (level3Address, level2Address));
}
/*
 * Method which gets the Level2 MAC Address for the
 * specified Level3 Address.
 * Used for spreading the characteristics to the single
 * hop neighbours during a broadcast
 */
LAddress::L2Type CBRSingleton::getMacAddressForNode(LAddress::L3Type level3Address) {
    LAddress::L2Type l2MacAddress;
    if (macAddressTable_.size() > 0) {
        addressTable_::iterator it= macAddressTable_.find(level3Address);
        if( it != macAddressTable_.end() ) {
            l2MacAddress  =  it->second;
        }
    }
    return l2MacAddress;
}
/*
 * Method which adds entries separately into the
 * configuration characteristics table for the
 * association check
 */
void CBRSingleton::populateConfCharacteristics(std::string nodeAddress,
        const char *nodeCharacs) {
    std::string thisNodeAddress = nodeAddress;
    characteristicEntries_ characteristicEntry;
    cStringTokenizer tokenizer(nodeCharacs);
    const char *token;
    while ((token = tokenizer.nextToken()) != NULL) {
        std::string elem1, elem2, currToken;
        currToken = token;
        char sep = ':';
        int t=0;
        /* split using the ':' seperator and extract the characteristic
                                                name and weight */
        for(size_t p=0, q=0; p!=currToken.npos; p=q) {
            if (t==0) {
                elem1 = currToken.substr(p+(p!=0),
                            (q=currToken.find(sep, p+1))-p-(p!=0));
                t++;
            } else if (t==1) {
                elem2 = currToken.substr(p+(p!=0),
                                        (q=currToken.find(sep, p+1))-p-(p!=0));
                characteristicEntry.insert(std::pair<std::string, double>
                                        (elem1, atof(elem2.c_str())));
                elem1 = "";
                elem2 = "";
                t=0;
            }
        }
    }
    ncCharac_.insert(std::pair<std::string, characteristicEntries_>
                                        (thisNodeAddress, characteristicEntry));
}
nodeConfCharacteristics_ CBRSingleton::getConfCharacteristics() {
    return ncCharac_;
}
/*
 * Utility method for printing all the associated characteristics
 * which are picked up from omnet configuration file for the
 * simulation run.
 */
void CBRSingleton::printConfCharacteristics() {
    for (std::map<std::string,
                characteristicEntries_>::iterator it=ncCharac_.begin();
                it!=ncCharac_.end(); ++it) {
        std::string nAddress    = it->first;
        std::map<std::string, double> naCharacteristics  =  it->second;
        for (std::map<std::string, double>::iterator
                        it1=naCharacteristics.begin();
                        it1!=naCharacteristics.end(); ++it1) {
            std::string characteristic  = it1->first;
            double weight =  it1->second;
        }
    }
```

```
}
/*
 * Method which gets the associated characteristic weight
 * for the gradient source.
 */
double CBRSingleton::getNodeConfCharacteristicWeight(
        std::string nodeAddress, std::string characteristic) {
    std::string lookupAddress = nodeAddress;
    double weight=0;
    if (ncCharac_.size() > 0) {
        nodeConfCharacteristics_::iterator it= ncCharac_.find(lookupAddress);
        if( it != ncCharac_.end() ) {
           characteristicEntries_ entries  =  it->second;
           if (entries.size() > 0) {
               characteristicEntries_::iterator it1= entries.find(characteristic);
               if (it1 != entries.end()) {
                   double w  =  it1->second;
                   if (w > 0)
                       weight = w;
               }
           }
        }
    }
    return weight;
}
/*
 * Method which gets all the characteristic entries
 * for a node address and the given characteristic
 */
double CBRSingleton::getLocalNodeStoredCharacteristicWeight(
        std::string localNodeAddress, std::string search_characteristic) {
    std::string lookupAddress = localNodeAddress;
    bool foundNodeWeight = false;
    double weight=0;
    for (std::map<std::string,
                mapCharacteristicEntries_>::iterator
                it=localCharacteristicsTable_.begin();
                it!=localCharacteristicsTable_.end(); ++it) {
        std::string mainAddress    = it->first;
        mapCharacteristicEntries_ characteristicsEntries  =  it->second;
        if (mainAddress == lookupAddress) {
            for (std::map<std::string,
                                struct localCharacteristicEntry>::iterator it1=
                                characteristicsEntries.begin();
                                it1!=characteristicsEntries.end(); ++it1) {
                std::string characteristic  = it1->first;
                localCharacteristicEntry entry =  it1->second;
                if (characteristic == search_characteristic) {
                    foundNodeWeight = true;
                    weight = entry.weight;
                    break;
                }
            }
        }
        if (foundNodeWeight)
            break;
    }
    return weight;
}
/*
 * Method checks whether the node is indeed a strong
 * gradient source for a characteristic
 */
bool CBRSingleton::isConfiguredAsSourceNode(std::string nodeAddress,
        std::string characteristic) {
    std::string lookupAddress = nodeAddress;
    bool isNodeConfigured=false;
    if (ncCharac_.size() > 0) {
        nodeConfCharacteristics_::iterator it= ncCharac_.find(lookupAddress);
        if( it != ncCharac_.end() ) {
           characteristicEntries_ entries  =  it->second;
           if (entries.size() > 0) {
               characteristicEntries_::iterator it1= entries.find(characteristic);
               if (it1 != entries.end()) {
                   double w  =  it1->second;
                   if (w > 0)
                       isNodeConfigured = true;
               }
           }
        }
    }
    return isNodeConfigured;
}
/*
 * Method for getting the next node address
 * when propagtaing towards the strong
 * gradient source
 */
std::string CBRSingleton::getNextNodeAddress(std::string nodeAddress,
        std::string characteristic) {
    std::string lookupAddress = nodeAddress;
    std::string nextNodeAddress = "";
    if (localCharacteristicsTable_.size() > 0) {
        for (std::map<std::string,
                        mapCharacteristicEntries_>::iterator it=
                        localCharacteristicsTable_.begin();
                        it!=localCharacteristicsTable_.end(); ++it) {
```

```
            mapCharacteristicEntries_ characteristicsEntries  =  it->second;
            std::string mainAddress    = it->first;
            if (mainAddress == lookupAddress) {
                for (std::map<std::string,
                                    struct localCharacteristicEntry>::iterator it1=
                                    characteristicsEntries.begin();
                                    it1!=characteristicsEntries.end(); ++it1) {
                    std::string charact  = it1->first;
                    localCharacteristicEntry entry =  it1->second;
                    if (charact == characteristic) {
                        nextNodeAddress = entry.nextWeightAddress;
                        break;
                    }
                }
            }
        }
    }
    return nextNodeAddress;
}
/*
 * Method for acquiring the reverse route path
 */
std::string CBRSingleton::getForwardingNodeAddress(std::string nodeAddress,
                                                                    std::string characteristic) {
    std::string forwardingNodeAddress = "";
    bool matchNodeAddress = false;
    if (localCharacteristicsTable_.size() > 0) {
        for (std::map<std::string,
                        mapCharacteristicEntries_>::iterator it=
                        localCharacteristicsTable_.begin();
                        it!=localCharacteristicsTable_.end(); ++it) {
            mapCharacteristicEntries_ characteristicsEntries  =  it->second;
            forwardingNodeAddress    = it->first;
            for (std::map<std::string,
                            struct localCharacteristicEntry>::iterator it1=
                            characteristicsEntries.begin();
                            it1!=characteristicsEntries.end(); ++it1) {
                std::string charact  = it1->first;
                localCharacteristicEntry entry =  it1->second;
                if (charact == characteristic && nodeAddress == entry.nextWeightAddress) {
                    matchNodeAddress = true;
                    break;
                }
            }
            if (matchNodeAddress)
                break;
        }
    }
    /* Now lets look into reverse path technique */
    if (!matchNodeAddress) {
        forwardingNodeAddress = "";
        /*std::string nextNode = getNextNodeAddress(nodeAddress, characteristic);
        long nAddress = atol(nextNode.c_str());
        nAddress = nAddress + 1;
        std::ostringstream nextNAddress;
        nextNAddress << nAddress;
        if (localCharacteristicsTable_.size() > 0) {
            for (std::map<std::string, >::iterator
                            it=localCharacteristicsTable_.begin();
                            it!=localCharacteristicsTable_.end(); ++it) {
                mapCharacteristicEntries_ characteristicsEntries  =  it->second;
                forwardingNodeAddress    = it->first;
                for (std::map<std::string,
                                    struct localCharacteristicEntry>::iterator it1=
                                    characteristicsEntries.begin();
                                    it1!=characteristicsEntries.end(); ++it1) {
                    std::string charact  = it1->first;
                    localCharacteristicEntry entry =  it1->second;
                    if (charact == characteristic
                                            && nextNode == entry.nextWeightAddress
                                            && nextNAddress.str() == forwardingNodeAddress) {
                        matchNodeAddress = true;
                        break;
                    }
                }
                if (matchNodeAddress)
                    break;
            }
        }*/
    }
    return forwardingNodeAddress;
}
double CBRSingleton::gradientCost(double input)
{
    int i, j, toggle = 1;
    double angle, product, sum = 0.0;
    angle = input;
    while (angle >= (2*M_PI))
        angle -= (2*M_PI);
    for (i = 0; i < 100; i++)
    {
        product = 1.0;
        for (j = (2*i); j > 0 ; j--)
            product *= angle / j;
        sum +=  product * toggle;
        toggle = -toggle;
    }
```

```
    return sum;
}
/*
 * Method for the calculation of our
 * Gradient weight cost
 */
double CBRSingleton::prepareWeightCost(double rssiFeed, double characteristicWeight) {
    double costGradient = 0;
    double wDiff = sin(characteristicWeight*PI/50000);
    costGradient = characteristicWeight - wDiff - CONST_COST_GRADIENT;
    return costGradient;
}
/*
 * Populates our local characteristic table
 */
void CBRSingleton::insertAdvCharacteristic(std::string nodeAddress,
                        std::string characteristic, std::string nextWeightAddress,
                                double nextWeight, double weight, int hopCount, long seqNumber) {
    struct localCharacteristicEntry lcEntry;
    lcEntry.nextWeightAddress = nextWeightAddress;
    lcEntry.nextWeight = nextWeight;
    lcEntry.weight = weight;
    lcEntry.hopCount = hopCount;
    lcEntry.seqNumber = seqNumber;
    mapCharacteristicEntries_ mcEntries;
    if (localCharacteristicsTable_.find(nodeAddress)
            == localCharacteristicsTable_.end()) {
        /* Not found */
        mcEntries.insert(std::pair<std::string, struct localCharacteristicEntry>
                                                (characteristic, lcEntry));
        localCharacteristicsTable_.insert(std::pair<std::string,
                                                mapCharacteristicEntries_>(nodeAddress, mcEntries));
    } else {
        /* Found */
        localCharacteristicsMap_::iterator it = localCharacteristicsTable_.find(nodeAddress);
        if (it != localCharacteristicsTable_.end()) {
            // Try to update it now with new values
            std::map<std::string, struct localCharacteristicEntry>::iterator it1;
            // Update map for characteristic
            it1 = it->second.find(characteristic);
            if (it1 == it->second.end()) {
                // entry not present therefore insert new entry
                it->second.insert(std::pair<std::string, struct localCharacteristicEntry>
                                                                (characteristic, lcEntry));
            } else {
                // entry is present, therefore update it
                mcEntries.insert(std::pair<std::string, struct localCharacteristicEntry>
                                                                (characteristic, lcEntry));
                it->second = mcEntries;
            }
        }
    }
}
std::string CBRSingleton::LongToString(long value)
{
  std::string mystring;
  stringstream mystream;
  mystream << value;
  mystring = mystream.str();
  return mystring;
}
/*
 * Utility method for number padding
 * for keeping values aligned in the
 * display table
 */
std::string CBRSingleton::getPrintStringForDouble(double weight) {
    std::ostringstream sstream;
    sstream << weight;
    std::string weightString = sstream.str();
    std::string paddedString;
    paddedString.append("  ");
    paddedString.append(weightString);
    int paddedSpaces = (15 - weightString.size());
    for (int p=0; p < paddedSpaces; p++)
        paddedString.append(" ");
    return paddedString;
}
/*
 * Utility method for string padding
 * for keeping values aligned in the
 * display table
 */
std::string CBRSingleton::getPrintStringForString(std::string str) {
    std::string paddedString;
    paddedString.append("  ");
    paddedString.append(str);
    int paddedSpaces = (15 - str.size());
    for (int p=0; p < paddedSpaces; p++)
        paddedString.append(" ");
    return paddedString;
}
/*
 * Print our local characteristics table
 * to std output stream for display
 */
```

```
void CBRSingleton::printLocalCharacteristicsTable() {
 std::cout << "+-----------------+-----------------+-----------------+-----------------"
 << "+-----------------+"
 << endl;
 std::cout << "|      Total      |   Node Address  | Characteristic  |NextWeightAddress|"
 << "Next Weight   |      Weight     |     Hop Count   | Sequence Number |"
 << endl;
 std::cout << "+-----------------+-----------------+-----------------+-----------------"
 << "+-----------------+"
 << endl;
    for (std::map<std::string, mapCharacteristicEntries_>::iterator it=
                        localCharacteristicsTable_.begin();
                        it!=localCharacteristicsTable_.end(); ++it) {
        std::string mainAddress    = it->first;
        mapCharacteristicEntries_ characteristicsEntries  =  it->second;
        for (std::map<std::string, struct localCharacteristicEntry>::iterator it1=
                              characteristicsEntries.begin();
                              it1!=characteristicsEntries.end(); ++it1) {
            std::string characteristic  = it1->first;
            localCharacteristicEntry entry =  it1->second;
            std::cout << "|" <<  getPrintStringForDouble(localCharacteristicsTable_.size())
                      << "|" <<  getPrintStringForString(mainAddress)
                      << "|" <<  getPrintStringForString(characteristic)
                      << "|" <<  getPrintStringForString(entry.nextWeightAddress)
                      << "|" <<  getPrintStringForDouble(entry.nextWeight)
                      << "|" <<  getPrintStringForDouble(entry.weight)
                      << "|" <<  getPrintStringForDouble(entry.hopCount)
                      << "|" <<  getPrintStringForDouble(entry.seqNumber)
                      << "|" << endl;
        }
    }
 std::cout << "+-----------------+-----------------+-----------------+-----------------"
 << "+-----------------+"
 << endl;
}
/*
 * Get the total number of entries present
 * inside the local characteristics table
 */
int CBRSingleton::getTotalAddressRoutes() {
    return localCharacteristicsTable_.size();
}
class CBR_Timer :  public cOwnedObject
{
  protected:
    CBRSingleton* ntwkController;
    mapCharacteristicEntries_* singleEntry;
  public:
    virtual void removeTimer();
    CBR_Timer(CBRSingleton* controller);
    CBR_Timer();
    ~CBR_Timer();
    virtual void expire() = 0;
    virtual void removeQueueTimer();
    virtual void resched(double time);
    virtual void setCharacteristicEntry(mapCharacteristicEntries_ *e) {singleEntry = e;}
};
/* Timer to manage individual characteristic entries inside the local table. */
class CBR_CharacteristicLinkTimer : public CBR_Timer
{
  public:
    CBR_CharacteristicLinkTimer(CBRSingleton* controller) : CBR_Timer(controller) {}
    CBR_CharacteristicLinkTimer(CBRSingleton* controller, mapCharacteristicEntries_* entry)
        : CBR_Timer(controller) { singleEntry = entry; }
    CBR_CharacteristicLinkTimer():CBR_Timer() {}
    void expire();
};
#endif
```

A.4 Messages

A.4.1 CbrPacket

```
cplusplus {{
    #include "SimpleAddress.h"
}}
class LAddress::L3Type extends void;
// Characteristic based routing network packet contains a
// destination and source network address.
// Additionally a time to live (ttl) field like hop count
// can be defined in order to limit the
// maximum number of hops the messge will travel. The
// sequence number is guranteed to be unique for all packets
// generated by one host
//
// @author Suhaib Naseem
packet CbrPkt
{
```

```
        LAddress::L3Type destAddr; // destination address
        LAddress::L3Type srcAddr; // source address
        int hopCount = 0; // hop count
        unsigned long seqNum = 0; // sequence number
        // packet delaing with characteristic by name
        string characteristic = "";
        // characteristic weight on propagation
        double weightOnPropagate = 0;
        // characteristic weight
        double weight = 0;
        string data; // route data
}
packet CbrReqPkt extends CbrPkt
{
        LAddress::L3Type reqSrcAddr; // request source address
        // transaction number of the request message
        long transactionId = 0;
        simtime_t messageTime;
}
packet CbrRepPkt extends CbrPkt
{
        LAddress::L3Type reqSrcAddr; // request source address
        // transaction number of the reply message
        long transactionId = 0;
        string payload; // characteristic source reply data
        simtime_t messageTime;
}
packet CbrTracePkt extends CbrPkt
{
        LAddress::L3Type reqSrcAddr; // request source address
        // transaction number of the request message
        long transactionId = 0;
        simtime_t origTime;
        simtime_t messageTime;
}
```

A.5 Configuration

A.5.1 omnet.ini

```
[General]
network = CharacteristicRoutingNetwork
cmdenv-express-mode = true
record-eventlog = false
sim-time-limit = 5000s
tkenv-default-config =
cmdenv-event-banners = true
cmdenv-module-messages = true
**.vector-recording = true
**.result-recording-modes = all
##########################################################################
#       Parameters for the entire simulation                             #
##########################################################################
*.playgroundSizeX = 850m
*.playgroundSizeY = 850m
*.playgroundSizeZ = 100m
CharacteristicRoutingNetwork.numHosts = 16
# uncomment to enable debug messages for all modules
# **.debug = 0
**.coreDebug = false
**.debug = true
##########################################################
#                       WorldUtility parameters          #
##########################################################
**.world.useTorus = false
##########################################################################
#       Parameters for the ConnectionManager                             #
##########################################################################
**.connectionManager.carrierFrequency = 2.4e+9Hz  # [Hz]
# max transmission power [mW]
**.connectionManager.pMax = 110.11mW  # [mW]
# signal attenuation threshold [dBm]
**.connectionManager.sat = -120dBm  # [dBm]
# path loss coefficient alpha
**.connectionManager.alpha = 4
**.connectionManager.sendDirect = false
##########################################################################
#       Parameters for the Application Layer                             #
##########################################################################
# debug switch
**.appl.headerLength = 1024bit
**.appl.burstSize = 3
##########################################################################
#       Parameters for the Network Layer                                 #
##########################################################################
**.netwl.headerLength = 32bit# in bits
**.netwl.stats = false
##########################################################################
#       Parameters for the Mac Layer                                     #
##########################################################################
```

```
# debug switch
**.mac.headerLength = 272 bit
**.mac.queueLength = 14
**.mac.bitrate = 2E+6bps# in bits/second
**.mac.autoBitrate = false
### values if no fading is modelled, gives at most 1% packet error rate
**.mac.snr2Mbit = 1.46dB  # [dB]
**.mac.snr5Mbit = 2.6dB  # [dB]
**.mac.snr11Mbit = 5.68dB  # [dB]
**.mac.rtsCtsThreshold = 400
**.mac.neighborhoodCacheSize = 30
**.mac.neighborhoodCacheMaxAge = 100s  # [s]
**.mac.txPower = 110.11mW  # [mW]
#############################################################################
#       Parameters for the Phy                                              #
#############################################################################
**.phy.usePropagationDelay = false
**.phy.thermalNoise = -110dBm  # [dBm]
**.phy.analogueModels = xmldoc("config.xml")
**.phy.decider = xmldoc("config.xml")
**.phy.sensitivity = -119.5dBm  # [dBm]
**.phy.maxTXPower = 110.11mW
**.phy.initialRadioState = 0
**.phy.useThermalNoise = true
################ NETW layer parameters ####################
*.host[*].netwl.maxTtl = 3
*.host[*].netwl.boredTime = 0.5
*.host[*].netwl.numHosts = 16
################ APPL layer parameters ####################
#*.host[*].applicationType = "TestApplication"
**.appl.trafficParam = ${traffic = 1..19 step 2}s
**.appl.nbPackets = 1000
**.appl.initializationTime = 10s
CharacteristicRoutingNetwork.host[0].netwl.nodeCharacteristics = "AAAABBAA:5000"
CharacteristicRoutingNetwork.host[1].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[2].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[3].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[4].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[5].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[6].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[7].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[8].netwl.nodeCharacteristics = "AAAABBBB:6000"
CharacteristicRoutingNetwork.host[9].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[10].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[11].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[12].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[13].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[14].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[15].netwl.nodeCharacteristics = "AAAA:0 BBBB:0"
CharacteristicRoutingNetwork.host[0].appl.flood = false
CharacteristicRoutingNetwork.host[1].appl.flood = true
CharacteristicRoutingNetwork.host[2].appl.flood = true
CharacteristicRoutingNetwork.host[3].appl.flood = true
CharacteristicRoutingNetwork.host[4].appl.flood = true
CharacteristicRoutingNetwork.host[5].appl.flood = true
CharacteristicRoutingNetwork.host[6].appl.flood = true
CharacteristicRoutingNetwork.host[7].appl.flood = true
CharacteristicRoutingNetwork.host[8].appl.flood = false
CharacteristicRoutingNetwork.host[9].appl.flood = true
CharacteristicRoutingNetwork.host[10].appl.flood = true
CharacteristicRoutingNetwork.host[11].appl.flood = true
CharacteristicRoutingNetwork.host[12].appl.flood = true
CharacteristicRoutingNetwork.host[13].appl.flood = true
CharacteristicRoutingNetwork.host[14].appl.flood = true
CharacteristicRoutingNetwork.host[15].appl.flood = true
################ Mobility parameters #####################
**.host[*].mobility.initFromDisplayString = false
CharacteristicRoutingNetwork.host[*].mobilityType = "ConstSpeedMobility"
CharacteristicRoutingNetwork.host[*].mobility.debug = false
CharacteristicRoutingNetwork.host[*].mobility.speed = 10.0mps
CharacteristicRoutingNetwork.host[*].mobility.updateInterval = 3.0s
*.host[0..15].mobility.constraintAreaMinX = 0m
*.host[0..15].mobility.constraintAreaMinY = 0m
*.host[0..15].mobility.constraintAreaMinZ = 0m
*.host[0..15].mobility.constraintAreaMaxX = 550m
*.host[0..15].mobility.constraintAreaMaxY = 600m
*.host[0..15].mobility.constraintAreaMaxZ = 100m
*.host[0].mobility.initialX = 50m
*.host[0].mobility.initialY = 100m
*.host[0].mobility.initialZ = 100m
*.host[1].mobility.initialX = 150m
*.host[1].mobility.initialY = 100m
*.host[1].mobility.initialZ = 100m
*.host[2].mobility.initialX = 250m
*.host[2].mobility.initialY = 100m
*.host[2].mobility.initialZ = 100m
*.host[3].mobility.initialX = 350m
*.host[3].mobility.initialY = 100m
*.host[3].mobility.initialZ = 100m
*.host[4].mobility.initialX = 50m
*.host[4].mobility.initialY = 200m
```

```
*.host[4].mobility.initialZ = 100m
*.host[5].mobility.initialX = 150m
*.host[5].mobility.initialY = 200m
*.host[5].mobility.initialZ = 100m
*.host[6].mobility.initialX = 250m
*.host[6].mobility.initialY = 200m
*.host[6].mobility.initialZ = 100m
*.host[7].mobility.initialX = 350m
*.host[7].mobility.initialY = 200m
*.host[7].mobility.initialZ = 100m
*.host[8].mobility.initialX = 50m
*.host[8].mobility.initialY = 300m
*.host[8].mobility.initialZ = 100m
*.host[9].mobility.initialX = 150m
*.host[9].mobility.initialY = 300m
*.host[9].mobility.initialZ = 100m
*.host[10].mobility.initialX = 250m
*.host[10].mobility.initialY = 300m
*.host[10].mobility.initialZ = 100m
*.host[11].mobility.initialX = 350m
*.host[11].mobility.initialY = 300m
*.host[11].mobility.initialZ = 100m
*.host[12].mobility.initialX = 50m
*.host[12].mobility.initialY = 400m
*.host[12].mobility.initialZ = 100m
*.host[13].mobility.initialX = 150m
*.host[13].mobility.initialY = 400m
*.host[13].mobility.initialZ = 100m
*.host[14].mobility.initialX = 250m
*.host[14].mobility.initialY = 400m
*.host[14].mobility.initialZ = 100m
*.host[15].mobility.initialX = 350m
*.host[15].mobility.initialY = 400m
*.host[15].mobility.initialZ = 100m
*.host[0].appl.nodeAddr = 1
*.host[0].appl.dstAddr = 10
*.host[0].appl.flood = false
*.host[0].appl.payloadSize = 100byte
*.host[0].appl.headerLength = 128b
*.host[1].appl.nodeAddr = 2
*.host[1].appl.dstAddr = 10
*.host[1].appl.flood = false
*.host[1].appl.payloadSize = 100byte
*.host[1].appl.headerLength = 128b
*.host[2].appl.nodeAddr = 3
*.host[2].appl.dstAddr = 10
*.host[2].appl.flood = false
*.host[2].appl.payloadSize = 100byte
*.host[2].appl.headerLength = 128b
*.host[3].appl.nodeAddr = 4
*.host[3].appl.dstAddr = 10
*.host[3].appl.flood = false
*.host[3].appl.payloadSize = 100byte
*.host[3].appl.headerLength = 128b
*.host[4].appl.nodeAddr = 5
*.host[4].appl.dstAddr = 10
*.host[4].appl.flood = false
*.host[4].appl.payloadSize = 100byte
*.host[4].appl.headerLength = 128b
*.host[5].appl.nodeAddr = 6
*.host[5].appl.dstAddr = 10
*.host[5].appl.flood = false
*.host[5].appl.payloadSize = 100byte
*.host[5].appl.headerLength = 128b
*.host[6].appl.nodeAddr = 7
*.host[6].appl.dstAddr = 10
*.host[6].appl.flood = false
*.host[6].appl.payloadSize = 100byte
*.host[6].appl.headerLength = 128b
*.host[7].appl.nodeAddr = 8
*.host[7].appl.dstAddr = 10
*.host[7].appl.flood = false
*.host[7].appl.payloadSize = 100byte
*.host[7].appl.headerLength = 128b
*.host[8].appl.nodeAddr = 9
*.host[8].appl.dstAddr = 10
*.host[8].appl.flood = false
*.host[8].appl.payloadSize = 100byte
*.host[8].appl.headerLength = 128b
*.host[9].appl.nodeAddr = 10
*.host[9].appl.dstAddr = 10
*.host[9].appl.flood = false
*.host[9].appl.payloadSize = 100byte
*.host[9].appl.headerLength = 128b
*.host[10].appl.nodeAddr = 11
*.host[10].appl.dstAddr = 10
*.host[10].appl.flood = false
*.host[10].appl.payloadSize = 100byte
```

```
*.host[10].appl.headerLength = 128b
*.host[11].appl.nodeAddr = 12
*.host[11].appl.dstAddr = 10
*.host[11].appl.flood = false
*.host[11].appl.payloadSize = 100byte
*.host[11].appl.headerLength = 128b
*.host[12].appl.nodeAddr = 13
*.host[12].appl.dstAddr = 10
*.host[12].appl.flood = false
*.host[12].appl.payloadSize = 100byte
*.host[12].appl.headerLength = 128b
*.host[13].appl.nodeAddr = 14
*.host[13].appl.dstAddr = 10
*.host[13].appl.flood = false
*.host[13].appl.payloadSize = 100byte
*.host[13].appl.headerLength = 128b
*.host[14].appl.nodeAddr = 15
*.host[14].appl.dstAddr = 10
*.host[14].appl.flood = false
*.host[14].appl.payloadSize = 100byte
*.host[14].appl.headerLength = 128b
*.host[15].appl.nodeAddr = 16
*.host[15].appl.dstAddr = 10
*.host[15].appl.flood = false
*.host[15].appl.payloadSize = 100byte
*.host[15].appl.headerLength = 128b
##########################################################################
#       Parameters for the Host                                          #
##########################################################################
*.host[0].netwl.na = 1
*.host[0].nic.id = 10001
*.host[1].netwl.na = 2
*.host[1].nic.id = 10002
*.host[2].netwl.na = 3
*.host[2].nic.id = 10003
*.host[3].netwl.na = 4
*.host[3].nic.id = 10004
*.host[4].netwl.na = 5
*.host[4].nic.id = 10005
*.host[5].netwl.na = 6
*.host[5].nic.id = 10006
*.host[6].netwl.na = 7
*.host[6].nic.id = 10007
*.host[7].netwl.na = 8
*.host[7].nic.id = 10008
*.host[8].netwl.na = 9
*.host[8].nic.id = 10009
*.host[9].netwl.na = 10
*.host[9].nic.id = 10010
*.host[10].netwl.na = 11
*.host[10].nic.id = 10011
*.host[11].netwl.na = 12
*.host[11].nic.id = 10012
*.host[12].netwl.na = 13
*.host[12].nic.id = 10013
*.host[13].netwl.na = 14
*.host[13].nic.id = 10014
*.host[14].netwl.na = 15
*.host[14].nic.id = 10015
*.host[15].netwl.na = 16
*.host[15].nic.id = 10016
*.host[*].netwl.na = 0
*.host[*].nic.id = 0
```

A.5.2 CbrApplLayer.ned

```
import org.mixim.base.modules.IBaseApplLayer;
simple CbrApplLayer like IBaseApplLayer
{
        parameters:
                bool   debug = default(false); // debug switch
                bool   stats = default(true);  // stats switch
                bool   trace = default(true);  // trace switch
                // mean time between packets (poisson arrival rate)
                double trafficParam @unit(s);
                double nodeAddr; // node address
                double dstAddr;        // packet destination node address
                bool flood; // send packets continuously
                double payloadSize @unit(byte); // number of bytes per packet
                double nbPackets; // number of packets to generate
                // length of the application message header (in bits)
                int headerLength @unit("bit");
        gates:
                // to receive data from communications stack
                input lowerLayerIn;
                // to send data to communications stack
                output lowerLayerOut;
                // to receive control messages from communications stack
                input lowerControlIn;
                // to send  control messages from communications stack
```

```
        output lowerControlOut;
}
```

A.5.3 CbrNetwLayer.ned

```
import org.mixim.base.modules.BaseNetwLayer;
simple CbrNetwLayer extends BaseNetwLayer
{
    parameters:
        @class(CbrNetwLayer);
        headerLength = 12 bit;
        double na;
        double maxTtl;
        double boredTime;
        int numHosts;
        string nodeCharacteristics = default("AAAA:2000 AABB:1000");
}
```

A.5.4 CharacteristicRoutingNetwork.ned

```
import org.mixim.base.modules.BaseNetwork;
import org.mixim.modules.node.Host80211;
network CharacteristicRoutingNetwork extends BaseNetwork
{
    parameters:
        int numHosts; // total number of hosts in the network
    submodules:
        host[numHosts]: Host80211 {
            @display("p=148,94;b=42,42,rect,yellow;i=device/wifilaptop");
        }
    connections allowunconnected:
}
```

A.5.5 Host80211.ned

```
module Host80211 extends WirelessNode
{
    parameters:
        applicationType = "org.tcd.WSNCBRRouting.CbrApplLayer";
        networkType = "org.tcd.WSNCBRRouting.CbrNetwLayer";
        nicType = "Nic80211";
}
```

A.5.6 config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
        <AnalogueModels>
                <AnalogueModel type="SimplePathlossModel">
                    <parameter name="alpha" type="double" value="4.0"/>
                    <parameter name="carrierFrequency" type="double" value="2.412e+9"/>
            </AnalogueModel>
        </AnalogueModels>
        <Decider type="Decider80211">
                <!-- SNR threshold [NOT dB]-->
                <parameter name="threshold" type="double" value="0.12589254117942"/>
        </Decider>
</root>
```

Bibliography

[1] Vincent D. Park , Joseph P. Macker, "Anycast Routing for Mobile Services, 1999"

[2] Chalermek Intanagonwiwat, "Directed diffusion: a scalable and robust communication paradigm for sensor networks"

[3] Jabed Faruque, Konstantinos Psounis, and Ahmed Helmy, "Analysis of Gradient-based Routing Protocols in Sensor Networks"

[4] Gary Zhong, Songwu Lu and Lixia Zhang, "GRAdient Broadcast: A Robust Data Delivery Protocol for Large ScaleSensor Networks"

[5] Shunliang Mei, Youzheng Wang, Jing Wang, "Stable enhancement for AODV routing protocol"

[6] Guoxian Yang, Stefan Weber, "Uisce: Characteristic-based routing in mobile ad hoc networks"

[7] Anindya Basu, Alvin Lin, Sharad Ramanathan, "Routing using potentials: a dynamic traffic-aware routing algorithm"

[8] Olga Ratsimor, Dipanjan Chakraborty, Anupam Joshi, Timothy Finin, "Alliance-based Service Discovery for Ad-Hoc Environments"

[9] Chakraborty D., Joshi A., Yesha, Y, Finin T., "GSD: a novel group-based service discovery protocol for MANETS"

[10] Helal S., Desai, N., Verma V., Choonhwa Lee, "Konark - a service discovery and delivery protocol for ad-hoc networks"

[11] Feng Zhu, Mutka, M., Ni L, "Splendor: A secure, private, and location-aware service discovery protocol supporting mobile services"

[12] Klein M., Konig-Ries B., Obreiter P, "Service rings - a semantic overlay for service discovery in ad hoc networks"

[13] Michael Klein , Birgitta Knig-ries , Philipp Obreiter, "Lanes - A Lightweight Overlay for Service Discovery in Mobile Ad Hoc Networks, 2003"

[14] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, Tim Finin, "Towards Distributed Service Discovery in Pervasive Computing Environments"

[15] 'Mesh Topology'. Pace University. Available at http://webpage.pace.edu/ms16182p/networking/mesh.html

[16] 'Location-based MANETs/VANETs'. Available at http://www.ics.uci.edu/ keldefra/manet.htm

[17] 'A wireless protocol for vehicular communications'. Broadband Wireless Access (BWA) Lab. Available at http://www.ms-aloha.eu/

[18] Yang Guoxian, "Anonymous routing Based on Characteristics protocol", MSC Thesis, Faculty of Computer Science, Trinity, 2007

[19] 'OMNET++ v4.3 Simulator Development Environment'. Available at http://www.omnetpp.org/omnetpp

[20] 'MiXiM v2.1 Simulator for MANETs using the OMNeT++ simulation engine.'. Available at http://sourceforge.net/projects/mixim/

[21] Murthy C. Siva Ram, Manoj B.S., "Ad Hoc Wireless Networks. Prentice Hall Communications Engineering and Emerging Technologies Series, 2004", pages 306-450.

[22] Ricardo Simon C., "Unstructured Decentralised Data Distribution in Wireless Sensor Networks", PHD Thesis, Faculty of Computer Science, Trinity, 2012

[23] 'OMNET++ user's guide'. Available at http://www.omnetpp.org/doc/omnetpp/UserGuide.pdf

[24] 'OMNET++ user manual'. Available at http://www.omnetpp.org/doc/omnetpp/Manual.pdf

[25] Antonio Carzaniga, Matthew J. Rutherford, and Alexander L. Wolf, "A Routing Scheme for Content-Based Networking"

[26] Mian, A.N. ; Univ. of Rome La Sapienza, Rome ; Baldoni, R. ; Beraldi, R., "A Survey of Service Discovery Protocols in Multihop Mobile Ad Hoc Networks"