Ketan Devadiga

Development of a Multicast Routing Protocol for Low power and Lossy Networks

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.
Espoo 16.12.2013

Thesis supervisor:

Prof. Antti Ylä-Jääski

Thesis advisors:

Zhonghong Ou (Post Doc Researcher)

Yang Deng (Doctoral Student)

Aalto University
School of Science

AALTO UNIVERSITY
SCHOOL OF SCIENCE

ABSTRACT OF THE
MASTER'S THESIS

Author: Ketan Devadiga

Title: Development of a Multicast Routing Protocol for Low power and Lossy Networks

Date: 16.12.2013 Language: English Number of pages:7+58

Department of Computer Science and Engineering

Professorship: Data Communications Code: T-110

Supervisor: Prof. Antti Ylä-Jääski

Advisors: Zhonghong Ou (Post Doc Researcher), Yang Deng (Doctoral Student)

The Internet of things (IoT) is a new paradigm that has been gaining popularity in recent years. As the name "Internet of things" suggests, things surrounding us will be able to interact with each other and also connect to the Internet, thus forming a worldwide network of connected objects. The number of potential applications of this concept is huge and includes various domains such as home environment, transportation, healthcare and so on. To enable the Internet of things, different technologies and standards have been proposed. Among them, the IP for Smart Objects (IPSO) alliance promotes the use of Internet Protocol (IP) as the network technology for IoT. The Internet Engineering Task Force (IETF), as part of its IoT related activities, has been working on using IPv6 to connect devices in low power wireless personal area networks (LoWPANs).

The devices operating in LoWPANs are constrained on resources such as memory, processing power and sometimes energy (in case, they are operating on battery). Hence protocols designed for such networks have to consider the limitations of the devices. There has been considerable research done to design protocols that enable and support IPv6 in LoWPANs. However, there is not much effort in the area of multicast communication. There are various scenarios where efficient multicast communication would be beneficial. For example, consider a group of lights in a room that can be controlled by an actuator. In such scenarios, well designed multicast protocols would be useful in saving resources of the nodes. In this thesis, we design and implement a multicast routing protocol for low power and lossy networks. The protocol is implemented on Contiki OS, an operating system developed for the Internet of things. In addition, we test this protocol using Cooja, a cross-layer simulator developed for Contiki OS.

Keywords: multicast, Internet of Things, Contiki, 6LoWPAN

Acknowledgements

This thesis was carried out at the Data Communications Software Research Group at Aalto University School of Science. The thesis was funded by the MAMMoTH (Massive Scale Machine-to-Machine Service) project.

First of all, I would like to thank Professor Antti Ylä-Jääski and my instructors Zhonghong Ou and Yang Deng for providing me an opportunity to work in this project. I am thankful to them for their guidance and advice during the course of this thesis work . I would also like to thank my colleagues in the Data Communications Software Lab. I have received many interesting suggestions from them that has been useful for my research work.

Finally, I would like to express my gratitude to family and friends for their endless support during my study years.

Otaniemi, 16.12.2013 Ketan Devadiga

Contents

Abbreviations and Acronyms

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ACK	Acknowledgement
ACL	Access Control List
CBT	Core-based Trees
CC	Consistency Check
CoAP	Constrained Application Protocol
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DAD	Duplicate Address Detection
DAO	Destination Advertisement Object
DIO	Destination Information Object
DIS	Destination Information Solicitation
DLL	Data Link Layer
DODAG	Destination Oriented Directed Acyclic Graph
DR	Designated Router
DVMRP	Distance Vector Multicast Routing Protocol
IANA	Internet Assigned Numbers Authority
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IoT	Internet of things
IP	Internet Protocol
IPv6	Internet Protocol Version 6
LLC	Logical Link Control
LLN	Low power and Lossy Networks
LoWPAN	Low power Wireless Personal Area Networks
LPP	Low Power Probing
LR-WPAN	Low rate Wireless Personal Area Network
MAC	Media Access Control
MLD	Multicast Listener Discovery
MOSPF	Multicast Open Shortest Path First
MP2P	Multipoint to Point
MPL	Multicast Protocol for Low power and Lossy Networks
MTU	Maximum Transmission Unit
ND	Network Density
OS	Operating System
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
P2MP	Point to Multipoint
P2P	Point to Point
PIM	Protocol Independent Multicast
PIM-DM	Protocol Independent Multicast-Dense Mode
PIM-SM	Protocol Independent Multicast-Sparse Mode
PDU	Protocol Data Unit

RIP	Routing Information Protocol
RP	Rendezvous Point
RPL	IPv6 Routing Protocol for Low power and Lossy Networks
RPT	Rendezvous Point Tree
SMRF	Stateless Multicast Forwarding with RPL
SSM	Source Specific Multicast
SPT	Shortest Path Tree
TCP	Transmission Control Protocol
UDGM	Unit Disk Graph Medium
UDP	User Datagram Protocol
USB	Universal Serial Bus

1 Introduction

There is a lot of hype surrounding the Internet of things (IoT) idea. It is expected to have a high impact on everyday life[1]. In the future, we might see its effect in our homes, offices, industries and so on. However, there are several technological obstacles faced by IoT. Many organizations such as research institutes and industrial consortiums are working towards standardization of technologies for IoT. One such effort is to use Internet Protocol (IP) as the networking technology. The number of devices in IoT is expected to be in billions. Ericsson predicts that by 2020, around 50 billion devices would be connected to the Internet[2]. There will be the problem of addressing those billions of devices. The large address space of Internet Protocol Version 6 (IPv6) has been proposed as a solution to this problem. A working group of the Internet Engineering Task Force (IETF) have designed the IPv6 over Low power Wireless Personal Area Networks (6LoWPAN)[3] standard which enables low power wireless nodes to use IPv6. Another IETF working group has developed the IPv6 routing protocol for low power and lossy networks (RPL)[4]. Constrained application Protocol (CoAP)[5] is an application layer protocol developed for low power and lossy networks (LLNs) by IETF. Although there has been large research efforts on protocols and standards related to 6LoWPAN, the area of multicast communication has not attracted much attention.

As far as we know, the Multicast Protocol for Low power and Lossy Networks (MPL)[6] and the Stateless Multicast Forwarding with RPL (SMRF)[7] are the only multicast protocols developed specifically for LLNs. MPL uses the Trickle algorithm[8] to enable multicast communication in constrained networks. In case of MPL, there is no topology maintained nor any concept of multicast groups. Without any topology for data transmission, all multicast data packets are broadcast throughout the network which might not be energy efficient. There is also the possibility of packets arriving out of order. SMRF uses features of RPL to enable multicast communication in LLNs. However, it allows data transmission in just one direction. This might be useful in applications such as code dissemination, but will not meet the requirements of several other application scenarios that might benefit from multicast communication.

Several real-world sensor network applications require multicast communication. Data reporting, data monitoring, powering on or off a group of devices are some examples that can use multicast communication. The devices operating in LLNs are resource constrained. They have limited memory, processing power, energy and generally support low data transfer rates. An efficient multicast communication protocol will help in saving bandwidth and energy of the devices.

For normal networks, there has been various multicast solutions developed. The IP multicast is one standard that has been popular and commercially used. In this method of multicast communication, a group of receivers are considered as a multicast group and each group is represented by an IP address. The Internet Assigned Numbers Authority (IANA) has reserved a range of IP addresses that can only be used as multicast destination addresses. The receivers interested in listening to a group address use a group management protocol to report their intent

to a nearby router. The routers construct multicast distribution trees using Protocol Independent Multicast (PIM)[9]. PIM is the multicast routing protocol that runs on routers and uses information from unicast routing protocol to build multicast distribution trees. These distribution trees are used to transmit multicast data from the sources to the receivers.

The goal of this thesis is to design and implement a multicast routing protocol for 6LoWPANs. The protocol that we have designed is based on the Source Specific Multicast (SSM)[10] protocol. SSM is an extension to the PIM protocol. It uses a subset of the PIM solution and tries to solve some issues faced by PIM. We have modified the SSM protocol so that it is suitable for 6LoWPANs.

1.1 Scope of the thesis

The scope of the thesis is limited to the design, implementation and testing of the routing protocol. The group management protocol has not been implemented.

During the design and experiments, we have considered factors such as packet loss, energy consumption and latency. Security issues have not been considered. The protocol has been implemented on Contiki OS and uses RPL as the unicast routing protocol.

RPL constructs Destination Oriented Directed Acyclic Graph (DODAG) trees, using which the routing tables are maintained. There could be multiple DODAG trees in a constrained network. As of now, the multicast protocol is designed to send and receive traffic within a single DODAG tree. Traffic flow between multiple DODAG trees is not supported.

1.2 Structure of the thesis

The remainder of the thesis is structured as follows: Chapter 2 describes various technologies related to our work. In Chapter 3, we explain how we designed the protocol and its working. In Chapter 4, we describe the implementation details. In Chapter 5, we analyse the protocol based on our experiments. In Chapter 6 we discuss the future work required to improve this protocol and to have a complete multicast solution. In Chapter 7, we provide concluding remarks.

2 Background

In this section, we give a brief overview of some of the background topics related to this thesis. We briefly explain the following topics: IEEE 802.15.4, 6LoWPAN, RPL, Contiki OS and the PIM architecture.

2.1 IEEE 802.15.4

The IEEE 802.15.4 standard[11, 12] defines the physical and media access control (MAC) layers of low rate wireless personal area networks (LR-WPANs). This standard was developed considering characteristics of LR-WPANs such as low data rate, low power consumption etc. In order to suit these characteristics, the standard adopts techniques such as reduction in frequency and amount of data transfer, reduced frame overhead and strict power saving mechanisms.

For the physical layer, the standard uses the unlicensed 2.4 GHz band for worldwide operation and the 868/915 MHz bands for Europe and United States respectively. Additional bands were added in later revisions. The 868/915 MHz and other bands would help in case of interference from other technologies associated with the 2.4 GHz band. The 2.4 GHz bands provides a transmission rate of 250 Kb/s while the 868/915 MHz provides rate of 20 Kb/s and 40 Kb/s respectively. The 868/915 MHz together supports 11 channels while the 2.4 GHz supports 16 channels, hence providing a total of 27 channels across the three bands.

The data link layer (DLL) is divided into two sub layers: the MAC and the logical link control (LLC). The logical link control layer is defined in other standards and the IEEE 802.15.4 defines only the MAC sublayer. The network topology in LR-WPAN can either be a star topology or some sort of extended connected topology such as mesh, ring or cluster. In order to allow for these type of topologies, the MAC frame also called the MAC protocol data unit (PDU) is kept very flexible. There are four types of frames supported: data, beacon, acknowledgement and MAC command frames. The latter two frame types are used for MAC communication and only the data and beacon frames contain data from higher layers. The entire MAC PDU should not exceed a length of 127 bytes.

The standard supports two types of channel access mechanisms: nonbeacon-enabled and beacon-enabled mode. In case of beacon enabled mode, the data frames are sent using the slotted carrier sense multiple access with collision avoidance (CSMA/CA) method. In this method, the nodes are synchronized by the beacon frames sent by a special node called the coordinator. In nonbeacon-enabled network, the data frames are sent using the unslotted CSMA/CA method. In this case, whenever a collision is detected, the nodes backoff for a random time before retrying. Further, in both cases, for transmission of acknowledgement frames, no CSMA is used since they are transmitted immediately after the data frame.

The standard supports various security suites which can be broadly categorized into four types: no security, encryption only security, authentication only security and security with both encryption and authentication[13]. The radio chips have to implement access control lists (ACL) that contains information regarding which

security suite has to be used. However, radio chip designers do not have to support all the security suites. The standard mandates only no security and security with both authentication and encryption using 64 bytes be supported.

2.2 6LoWPAN

6LoWPAN[3, 14] is the standard protocol defined to enable IPv6 over the IEEE 802.15.4 standard. The IEEE 802.15.4 standard defines the media access control (MAC) layer and the physical layer of the Open Systems Interconnection (OSI) model for low-rate wireless personal area networks (LR-WPANs). The 6LoWPAN is an adaptation layer on the top of IEEE 802.15.4 layer so that IPv6 can be enabled on it. Figure 2.1 shows the OSI model along with the 6LoWPAN layer between the IP layer and the MAC layer.

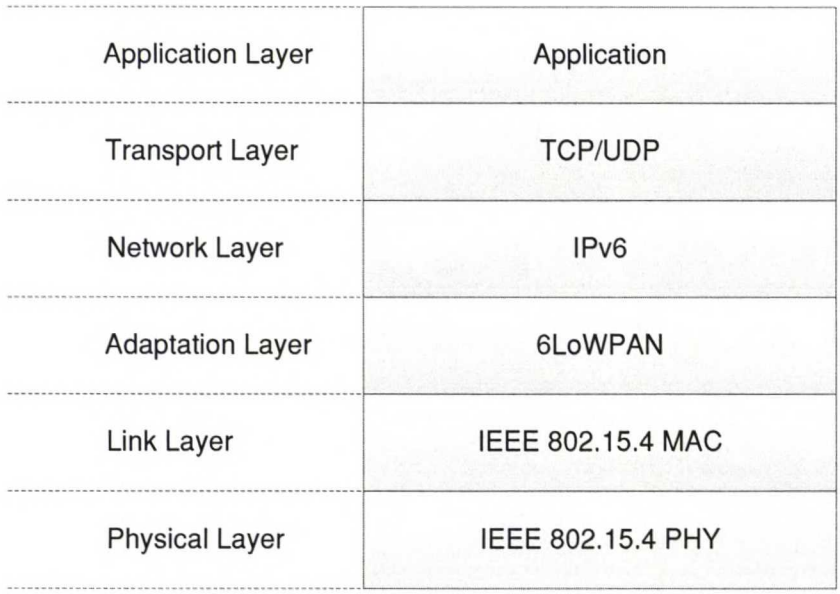

Figure 2.1: 6LoWPAN Network Stack

The IEEE 802.15.4 data frames carry IPv6 packets. The maximum transmission unit (MTU) size for IPv6 is 1280 octets. But, the IEEE 802.15.4 protocol data unit (PDU) size may vary depending on the frame header size. The maximum physical layer packet size in IEEE 802.15.4 is 127 octets. If we consider the MAC header of 25 bytes, then the MAC protocol data unit would have 102 bytes left. If link-layer security options are added to the MAC frame header, i.e 21 bytes in the worst case, it would leave 81 octets for the IPv6 packet. This size of the PDU for MAC frames is insufficient to support an MTU of 1280 bytes. In order to support

the MTU size required, the 6LoWPAN provides a fragmentation and reassembly mechanism. Moreover, with a PDU size of 81 octets left, the IPv6 header and an upper transport layer protocol like UDP would require 40 and 8 bytes respectively, thus leaving 33 bytes for application data. To improve the size left for application data, the 6LoWPAN layer provides header compression schemes.

As mentioned earlier, the IEEE 802.15.4 defines two modes of operation: nonbeacon-enabled and beacon-enabled mode. However, 6LoWPAN does not enforce any mode of operation and any of the two can be used.

The 6LoWPAN defines different types of encapsulation headers. These encapsulation headers are included in the payload of the IEEE 802.15.4 MAC protocol data unit. The IPv6 packet is then included as payload after these encapsulation headers. The following types of encapsulation headers are supported: dispatch headers, fragmentation headers and mesh headers. The dispatch header specifies the type of headers that follow. The fragmentation header is used for the fragmentation and reassembly of IPv6 packets. The mesh header is used in case of mesh-under type of routing. Two types of routing are supported in 6LoWPAN: mesh-under and route-over. In case of mesh-under, the routing decisions are made at the adaptation layer and in the latter case, the routing decisions are made at the network layer.

Various compression schemes have been defined in 6LoWPAN. These schemes are used to compress IPv6 and UDP headers. Initially, two schemes, LOWPAN_HC1 and LOWPAN_HC2 were supported. But these schemes were effective only in case of link-local unicast communication and not for global and multicast communication. Later the LOWPAN_IPHC and LOWPAN_NHC schemes were added. The LOWPAN_IPHC is used for encoding IPv6 headers and LOWPAN_NHC for next header compression such as UDP. In an IPv6 header, the following fields could be compressed: version number, traffic and flow label, source and destination addresses, hop limit and the payload length. The version number is always six; the traffic and flow label are zero; depending on the scenario, the source and destination IP addresses can be derived from the IEEE 802.15.4 addresses or link-local prefix or also from a prefix assigned to the whole 6LoWPAN. The hop limit can be assigned to a standard value by the sender while the payload length can be derived from the IEEE 802.15.4 header. The LOWPAN_NHC can be used to compress UDP. For UDP, the ports and the checksum value can be compressed.

2.3 RPL

RPL[15] is the routing protocol developed for routing IPv6 packets on low power and lossy networks. Unlike traditional routing protocols that use hop count as its routing metric, RPL uses an objective function as its routing metric. Depending on the network's objective function, nodes select their parents and optimize routes. Based on applications, different networks might have different objectives. Some of the parameters to decide on the objective are latency, energy etc. RPL supports three types of traffic: multipoint-to-point(MP2P), point-to-multipoint (P2MP) and point-to-point(P2P) traffic.

Unlike normal networks, constraint networks have to discover their peer nodes

and form network topologies by themselves. RPL provides this by organizing the topology as a Directed Acyclic Graph (DAG). In RPL, a DAG will have a sink or root node, thus forming a Destination Oriented Directed Acyclic Graph (DODAG). Each DODAG structure will have a unique root called the DODAG root. Some terminologies with respect to RPL are explained below.

An RPL instance can be a collection of one or more DODAGs. The DODAGs can be connected to each other through their respective DODAG roots via an LLN link or a normal link that need not be constrained in nature. The DODAGs can be disjoint too. For example, a single DODAG with a single root can be an RPL instance such as a control point in a home automation system. A number of disjoint DODAGs with their own DODAG roots and sending data to a monitoring system can also be an RPL instance. Each RPL instance is identified using an RPLInstanceID which is also associated with each RPL packet in the instance. Each DODAG is identified using a DODAGID. A DODAG is also associated with a DODAGVersionNumber. The DODAGVersionNumber changes whenever a DODAG tree is reconstructed. A rank number associated with each node indicates the nodes relative position with respect to the DODAG root. The scope of rank number is the DODAGVersionNumber. Figure 2.2 shows an RPLInstance comprising of two different DODAGs having their own DODAG root. We can see that each DODAG tree has its own DODAGID and DODAGVersionNumber. In Figure 2.3, since one of the nodes has moved, the DODAG with DODAGID ID2 has reconstructed itself and increased its DODAGVersionNumber, but the other DODAG has not changed its DODAGVersionNumber. We can also observe the rank numbers associated with each node and that it has changed in the second figure since the node moved. The selection of rank numbers is dependent on the objective function of the RPL network.

RPL control messages are used to construct upward and downward routes in a DODAG tree. To construct upward routes Destination Information Object (DIO) messages are used. The upward route formation takes the following steps.

- A node which is preconfigured as the DODAG root advertises its presence, a DODAGID and other necessary metrics to all its link local neighbours. These metrics are used by the nodes to select their parents.
- Nodes listen to DIO messages either to join a new DODAG by selecting new parents or to maintain their DODAG tree. The selection of parents is done based on their optimization objective function.
- Some nodes that are configured as routers update their routing tables with information in the DIO message and advertise their own DIO messages to its link-local neighbours. A leaf node will just select a route and does not advertise the DIO information.

To construct downward routes, RPL uses the Destination Advertisement Object (DAO) messages. Downward routes are required for P2MP and P2P traffic. There are two modes supported for downward routes: storing and non-storing. In case of non-storing mode, the packet travels all the way to the root node before travelling downward towards its receiver node. The DODAG root sends packets to its

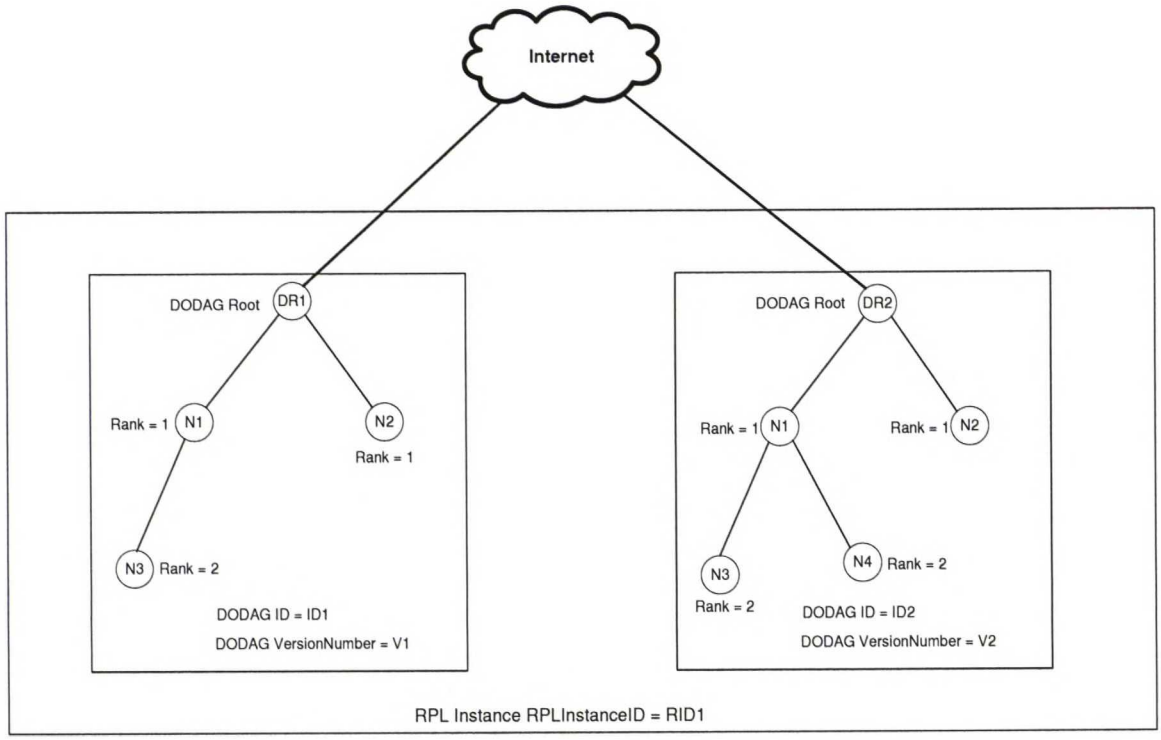

Figure 2.2: An RPL instance, DODAG trees and nodes

destination using source routing. In storing node, the packet travels just till the common node of the sender and receiver, then starts travelling downwards towards the receiver. All the non-leaf nodes act as routers in this mode and have to store routing information. The downward routes formation takes place in the following way.

- The nodes send DAO messages towards the root node via their parent nodes.
- In case of storing mode, the intermediate router nodes store the routing information and forward the DAO message towards the root. In non-storing mode, the intermediate nodes do not store any routing information and just forward the messages towards the root nodes via its parents.
- In storing mode, each router including the root has a list of nodes that it has routes to. In case of non-storing mode, the root has all source routing information obtained via the DAO messages.

Other than DAO and DIO messages, there are three other types of RPL control messages: Destination Information Solicitation (DIS) message, DAO-ACK message and Consistency Check (CC) message. The DIS message is used to solicit DIO message from a particular node. The DAO-ACK messages is issued as a response

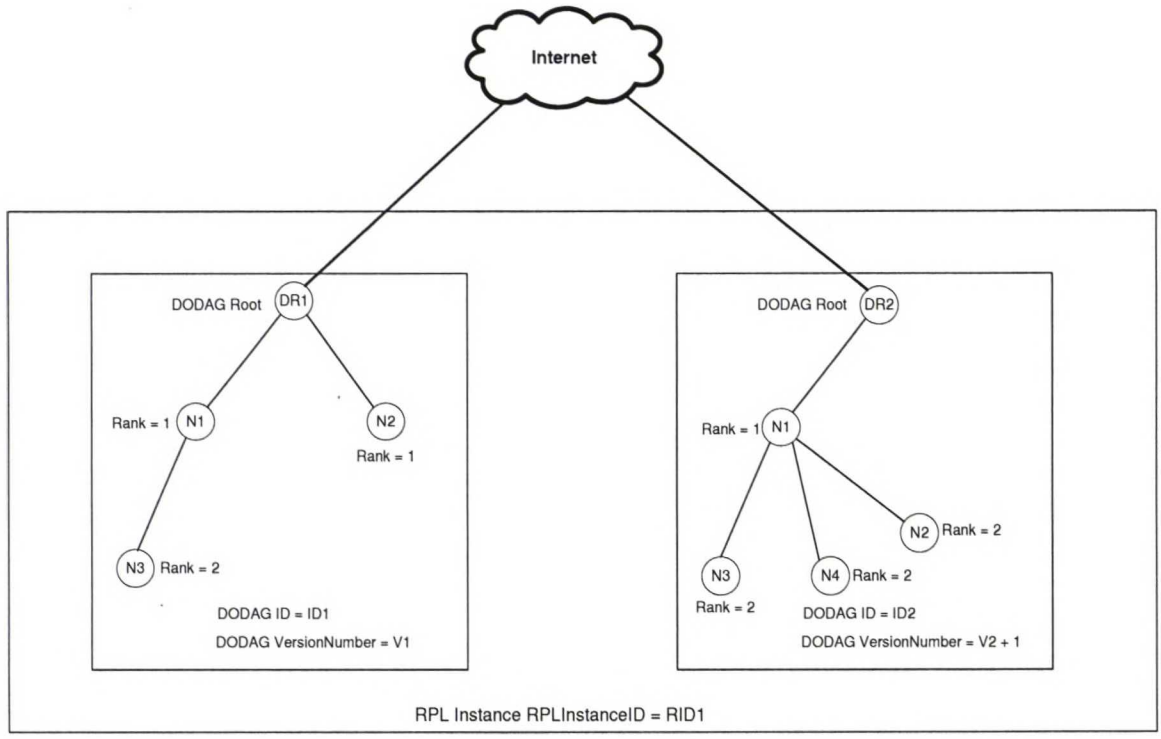

Figure 2.3: The RPL network after one of the DODAG tree is updated

to unicast DAO messages. All the explained RPL control messages can be sent in a secure manner using security mechanisms which will be explained later. The task of securing message counters and sending challenge/response is accomplished using the CC control message.

It is possible that loops are formed due to various reasons such as loss in control packets. Loop avoidance is implemented in RPL using RPL packet information that is transferred in data packets. Using the IPv6 Hop-by-Hop option header, it is possible to send RPL information such as rank error which can be used for loop detection.

RPL also provides security mechanisms and supports message confidentiality and integrity. Although it is designed to use link-layer mechanisms for security, RPL has its own security support in case link-layer does not provide any security. There are three security models: unsecured, pre-installed and authenticated. In the "unsecured" model, as the name specifies, RPL does not provide security rather link-layer or application layer security mechanisms can be used. In the "pre-installed" model, all the nodes joining a particular RPL instance would have pre-installed keys which would help them to send RPL messages securely. In the "authenticated" model, the leaf nodes can use pre-installed keys to join the RPL instance while the router nodes have to acquire keys from an authentication authority and use those

keys to join the RPL instance.

2.4 Contiki OS

Contiki[16] is a lightweight operating system developed for low power constrained devices by the Swedish Institute of Computer Science (SICS). Contiki supports dynamic loading and unloading of individual programs and services. The contiki kernel is event driven. However, contiki provides support for pre-emptive multi-threading through a library. This library can be linked by services that require multi-threading support. The codebase for contiki is open source and written in C language. There is support for various microcontroller architectures such as Atmel AVR, MSP430 etc.

A contiki system consists of the following components: the kernel, libraries, the program loader and a number of processes. A process can either be a service or an application. The functionality of a service can be used by many processes. For example, the communication stack and sensor device drivers are services. The system is divided into two parts: the core and the loaded programs. Figure 2.4 gives an overview of the contiki system. As you can see from the figure, the core consists of the kernel, the program loader, some features of the language run-time and the communication stack. The stack also includes device drivers for the network hardware. The entire core is a single binary image and stored in devices before deployment and is generally not modified, although there is a possibility to modify it later. The program loader is used to load programs into the device, and they can use the communication stack or a directly attached storage such as EEPROM.

In the following subsections, we discuss various topics related to contiki OS.

2.4.1 Kernel architecture

Basically, a contiki kernel is a lightweight event scheduler that performs two tasks:

- dispatch events to running processes
- regularly call the poll handlers implemented by processes

There are two types of events supported: asynchronous events where the event is dispatched to the process after some delay, and synchronous events where the event is immediately dispatched to the process. Once the event has been dispatched to a process, the respective event handler in the process cannot be pre-empted and has to run to completion. However, there is a mechanism provided using a library wherein the event handler internally can allow pre-emption .

Other than dispatched events, the kernel also uses polling. Periodically the kernel calls all poll handlers that are implemented by the processes. The polling mechanism is generally used by processes to listen to hardware data.

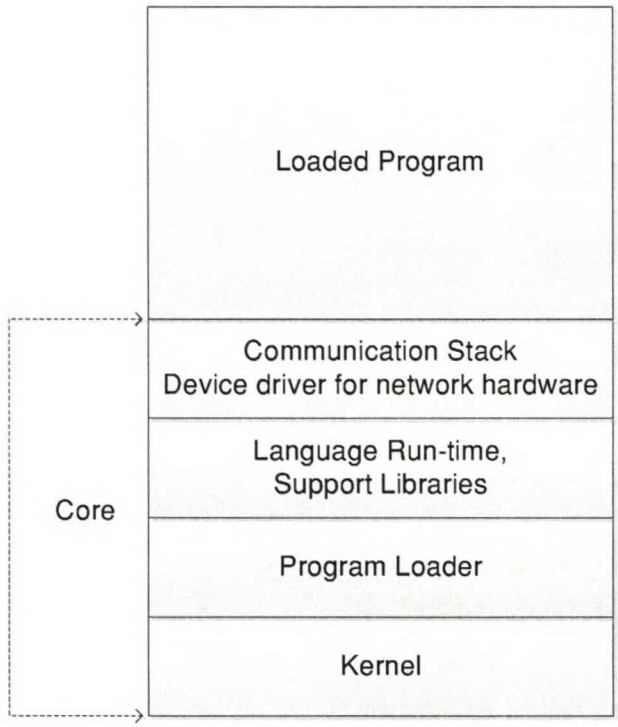

Figure 2.4: The Contiki System

2.4.2 Protothreads

Protothreads[17] are a programming abstraction to make event-driven programming simple. Basically, it allows you to program thread-like behaviour without actually using threads. Unlike normal threads, protothreads do not have a stack of their own and all protothreads use the same stack. This makes it lightweight which is suitable in case of memory constrained embedded devices. In contiki, C language constructs are used to implement protothreads. The kernel is a lightweight event scheduler and processes are implemented as protothreads. Whenever an event is posted to a process, the protothread in the process gets executed.

Whenever the event handler of the process is executed, its protothread is invoked. The protothread block is placed between the statements: PT_BEGIN and PT_END. Any code placed in between these statements are part of the protothread. The statement PT_WAIT_UNTIL in a protothread makes the protothread block conditionally until a specific event occurs, which provides the thread-like behaviour. Contiki also provides a PT_YIELD statement which blocks the protothread unconditionally.

The protothread implementation in contiki poses some restrictions on usage of

certain C constructs. In contiki, protothreads are implemented using switch constructs. Hence, while developing contiki applications, switch statements cannot be used inside protothreads. Since automatic variables with local-scope are placed in the stack, they will not be saved when there are block statements in a protothread. However, if you need to save variables across blocking wait statements, static local variables can be used.

2.4.3 uIP stack and IPv6 support

Contiki OS has support for IPv4 and IPv6 through the uIP and the uIPv6 implementations[18, 19, 20]. Both the implementation are tightly coupled with UDP and TCP protocol implementations and have the same application interface for developers. The uIP and uIPv6 stacks are designed to have a minimal set of features to support a full TCP/IP network stack so that it fits the memory restrictions of a low power constrained device. They adhere to RFC requirements but have removed many unnecessary features to reduce code size. Some of the features that are not supported are as follows: buffered packet for retransmission, sliding window, multiple interfaces and out-of-sequence TCP data.

In order to reduce memory usage, a single global buffer is used to handle packets which can hold a single packet of maximum packet size. The device driver puts an incoming packet in this buffer and notifies uIP which in turn notifies the application that is responsible for the packet. The application has to act on the packet or copy it to other buffer so that it frees the space for further incoming packets. In case a packet arrives while processing data, then the device driver or network device has to store the packet, or else the packet would be dropped. The same global buffer is used for sending packets too. While sending packet, an application sends the pointer to the data along with the length of data to uIP. uIP then adds relevant headers to the global buffer and sends it out the network along with the application data.

The uIP runs as a protothread in contiki. The main control loop in a uIP stack does the following two tasks repeatedly:

- check for an incoming packet
- check if a periodic timer has expired

Whenever, there is an incoming packet, the input handler of uIP is called. The input handler returns immediately and the application that is responsible for the packet processes the data. The periodic timer is used to fire the periodic timer handler. This handler is used to perform TCP tasks such as retransmissions which are related to delay.

Since our work deals mainly with IPv6, we will discuss some details on how uIPv6 is implemented in contiki. The uIPv6 implementation in contiki is Phase 1 compliant according to IPv6 certification tests conducted in 2008 and has received IPv6 Ready Silver Logo. The following features are fully supported: IPv6 RFC 2460[21], IPv6 addressing architecture (RFC 3513)[22], Neighbor discovery (RFC 4861)[23], ICMPv6 (RFC 4443)[24], Default address selection (RFC 3484)[25] and

Address Autoconfiguration(RFC 4862, but without MLD support)[26]. For uIPv6, the single global buffer would be of the length of MAC header plus 1280 which is the minimum link MTU as specified for IPv6. However, we have observed that on certain devices due to memory constraints, this is set to a much lower value. In addition to the global buffer, there are other buffers that support fragmentation and reassembly and queuing of packets.

When the IPv6 process starts, it assigns an IPv6 address combining the prefix fe80 :: 0 \64 and its MAC address as specified in RFC 4862. It also performs Duplicate address detection (DAD). Simultaneously, the neighbour discovery process is also started. Router solicitation messages are sent to receive information from neighbours which is used to set its global address and other network variables. For neighbour discovery, a neighbour cache, a prefix list and a default router list is used. In addition, an interface address list is used to store the various addresses assigned to the node. These data structures are kept updated due to events that are triggered during transmission and reception of data packets.

2.4.4 Cooja simulator

Cooja[27] is a cross-layer simulator developed for contiki OS. Generally, simulators are used to simulate nodes at any one particular layer, either at the hardware level or at the higher levels such as network layer. But the cooja simulator supports simulation at different layers and also cross-simulation between these layers. The cooja simulator is written in Java programming language. It can be easily extended to support additional sensor hardware, radio mediums etc. It provides a graphical user interface to perform various tasks such as creating new simulations, adding new nodes etc. The simulation can be saved in a configuration file. This configuration file can be modified to make changes to the simulation. There are also various plugins available that helps you in simulation. Moreover, it is also possible to extend cooja by writing new plugins.

While developing an application on contiki, we can code the contiki application in C, test the code on cooja and then use the same code to run the application on a real hardware. The application can be compiled in two ways. We can compile it for the native machine on which we are running cooja or we can compile it for the sensor hardware in which case cooja simulates the hardware for the sensor node. It is also possible to simulate nodes in cooja that are not contiki based. These kind of nodes will not take much resources as in simulated hardware and hence can be used to test higher level functionalities and protocols such as distributed algorithms. However, for our work in order to test the protocol, we have written code in C, compiled it for specific hardware and used the cooja simulator for testing. If we have to test on a real hardware, the same code can be compiled and run on the hardware device.

2.5 IP multicast and PIM

IP multicast is a widely used standard for multicast transmission in wide area networks. A multicast architecture needs to implement a routing protocol and a group

membership protocol[28]. The routing protocol is used by routers to construct multicast paths and to forward multicast data whereas the group membership protocol is used by the routers to be aware about hosts listening to multicast groups in its subnetwork. In case of IPv4, Internet Group Management Protocol (IGMP) and in case of IPv6, Multicast Listener Discovery Protocol (MLD) are used as group membership protocols. However, for routing, there were various protocols developed. For the initial IP multicast architecture[29], multicast extensions were developed for existing routing protocols such as Open Shortest Path First (OSPF)[30] and Routing Information Protocol (RIP)[31]. For example, Distance Vector Multicast Routing Protocol (DVMRP)[32] was an extension to RIP while Multicast Open Shortest Path First (MOSPF)[33] was an extension to OSPF . However, these suffered from scalability issues. The Core Based Tree protocol (CBT)[34] was developed to address this problem. Although it solved some problems, it had issues with delay. For each multicast group, CBT uses a single shared tree rooted at a central router called the core router. This resulted in traffic concentration at the core which might cause latency in case of high data rate applications. In order to solve these problems, the Protocol Independent Multicast (PIM)[9] was proposed and now has become the de facto standard for multicast routing.

In the following subsections, we will briefly describe Protocol Independent Multicast -Sparse mode (PIM-SM)[35], Source specific multicast (SSM)[10, 36] and about group membership protocols.

2.5.1 PIM-SM

PIM, as the name suggests, does not depend on any particular unicast routing protocol. It uses the routing information from the underlying unicast routing protocol to build multicast trees. The PIM protocol has two modes: Protocol Independent Multicast -Dense mode (PIM-DM) and Protocol Independent Multicast -Sparse mode (PIM-SM). PIM-DM is similar to DVMRP, i.e it floods the multicast data. The routers that do not need the multicast data send Prune messages towards the source. The router receiving a Prune message stops forwarding the data towards that interface, hence pruning that branch of the multicast tree. However, PIM-DM introduces the concept of protocol independence whereas DVMRP is dependent on RIP. PIM-DM is suitable for networks where the multicast receivers are densely spread throughout the network. But when the receivers are not dense, this would result in wastage of bandwidth because of flooding. PIM-SM is designed for multicast groups that are sparsely spread across the network. In this section, we briefly describe the working of PIM-SM.

In PIM-SM, routers that are interested in a particular multicast group, explicitly send Join messages in order to join the multicast tree for that group. There exists a Rendezvous Point (RP) router which is a meeting point between receivers and senders. This tree which includes an RP is called a shared tree or the RP tree (RPT). In addition, if the data rate is high, the protocol provides the routers the option to establish a shortest path tree (SPT) from the receiver to a particular source. When an SPT is formed, the router ceases to be a part of the shared tree. To be a part

of a multicast tree, the router has to store the state for that particular tree. The state contains the following information: source (S), multicast group address (G), incoming interface and outgoing interface. In an SPT, the state is denoted by the tuple (S,G). In an RPT, the router has to forward data coming from the RP router, which might originally be coming from any source, and hence the forwarding state is denoted by (*,G).

The PIM-SM protocol consist of three phases: the RP tree, the Register stop and the Shortest-Path tree phase.

In the first phase, a shared tree is formed between the RP router and the interested receivers. The following steps occur during the first phase.

- The hosts interested in listening to a group G communicates its interest to the nearest router on its subnet known as the Designated router(DR). It can use a group membership protocol such as IGMP or MLD.
- Once it receives an interest from one of its hosts, the DR sends an explicit Join message towards the RP router for that group. The Join message is called the (*,G) Join.
- As the Join message travels towards the RP, all the intermediate routers store the state (*,G) which indicates that they have to forward all data intended for the group G. This Join message can either reach all the way till the RP or till a router which already has the (*,G) stored.
- Eventually, the RP would have received several (*,G) Joins from various DRs and hence a data distribution tree from the RP to the receivers is established. As mentioned earlier, this tree is called the shared tree or RPT.
- The RP router can receive data from any source for this group. When a source intends to send data to a group G, the DR of the source encapsulates the data in a unicast message and sends it towards the RP. This process of sending encapsulated packets is called Registering and the messages are called PIM Register packets. The RP, on receiving the message, decapsulates it and sends it on the shared tree, thereby reaching the receivers.

In addition to these steps, it is necessary to maintain the shared tree. In order to do that, the DRs keep resending the Join messages towards RP. If a DR does not have any interested listeners for a group G, then it sends a (*,G) Prune message towards the RP, which deletes that part of the tree which is not needed.

The encapsulation-decapsulation method might be resource intensive operations for the routers. Hence in the second phase, a source-specific tree is formed between the senders and the RP. This takes place using the following steps.

- When the RP receives the first encapsulated data packet from a source S, it sends a source specific (S,G) Join towards the source S.
- As the (S,G) Join reaches hop-by-hop towards the DR of the source S, the intermediate routers store the state (S,G). This join can either reach all the

way till S or till a router which already has the state (S,G). It may also reach a router which has the state (*,G), in which case the data coming from S to G can take that short-cut instead of going all the way till RP.

- At this stage, a source specific tree is now established between RP and S. This (S,G) state in routers is used to forward all data that comes from S to the group G.
- The RP starts getting two copies of the data packets from S, one from the source specific tree and the other through encapsulated packets. When it starts receiving two copies of the same data packet, it sends a Register-Stop message towards S. The DR of S, then stops encapsulating the multicast data and hence the RP starts getting just one copy, i.e through the source specific tree.

At this stage, the traffic flows through the source specific tree from S to RP, and then through the shared tree from RP to receivers. However, there might be issues with latency since the data has to be sent all the way to the RP before reaching the receivers. To overcome this problem, the DR of the receiver has the option to construct a source specific shortest path tree (SPT). This takes place in the third phase in the following way.

- The DR of the receiver, when it decides to form the SPT, sends a source specific (S,G) Join towards S.
- Intermediate routers store the state (S,G). The (S,G) Join can reach all the way till S or till a router which already has the (S,G) state.
- After the (S,G) Join has reached S, an SPT is now established between S and the DR of the receiver. The data from S, will travel through this SPT to reach this receiver.
- However, the DR has not stopped receiving data from RP through the shared tree. When it receives two copies of the data packet, it sends an (S,G) Prune message for that RPT to the RP router. The intermediate routers when it receives this Prune message, will stop forwarding data from S to the group G.

At this final stage, an optimum shortest path tree is now established between S and this receiver. However, data from S would still be travelling to RP of that group, but would not be forwarded through the shared tree to this particular receiver.

2.5.2 SSM

Source-Specific multicast (SSM) is a multicast architecture which uses the Shortest-Path tree phase of PIM-SM. SSM addresses the following issues with PIM-SM: address allocation, access control and complexity associated with well known senders. In SSM, a multicast tree is associated with the group (S,G) which means that the data forwarded on this particular tree would be coming from the source S to the

destination group G. The group (S,G) is commonly called a channel. This concept of an SPT per channel would solve some issues with PIM-SM. It avoids the problem of collision of destination group addresses. For instance, if there are two senders S1 and S2 and they both want to send data on the group address G, SSM would still work since the channel (S1,G) is different from the channel (S2,G). Unlike PIM-SM which has no access control and listeners receive data from all sources sent to a group address, SSM receivers specifically subscribe to a particular source to receive data. In case of PIM-SM, even if there are well known source addresses, the protocol still has to go through the whole process of setting up RPT, Register-Stop etc. But in SSM, if there are well known senders, a receiver directly subscribes to the channel thus avoiding the complexity. However, the concept of channel also means that this model can only be used for one-to-many multicast communication.

As mentioned earlier, SSM uses the third phase of PIM-SM for its routing decisions. A receiver in order to subscribe to a channel and receive data would go through the following steps:

- A host interested in listening to a channel would communicate this interest to its DR using a group membership protocol. The group membership protocol has to support the host in specifying both the source address and the group destination address to its DR.
- The DR sends an (S,G) Join message towards the source S.
- Intermediate routers store the state (S,G) as the Join message travels towards the source S. Once the Join message reaches the DR of the source S, the SPT for that channel would be established.
- The group address G has to be in the range allocated for SSM by the Internet Assigned Numbers Authority (IANA). When a source sends data on the group G, the intermediate routers would first check if the address G is in the SSM allocated range. If it is in the specified range, the routers look at the source address and forward the data only if they are allowed to forward for that channel.
- Similar to PIM-SM, the DR can use the Prune message to unsubscribe from a particular channel.

Our multicast solution is based on the SSM architecture. We have used the concept of (S,G) channels and modified it so that it can be used for 6LoWPANs. The design of our multicast solution will be discussed later.

2.5.3 Group Membership Protocols

Group membership protocol enable hosts to report group memberships to their connected routers. IGMP and MLD are the commonly used group membership protocols for IP multicast. As discussed earlier, IGMP is used for IPv4 and MLD is used for IPv6 enabled hosts and routers. The IGMP version 1[28] was the first

standardized version. This version had issues with delays in hosts quitting a group membership which was rectified in IGMPv2[37]. In IGMPv3[38], the hosts were provided the option to filter sources. The hosts could specify a source along with the group address, so that they receive messages only from that source. The other option is to filter out a source i.e when they specify a source along with the group address, they receive messages from all sources except that particular source.

MLDv1[39] is developed based on IGMPv2 and MLDv2[40] based on IGMPv3. Both provide similar features as their respective IPv4 versions but for IPv6. With IPv4, IGMP uses the IP protocol number 2 while MLD is a sub-protocol of the Internet Control Message Protocol version 6 (ICMPv6) and it uses the IP protocol number 58.

IGMPv3 and MLDv2, both provide mechanisms to filter sources. Hence, it is used as the group membership protocols in case of SSM[41]. Moreover, both IGMPv3 and MLDv2 are both compatible with earlier versions, which means they can be used for other multicast protocols as well. For this thesis, we have not developed any group membership protocol. In order to implement a complete multicast solution, it is required to develop a group membership protocol. Since our multicast routing protocol is based on SSM, the group membership protocol can be derived based on MLDv2.

3 Design

In this section, we explain how our protocol works. We also explain the motivation that led us to this design and the objectives we followed while designing the protocol.

3.1 Motivation

There can be various applications of multicast communication in LLNs. In any LoWPAN, the nodes would be sensors, actuators or routers. If we assume a LoWPAN covering a floor in a building management system, then an actuator might be used to turn on all the lights on a floor or in a particular room. There might be monitoring nodes that request data from several temperature sensors in the building. These are some of the scenarios in which multicast communication would be useful. We have tried to design our multicast routing protocol so that it can meet the needs of such applications. Since 6LoWPAN uses IP technology, we decided to explore existing multicast protocols in wide area networks. In this section, we will briefly explain why we chose the SSM protocol as basis for our design.

The PIM architecture, which is part of the IP multicast suite of protocols, is most widely used. PIM solves many issues associated with previous multicast protocols. Since PIM is not dependent on any particular unicast protocol, it proved to be a good candidate to be examined for use in 6LoWPANs. As discussed in Section 2.5, PIM has two versions: PIM-DM and PIM-SM. In case of PIM-DM, the receivers are densely spread. Initially, It broadcasts the multicast data across the network and later routers which are not interested in the data stops forwarding the data. This approach would not be suitable in LLNs as data would be flooded throughout the network. Since the devices in LLNs are resource constrained, flooding the network might not be a good idea.

PIM-SM is suitable when receivers are sparsely spread across the network. As discussed earlier, in PIM-SM, there are multicast trees established and multicast data traverse through these trees to reach the receivers. It would be suitable to try to adapt PIM-SM to be used in 6LoWPANs. PIM-SM has the concept of rendezvous point, a router which is the meeting point between senders and receivers. In RPL, there is the concept of DODAG root, which seemed suitable to take the role of rendezvous point of PIM-SM. However, as we explored further, the PIM-SM protocol turned out to be too complex to be implemented in 6LoWPANs. It consists of three phases starting with the construction of shared tree and ending with the construction of the shortest path tree. Designing these three phases to be used in 6LoWPANs would not be suitable for devices with resource constraints. However, one option could have been to use just the shared tree and use DODAG root as the rendezvous point. But this would cause all multicast data to traverse via the DODAG root which would result in unnecessary flow of multicast data all the way to the DODAG root before reaching the receivers.

However, using the shortest path tree would be more efficient in 6LoWPANs. The SSM protocol uses just the shortest path tree and hence was a better option to be adapted to 6LoWPANs. The shortest path tree of SSM creates an efficient tree

between the source and the receivers where the flow of data is minimum. As a result, there is no unnecessary flow of data in parts of the network that do not require multicast data. Moreover, the receivers subscribe to a specific source to receive multicast data. This feature is useful in certain application scenarios mentioned earlier.

3.2 Objectives

As mentioned before, the devices in low power, low data rate networks have several constraints and hence the protocols for them has to designed accordingly. Here, we briefly describe the objectives we had in mind while designing the protocol.

1. Power consumption

   In order to last for a longer time, the sensor devices in LLNs need to have efficient energy saving mechanisms. Most of the energy in a sensor device is used by transceivers i.e energy is spent most in sending and receiving data. While designing the protocol, we have tried to reduce the amount of time spent by the transceivers in sending data so that we reduce the energy consumed by the device.

2. Memory restrictions

   Constrained devices will have memory restrictions. For example, a Zolertia Z1 mote[42] comes with a second generation MSP430F2617 microcontroller that has 8KB RAM and 92KB flash memory[43]. It is necessary that we take into account memory restrictions while designing protocols for LLNs. The multicast routing protocol would consist of routing tables which should be designed not to consume too much memory.

3. Scalability

   As the number of nodes in a network increases, the protocol should not degrade its performance. If we consider multicast in particular, then the increase in number of listeners to a multicast group should not decrease the overall performance of the protocol.

4. Network repairs

   In a 6LoWPAN network, the nodes might fail due to various reasons causing disruptions in the network. RPL provides repair mechanisms to overcome these disruptions. The protocol that we developed is dependent on RPL as its unicast routing protocol. Hence it has to be designed to be aware of the repairs carried out by RPL and accordingly adjust its routing mechanisms.

3.3 Protocol Overview

We have already discussed that our multicast routing protocol is based on the Source Specific Multicast (SSM) protocol. The SSM protocol has to be modified so that it

works on 6LoWPANs. Normal wired networks are different from 6LoWPANs. We list a few differences between them, especially focusing on protocols and functionalities relevant for our design.

- As discussed before in Section 2.2 the MAC layer is the IEEE 802.15.4 standard. The 6LoWPAN layer acts as an adaptation layer between the MAC and the IPv6 layer. At the network layer, the most commonly used routing protocol is RPL. For our protocol design, we assume RPL as the unicast routing protocol. The TCP/UDP layer sits on top of IPv6. We have also discussed in Section 2.4 that uIPv6 is implemented as the IPv6 layer in contiki OS.

- In wired networks, a router will have many interfaces and hence interfaces have to be a part of the protocol design. However, in wireless networks and hence in 6LoWPAN, a router node will have just one radio interface.

- As described in Section 2.3 , the RPL use DODAG trees to maintain routes. Each DODAG tree will have a DODAG root. Each node in the tree will have a parent via which it is connected to the DODAG root and to other nodes. The nodes in the tree can either be routers that have routing functionalities or leaf nodes which do not have routing functionalities.

- When a wireless node broadcasts any message, all nodes in its communication range will receive the message, unlike wired networks, where the router has to broadcast messages through all its interfaces.

- The IEEE 802.15.4 standard does not support multicast but does support broadcast frames. Hence at the MAC layer, there is no capability for a wireless node to receive multicast packets.

In this section, we will give an overview of our protocol design. We will describe how we have based our design on SSM and the modifications and additions that we have made so that the protocol can be used in 6LoWPANs.

Similar to SSM, our protocol will use states in routers to indicate that they are a part of the multicast tree. In SSM, the state was the tuple (S,G) where S was the address of the sender sending data on the destination group address G. As in the case of SSM, the group address G has to be an address in the range allocated for SSM by IANA. The tuple (S,G) was called a channel. However, for 6LoWPAN we have modified the state to the tuple (S,G,P,N). The additional variable P is the next-hop router address of the node on its path to S i.e it would be the parent of the node in the multicast tree. Note that P is the parent address in the multicast tree which need not be the same as its parent in the DODAG tree. The next variable in the state, N, indicates the number of child nodes a router node has on the multicast tree. This means that a router node will keep track of how many join requests it receives for a particular channel but not the address of its children. In our protocol, we have used broadcast messages for data transmission. Since broadcast messages are received by all nodes in the radio range, the parent address P in the state helps a node to receive it from its intended parent only. The variable N helps a node to

prune its part of the tree if there are no longer any children listening to a channel. We will explain in detail as to how this occurs in the next section.

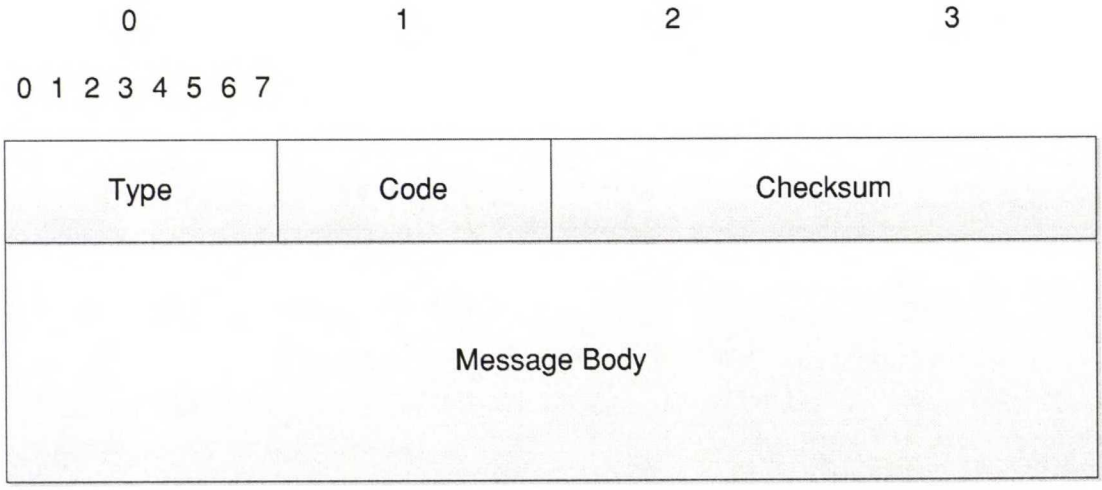

Figure 3.1: ICMPv6 Message Format

We have defined three types of control messages for our protocol: Join, Prune and Update messages. The Join message is used by routers when they want to join a channel and be a part of the multicast tree for that channel. The Prune message is used by routers when they no longer want to be part of the multicast tree for that channel. The Update messages are heartbeat messages which are sent by routers regularly to keep their part of the multicast tree alive. Similar to RPL control messages, the control messages in our protocol are Internet Control Message Protocol Version 6 (ICMPv6)[24] messages. The control message is an information message. It contains the ICMPv6 header followed by a message body. The Figure 3.1 shows an ICMPv6 message. For our experiments, we have chosen the message Type number 201. The Code field indicates the type of control message. The types are defined as shown in Table 3.1. The message body of the control message would contain information about the channel i.e the source of the channel S and the group address G.

Code	Type of Message
0x00	Join
0x01	Prune
0x02	Update

Table 3.1: ICMPv6 message codes

In an RPL DODAG tree, other than the root, the nodes could either be routers or leaf nodes. The leaf nodes could be compared to hosts in traditional wired networks. In addition to multicast routing protocol, a group membership protocol has to be designed which has not been covered in this thesis. The group membership protocol would be used by the leaf nodes to report their interest in a channel to its parent node in a DODAG tree. This parent node can be compared to the Designated router (DR) as described in SSM protocol. Hence, this node would be responsible for initializing the procedure in setting up its branch of the multicast tree for that channel. The mechanisms used for building multicast trees, pruning trees and data transmission is briefly described below.

- As mentioned earlier, the host node will use a group membership protocol to indicate its interest in joining a multicast channel to its parent node.
- Upon indication of interest in a channel from one of its child node, the parent node will issue a Join request for that channel towards the source of the channel. Meanwhile, it also stores the state for that channel in its multicast table.
- As the Join message traverses to the source of the channel, all the intermediate routers on its path will store the state for that channel in its multicast table before forwarding it. The state stored by the router nodes has been described earlier.
- If the Join message reaches a router that already has the state for that channel, then the router just updates the N variable in the state. The router does not forward the message further, thereby merging that branch of the tree into the already established tree for that channel.
- This Join message reaches the DR of the source. At that particular stage, the multicast tree for the channel is established. As new hosts join the channel, new branches for the multicast tree are created or added to the already existing tree.
- The procedure for pruning the tree is similar. A host indicates its interest to leave a tree to its parent node. The parent node then issues a Prune request towards the source of the channel.
- Whenever a Prune message is received by a router, it checks the number of subscribers to the channel. If the number of subscribers is just one, then it is safe to prune that part of the tree and forward the Prune message further. If there are more than one subscribers, then the router just updates its state, but keeps the tree intact by not forwarding the Prune message since there would be other subscribers still interested in the channel.
- The routers use broadcast to forward data from the source towards the leaf nodes of the multicast tree. The source of the channel broadcasts the data on the channel group address.

- When a router receives multicast data for a particular channel, the router forwards it by broadcasting it again if it satisfies two conditions: the router has to be a part of the multicast tree for that channel and the data has to be received from its parent on the multicast tree as specified by the P variable in the state. If these conditions fail, then the router drops the data packet.

In addition to these mechanisms, the protocol has to support rebuilding the multicast tree whenever there is a change in the DODAG tree due to RPL repairs. In low power and lossy networks, the nodes might undergo link degradation because of which some nodes might disappear. In such scenarios, RPL tries to rebuild the tree by initiating repairs. There are two kinds of repairs: global and local. In case of global repairs, the whole DODAG tree is updated, while with local repairs, the tree is updated in the affected area. These repairs might cause parent nodes to change, hence it is necessary that the multicast tree has to be rebuilt accordingly. In our protocol, the router which is a part of a multicast tree periodically checks for changes in its route to the source. If any change is detected, it initiates a new join towards the source via the new route. Further, it is necessary that stale branches of the multicast tree has to be pruned and overall, the multicast tree has to be kept alive. This is achieved by Update messages. Every router regularly sends Update messages per channel to its parent in the multicast tree. In this way, the parent node decides to keep its part of the multicast tree alive.

In the next section, we will explain the mechanisms in more detail with examples for each one of them.

3.4 Details

In this section we will discuss the following four mechanisms of the protocol in detail: join process, data transmission, prune process and the tree repair mechanism. For each process, we will explain it with a simple DODAG tree as example. In the diagrams, we have only shown the router nodes and the DODAG root. The leaf nodes are ignored since they are not part of the routing protocol. In a complete multicast solution, the leaf nodes would be communicating with the router nodes using the group membership protocol. Hence, in this section, whenever we mention a router node issuing a Join request, we assume that there are leaf nodes connected to the router which have already expressed interest in joining the channel. This means that the router is the Designated router for those leaf nodes. However, it is also possible that the router itself is interested in the channel.

3.4.1 Join Process

A router node surrounded by leaf nodes interested in a channel has to initiate the construction of multicast tree by issuing Join requests towards the source of the channel. If a multicast tree already exists, then the Join request might cause a new branch to be merged to the already existing tree. We explain this process with an example of a DODAG tree as shown in Figure 3.2. In the figure, DR is the DODAG root; the node N1 is the source of a channel sending data on the group G. Hence the

channel would be (N1,G). The nodes N3 and N4 are routers interested in joining the channel.

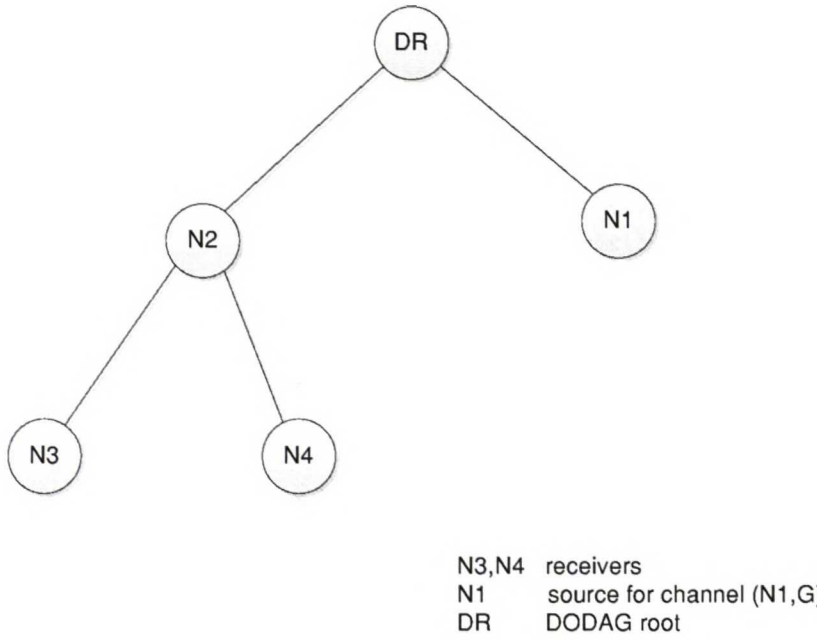

Figure 3.2: Join Process

Initially there would be no nodes listening to the channel i.e no multicast tree. The router node N3 interested in a channel issues a Join request. This is illustrated in Figure 3.3. The Join request is sent to the next-hop router towards N1 i.e the source of the channel. In this case, for N3, the next-hop router is N2. This routing information is obtained from the unicast routing table i.e RPL's routing table. In addition, N3 has to add the state into its multicast table so that it becomes a part of the multicast tree. N3 adds the state (N1, G, N2, 1). Here, we can see that the P variable is the node N2, i.e the next-hop router. The variable N is 1, since there are no other routers from which the node has received Join requests.

In Figure 3.4, the node N2 has received the Join message. It adds the state (N1, G, DR, 1) to its multicast table. Since DR is its next-hop router, it forwards the message to DR. We can also observe the dotted line between N3 and N2, which denotes the branch of the multicast tree that is established between N3 and N2 for the channel (N1,G).

Finally, the Join message reaches the source of the channel N1. At this stage, the multicast tree would have been fully established between the source N1 and the receiver N3. This is shown in Figure 3.5. The dotted lines denote the established multicast tree. The information stored in the multicast table for each node is also shown. From the figure, we can notice that the multicast tree has its root as the source of the channel and is built over the DODAG tree.

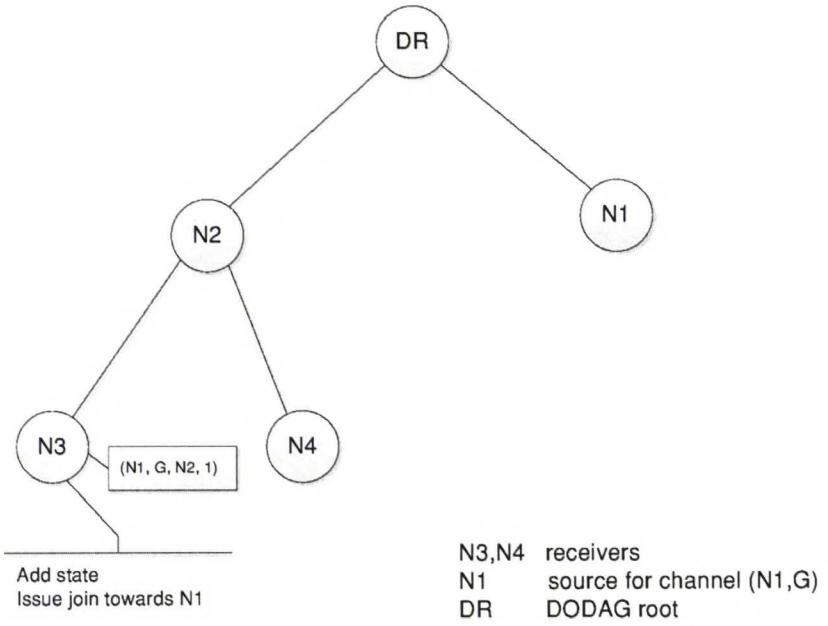

Figure 3.3: Join Process (contd..)

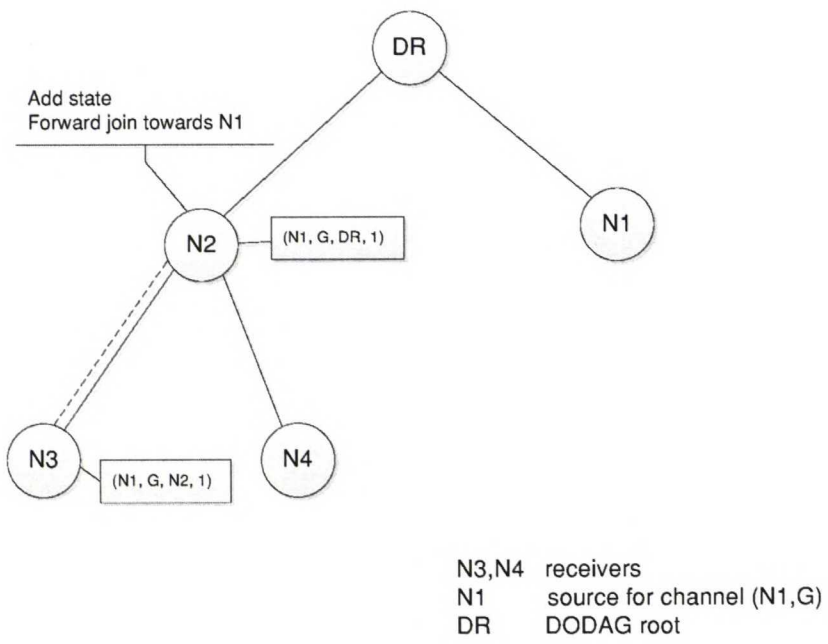

Figure 3.4: Join Process (contd..)

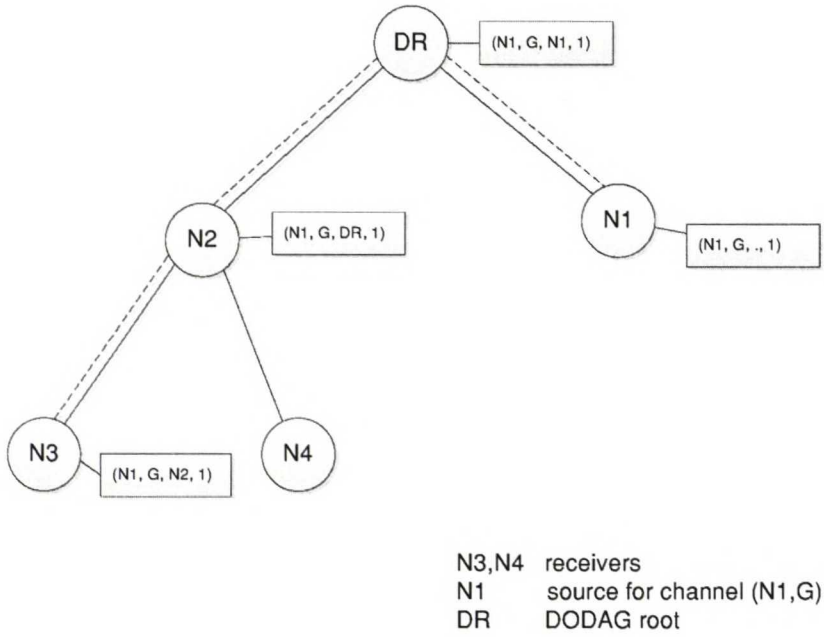

Figure 3.5: Join Process (contd..)

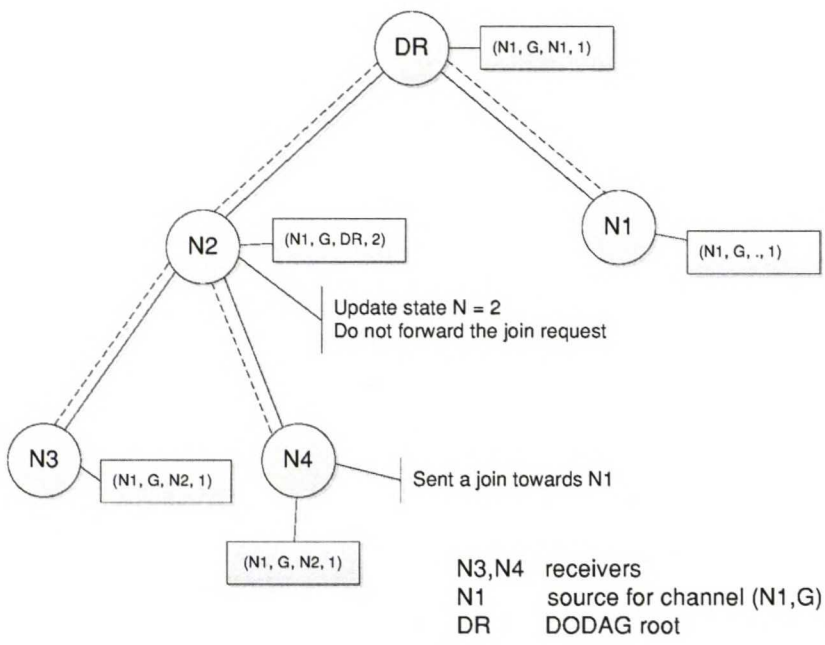

Figure 3.6: Join Process (contd..)

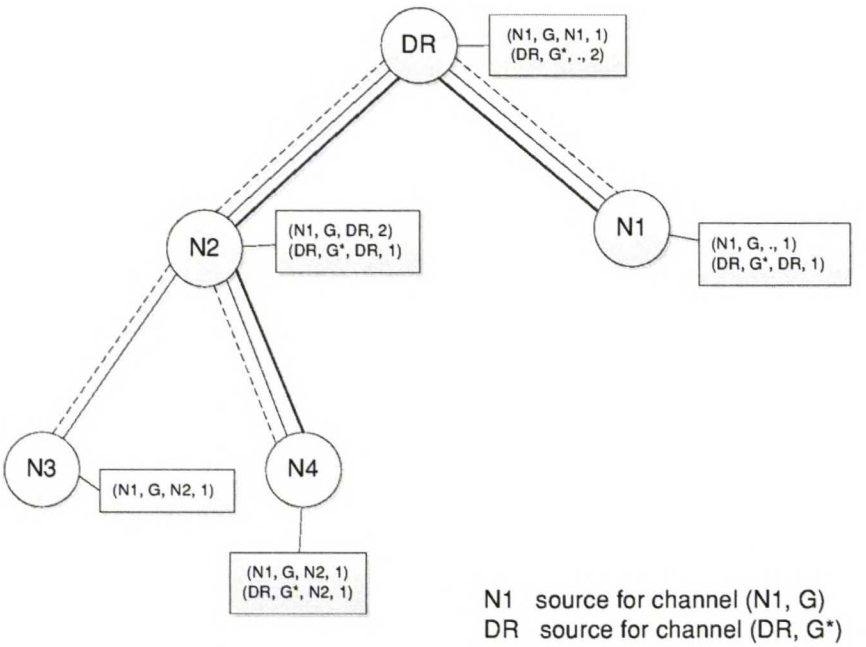

Figure 3.7: Join Process (contd..)

In Figure 3.6, the node N4 has sent a Join request to N2, since it has N2 as its next-hop router. It has also added the state to its multicast table. However, at N2, the behaviour is different. Since N2 already has the state in its table, it just increments the variable N in the state by 1 and does not forward the Join message further. In this way, that branch of the tree has been merged into the already established tree for the channel. In the same figure, we can see the fully established multicast tree including both the receivers N3 and N4.

Each channel will have its own multicast tree and hence, the router nodes will have a state per channel in its multicast table. Figure 3.7 shows a new multicast tree established for the channel (DR,G*). The nodes N1 and N4 are interested listeners. The bold lines denote the multicast tree for this channel with DR being the root of the multicast tree. We can also observe that at each node which are part of the new tree, there exists two states in the multicast table, one per channel.

3.4.2 Data Transmission

The protocol uses broadcast messages for data transmission. The source S of the channel (S,G) broadcasts data on the group address G. All the routers that are a part of the multicast tree broadcast the message again, so that eventually the message reaches all the receivers in the multicast tree. However, as explained earlier, the intermediate routers broadcast only if they have received the message from its intended parent in the multicast tree. To explain this process, we use one of the multicast tree example from the previous section. Figure 3.8 shows the multicast tree denoted by dotted lines with N1 as the source of the channel (N1,G). The circle around N1 indicates the range of data transmitted by the node. When N1 intends to send data on the channel, it broadcasts it. As shown in the figure, the node DR lies in the range of N1 and receives the data. When a node receives multicast data, lets say DR in this particular case, it checks if it is a part of the that multicast tree. The node DR has a state for the channel (N1,G), which matches with the packet's source and destination IP addresses. But the node has to also check if the packet has come from its immediate parent in the multicast tree. In this case, the P variable in the state is N1 matches the node from which the data has been received. The node does this check by comparing the packet's source MAC address from the MAC frame with the MAC address information stored in its neighbour discovery cache. Since the conditions are satisfied, the node DR now has to broadcast the data packet again so that it traverses further down the multicast tree.

In Figure 3.9, we can see that the node DR has broadcast the data packet. Both the nodes N1 and N2 lie in the range of DR and hence both of them receive the packet. The node N2 has a state for the channel (N1,G). Its immediate parent according to the state is DR, which is the node from which it received the packet. Hence it accepts and broadcasts it further. However, the node N1 rejects the packet. Though it has the state (N1,G) in its multicast table, it does not have an immediate parent in the state (denoted by a ".") and so it does not match with the data packet's source MAC address. Hence it rejects the packet and does not broadcast it again.

The Figure 3.10 shows the stage where the node N2 broadcasts the data. We

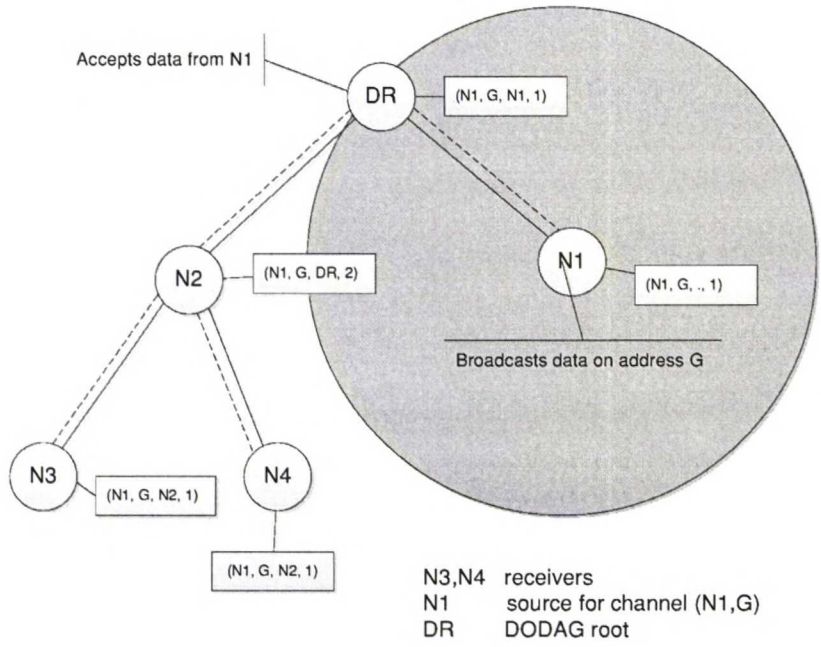

Figure 3.8: Data transmission

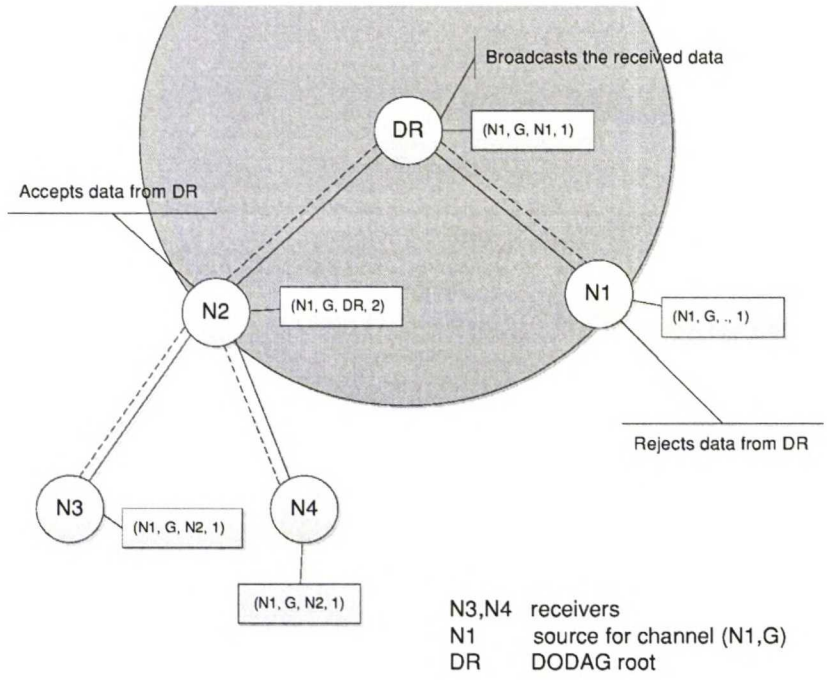

Figure 3.9: Data transmission (contd..)

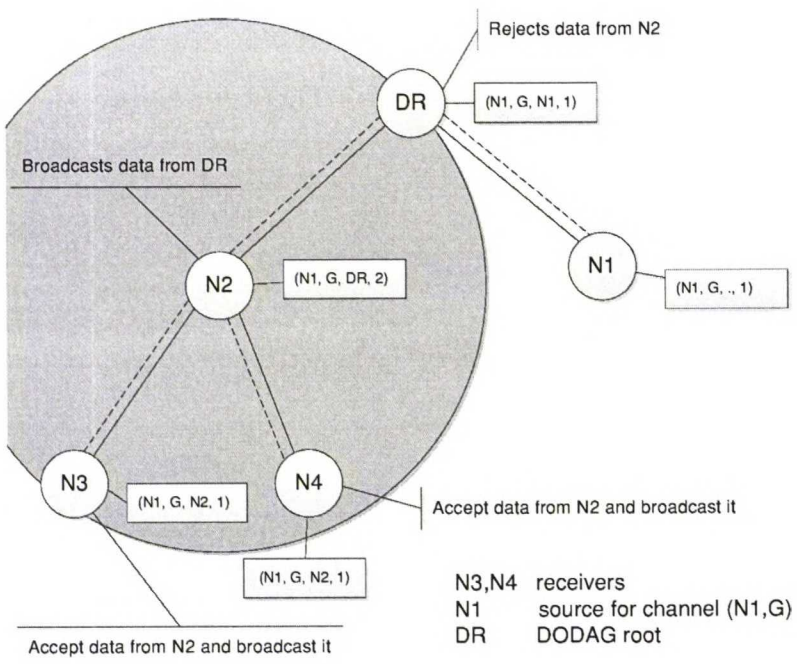

Figure 3.10: Data transmission (contd..)

can observe that the nodes N3, N4 and DR are in the range of node N2. The nodes N3 and N4 will accept the data and broadcast it again. Both of them are part of the multicast tree and their intended parent according to the state in the multicast table is N2, which is the node that it received the data from. The node DR rejects the data since its intended parent is N1 and not N2. At this stage, the nodes N3 and N4 have received the data and will broadcast it again. Any leaf nodes connected to the these nodes can receive the multicast data. However, the rules by which non-router nodes accept or reject data has to be covered by the group membership protocol and not the routing protocol.

3.4.3 Prune Process

The Prune process is used by routers to prune branches of multicast trees. If a router comes to know that they no longer have any leaf nodes listening to a channel, then that router can prune part of the multicast tree by sending a Prune message towards the source of the channel. As mentioned before, the router comes to know about leaf nodes and their interest in a channel using the group membership protocol. The Prune process can also be started in another scenario which will be explained in the next section. We will examine the Prune process using the same example from previous section. First, the node N4 will send a Prune message and then the node N3 will send a Prune message. In each step, we will see how the multicast tree gets pruned. In Figure 3.11, the node N4 will send a Prune message to its parent in the multicast tree. Since it no longer wants to be a part of the tree, it deletes the state (N1,G) from its multicast table. The node N2, upon receiving the Prune message decrements the N variable in its state by 1. The N variable denotes the number of Join requests it has received. In this case, the value of N is decremented to 1. This means that there are children in the multicast tree that are still interested. Hence N2 does not forward the Prune message further. From the figure, we can observe that the branch of the multicast tree between N2 and N4 has been pruned.

In Figure 3.12, the node N3 has sent a Prune message and also deleted the state (N1,G). However, at node N2, after decrementing N, the value of N is 0, which means there are no longer any children interested in the tree. Hence the node N2 deletes the state from its table and forwards the Prune message further towards the source of the channel. From the figure we can observe that both the branches of the tree from N3 and N4 to N2 have been pruned.

In Figure 3.13, we can see the whole multicast tree being pruned. The Prune message from N2 reaches DR. At DR, the value of N decrements to 0, and so the node deletes the state and forwards to N1. The node N1 also decrements the value of N to 0, and since there are no more children, deletes the state of the channel from its multicast table.

3.4.4 Tree repairs

The RPL protocol conducts repair of trees for different reasons and during such repairs, the parent of some nodes might change in the RPL DODAG tree. The movement of nodes might also cause changes in the RPL tree. Our protocol is

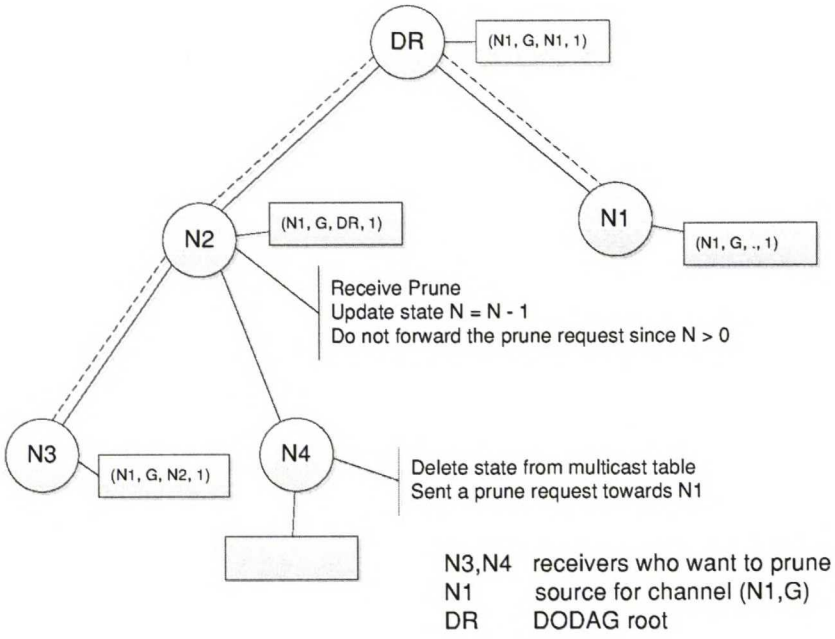

Figure 3.11: Prune Process

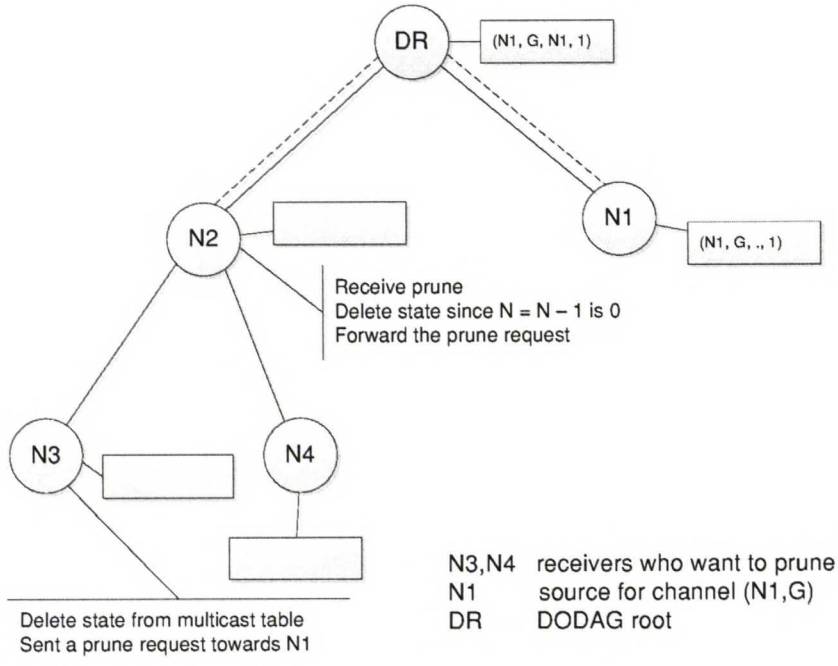

Figure 3.12: Prune Process (contd..)

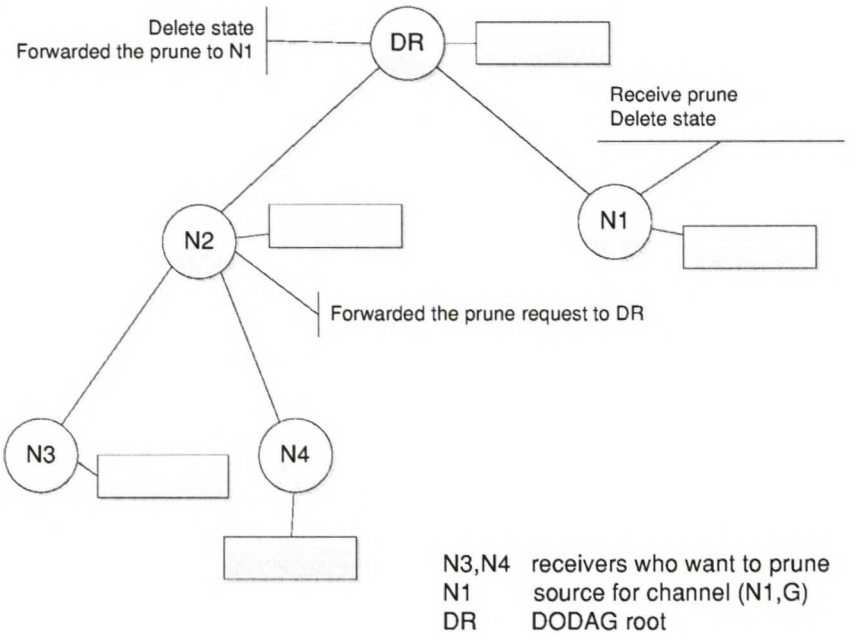

Figure 3.13: Prune Process (contd..)

designed to adjust accordingly to such changes and reconstruct the multicast tree. We will explain the process using an example of the multicast tree built for channel (DR,G*) as described in Section 3.4.1. In order to maintain the multicast tree, the nodes use Update messages. These messages are sent regularly by router nodes to their parent node in the multicast tree. The Figure 3.14 shows Update messages being sent by each router node that are part of the multicast tree to their respective parent node. When the router node receives an Update message, it knows that there are children still part of the tree and hence continues to be a part of the multicast tree. For example, in the figure, N2 keeps receiving Update message from N4. Thus it does not delete the state and in turn keeps sending Update messages to its parent node i.e DR. Each node waits for a preconfigured threshold time waiting for Update messages until it takes action. The frequency at which the nodes send Update message can also be preconfigured.

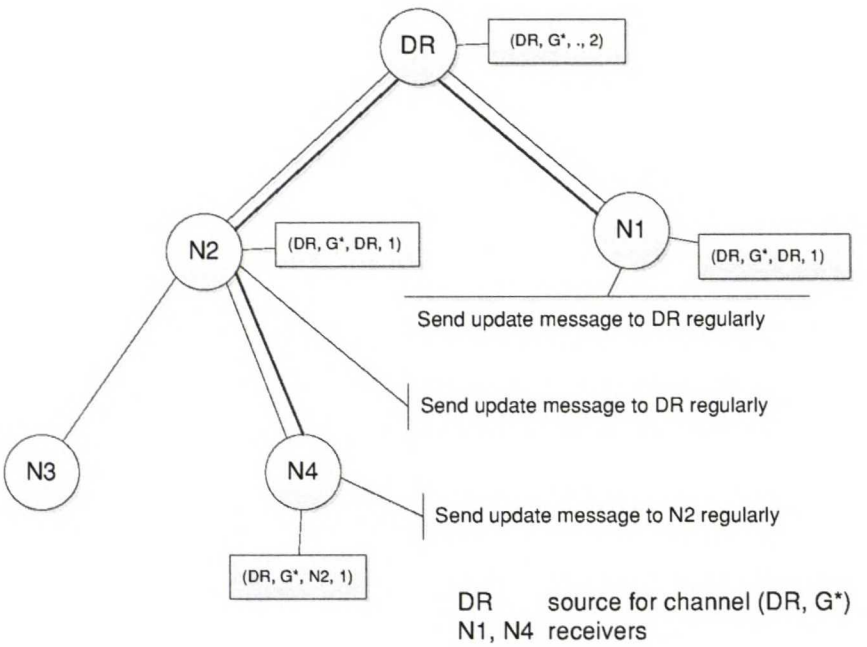

Figure 3.14: Tree Repair Process

In Figure 3.15, we show a scenario, where there is a change in the RPL tree. The node N4 has moved to a new location. Hence the RPL tree changes and N1 becomes the new parent of N4 in the RPL tree. Our protocol makes changes accordingly which is shown in Figure 3.16. Whenever a node detects a change in its route to the source of the channel, the node sends a new Join request towards the source of the channel, thus constructing a new branch of the multicast tree. In the example scenario, the node N4 detects a change in its next-hop router, i.e from N2 to N1. Hence it sends a new Join towards DR, and also Updates its state by changing the P variable. Meanwhile the node N2 keeps waiting for the Update messages for the

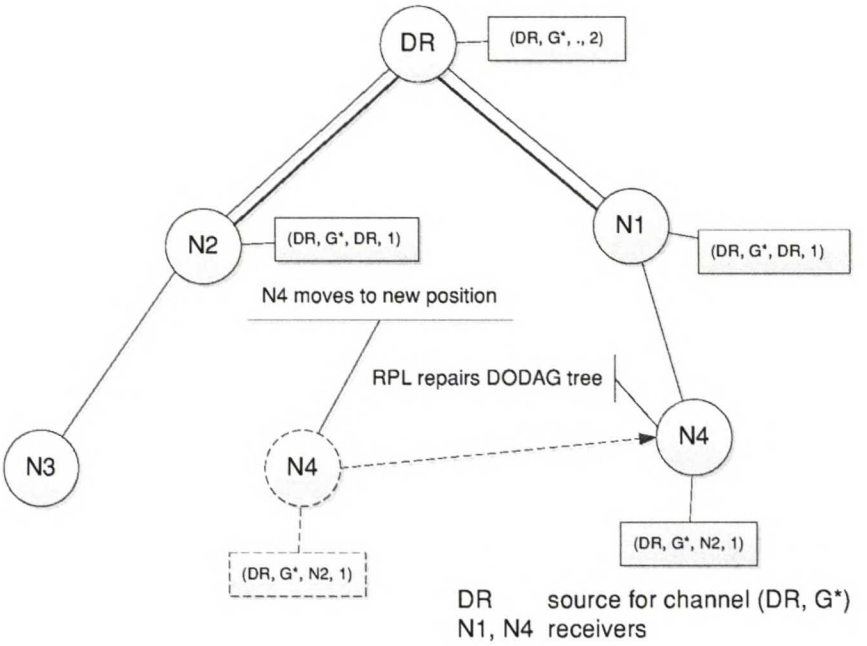

Figure 3.15: Tree Repair Process (contd..)

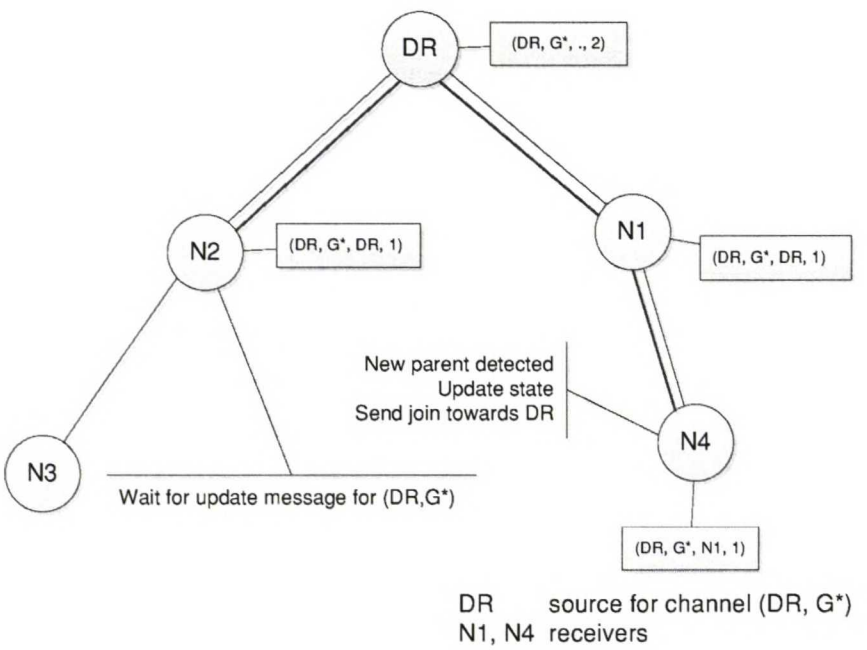

Figure 3.16: Tree Repair Process (contd..)

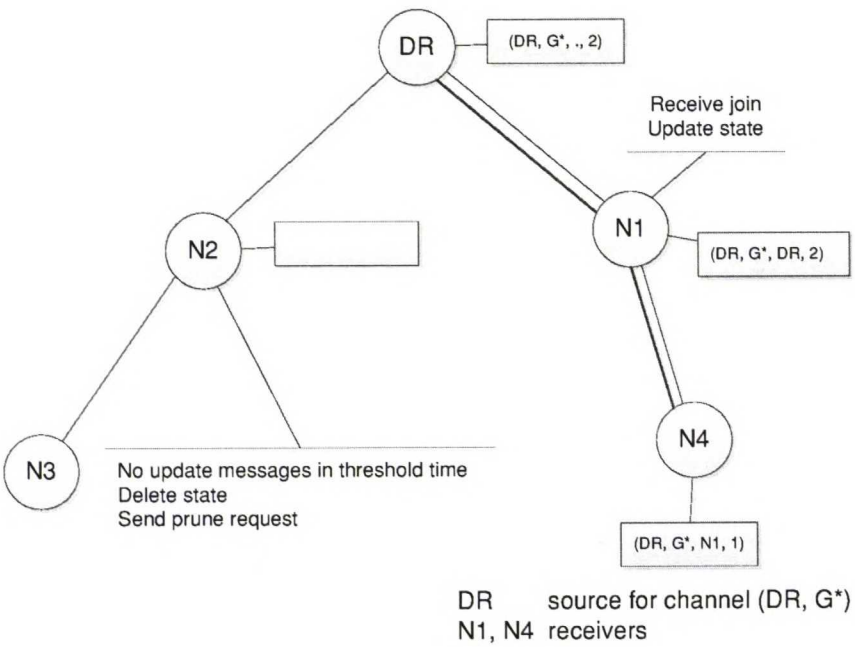

Figure 3.17: Tree Repair Process (contd..)

channel.

It is necessary that the stale branches of multicast trees be pruned. For example, in this case, the branch between N2 and DR is no longer required. Stale branches would result in data being unnecessarily broadcast which should be avoided. The pruning of stale branches is accomplished by Updates messages. In this case, the node N2 will wait for a threshold time expecting Update messages for that channel. When it realises that there are no longer any children listening to that channel, it deletes the state from its multicast table and initiates a Prune request towards the source of the channel. This is illustrated in Figure 3.17. We can also observe from the figure that a new branch has been established between N4 and N1. Thus the multicast tree has changed as per changes in the RPL tree.

4 Implementation

In this section, we will discuss the implementation of our protocol. We will describe the architecture of the contiki communication system and how our protocol fits into it. We will then discuss the components in the implementation and also include a few points about the data structures used.

4.1 Architecture

The communication system in contiki is shown in Figure 4.1. In the figure, we focus only on IPv6 packets since that is relevant for our implementation. The incoming and outgoing packet flow through the various components in the contiki network stack is also illustrated in the figure. An application sending packets would use the application interface provided by uIPv6 to send packets. From the uIPv6 layer, the packet flows to the 6LoWPAN layer to compress the headers and fragment the packet if necessary. The 6LoWPAN layer, then sends the packet through the MAC layer. The MAC layer puts the data packets in a queue. The radio duty cycling (RDC) layer sends the packets on this queue according to its duty cycling algorithm. The RDC layer uses the radio link layer to transmit the packets. The MAC layer does not remove the packet from the queue until it receives a link layer acknowledgement. The incoming packet flow is quite similar but in the opposite direction. In case of incoming packets, the 6LoWPAN layer is responsible for header decompression and reassembly of packets.

The layers in the communication stack are implemented as protothreads. However, in order to reduce code size, not all layers might be implemented as protothreads. The uIPv6, TCP and UDP layers are tightly coupled and is a single process. The TCP/UDP layer performs connection management i.e functions such as creating connection, listening on a port, connecting to a remote connection and so on. The uIP layer processes an incoming packet. It does checks on its header fields to decide if the packet is to be forwarded further or if it is intended for an application running in the same node. The RPL implementation in contiki maintains routing tables which is used by the uIP layer for forwarding packets. The RPL is not a separate process, but initialized by the uIPv6 and TCP/UDP layer. The RPL layer receives updates from another module regarding the state of the link to its neighbours. Based on these updates, RPL adjusts the forwarding tables accordingly. In addition to these mechanisms, the uIPv6 implementation also supports other functionalities such as neighbour discovery and stateless address configuration.

The 6LoWPAN layer in contiki is not a process, but initialized by the MAC layer process. For an outgoing packet, the relevant function in 6LoWPAN is invoked by the TCP/IP layer. For an incoming packet, the MAC layer calls the relevant function. The MAC layer is a separate process in itself. The default MAC layer is a CSMA/CA implementation. The unslotted version of IEEE 802.15.4 standard is supported. The RDC layer can either be a separate process or can also be initialized by the radio link layer. The ContikiMAC[44], a radio duty cycling algorithm designed by the developers of contiki, is the default RDC layer. Other than ContikiMAC, it has

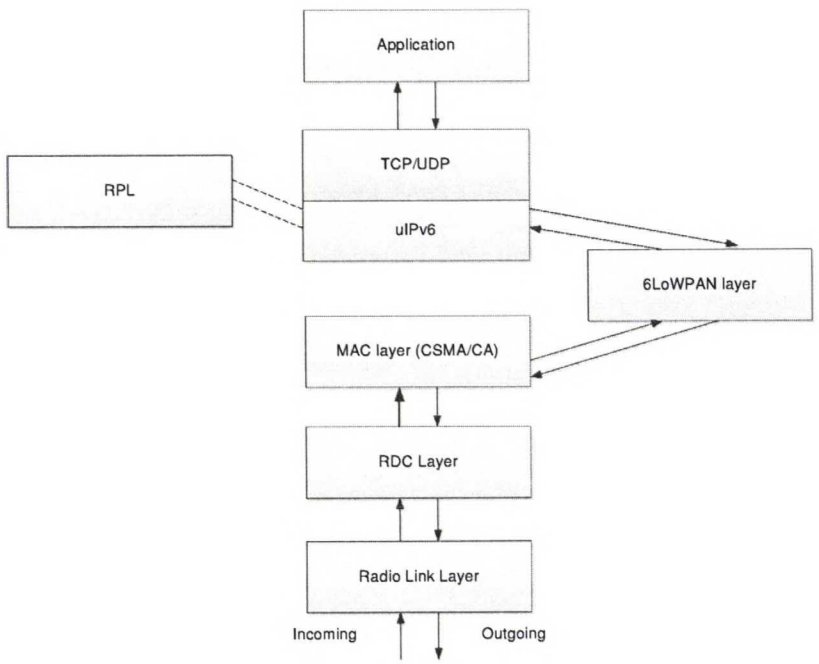

Figure 4.1: Contiki Communication Stack

two more options: X-MAC[45] and Low Power Probing (LPP)[46]. However, for our development and testing, we use NullRDC which does not perform any radio duty cycling mechanism. This means that the radio transceiver is kept on all the time. The radio link layer consists of the radio driver that is responsible for transmission and reception of packets.

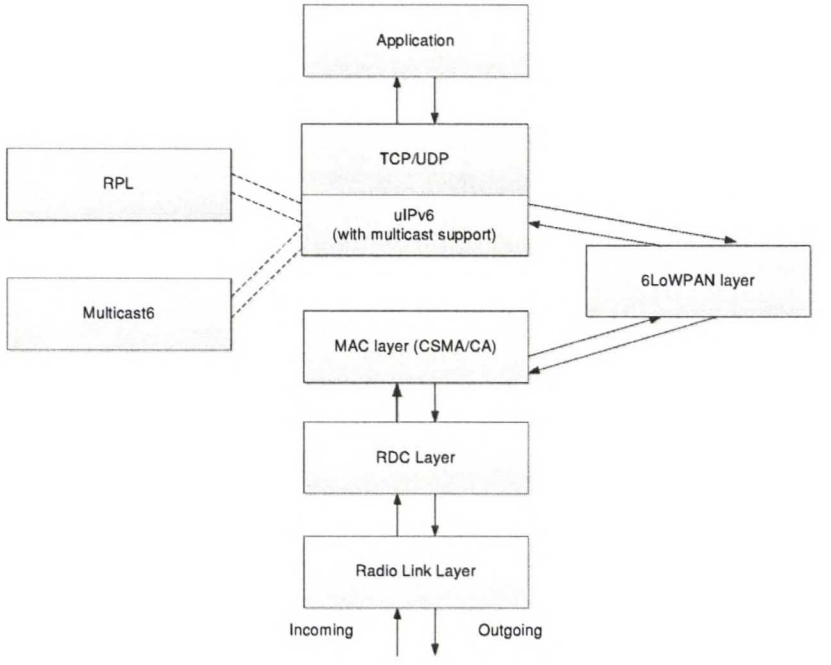

Figure 4.2: Contiki Communication Stack (with multicast)

The Figure 4.2 shows how our protocol fits in the contiki network stack. Similar to RPL, the multicast module will be initialized by the TCP/IP layer. The multicast module will be responsible for maintaining the multicast table. Whenever there is an incoming multicast packet, the uIPv6 will refer the multicast module to make decisions on whether to forward the packer further or to drop it. In order to construct its multicast table, the multicast module will use information in uIPv6 data structures that are maintained by RPL and other modules associated with uIPv6. The incoming and outgoing packet flow looks similar to the previous figure. However, at the uIPv6 layer, the multicast module is used to make forwarding decisions for multicast packets that have the destination address within the address range for SSM. In addition, the uIPv6 invokes the multicast module when there are multicast control packets i.e the Join, Prune and Update messages.

4.2 Components

The protocol can be divided into three logical components: control, data and the main component.

- Control

  The control component handles the multicast control packets. As mentioned earlier, the control packets are ICMPv6 messages with the Type number 201. Whenever the uIPv6 layer receives an ICMPv6 packet of the Type number 201, it calls the input handler in the control component. This is similar to the RPL functionality, where uIPv6 calls the relevant input handler of RPL when it receives an RPL control packet. This component is responsible for processing the control packet. After processing the packet and making any required changes in the multicast table, it either forwards the packet further or drops it depending on the scenario.

  The control component also provides functions that should be used to send control packets.

- Data

  The data component is used to handle data packets. Whenever the uIPv6 layer receives a packet whose destination address is within the address range specified for SSM, it invokes the input handler in the data component. However, the data component is not responsible for forwarding the packet. It checks if the packet satisfies the two mandatory conditions (explained in Section 3.3) required for forwarding a packet. Then, it informs the uIPv6 layer whether to drop the packet or broadcast it again. The uIPv6 layer finally takes action based on the information received from the data component.

- Main

  The main component is responsible for storage of the multicast table. It provides interface to the multicast table that can be used by other modules. This component is also responsible for handling Update messages and conducting tree repairs. A timer mechanism is used to periodically call a function that handles these functionalities.

  Although the main component and the data component are logically two separate entities, we have coded them in a single file in order to simplify the implementation.

4.2.1 Interaction with other modules

In the previous section, we mentioned about interfaces and handlers. In this section, we list all interactions with other modules and also within the components.

- The data and the control components have input handlers that are invoked by the uIPv6 layer. This has been explained in the previous section.

- The control packets can be sent using functions provided by the control component. Only the router nodes send control messages. Hence, these functions need not be used by applications running in nodes, but should be used by the group membership protocol running on the router. Other than this, the main

component also uses all these functions to send control messages. The Update message is sent only by the main component and not by any other module.

- As discussed before, the main component provides functions to manage the multicast table. These functions are used by the other two components. The functionalities required are: allocating a state in the table, finding a state and deleting a state. There are functions provided for the first two operations. However, deleting a state does not need a separate function call. In the next section, we explain briefly how it is done.
- The multicast module is initialized using a function invoked by the uIPv6 layer. This initialization function is defined in the main component.

4.3 Data structures and Timers

The main data structure required is for storage of states i.e the multicast table. The Listing 4.1 shows the declaration of an entry in the multicast table. It also shows the definition of the multicast table.

```
struct uip_mcast6_route {
    uint8_t used;
    uip_ipaddr_t sender_addr;
    uip_ipaddr_t mcast_grp;
    uip_ipaddr_t pref_parent;
    uint8_t num_of_joins;
    clock_time_t update_time;
};
typedef struct uip_mcast6_route uip_mcast6_route_t;

static uip_mcast6_route_t
       uip_mcast6_table[MAX_NUM_OF_MCAST6_GROUPS];
```

Listing 4.1: Data Structures

In the declaration of the structure *uip_mcast6_route*, the variables *sender_addr*, *mcast_grp*, *pref_parent* and *num_of_joins* represent the four variables in a state i.e S, G, P and N respectively. The variable *used* indicates if a routing state is in use or not. The variable *update_time* denotes the time when the most recent Update message was received. We define the multicast table as an array. The maximum number of entries can be preconfigured by defining the C preprocessor symbol MAX_NUM_OF_MCAST6_GROUPS. Since the multicast table is an array, the variable *used* indicates whether an entry in the array is in use or can be allocated for a new state. Setting the variable *used* to 0 means that the state has been deleted. Hence we do not need a separate function for deletion of states.

In contiki, there is an option to dynamically allocate memory using the memb memory block allocator. The memb library provides you a set of functions for dynamic memory allocation. However, a memory block has to be already allocated initially as an array which will be stored in the static memory. The memb library has been used to store uIPv6 data structures. But in case of uIPv6, there are various tables such as the routing table, neighbour cache etc. For our protocol, we have just one multicast table and hence there was no need of using the memb library. Instead of using array, if we use dynamic memory allocation, it would unnecessarily complicate the code and increase code size.

In order to implement the tree repair functionality, the protocol uses one timer. The ctimer library provided by contiki is used. In contiki, there are different timer libraries used to provide timer mechanisms. The ctimer library is used to invoke a callback function when the timer expires. The timer can then be reset inside the callback function. In our protocol, the initialization function sets this timer the first time. The frequency with which this callback function is called can be preconfigured. This callback function is responsible for the following two tasks related to tree repairs: detecting changes in route towards the source and handling Update messages. The process of tree repairs has been described earlier in Section 3.4.4.

5 Analysis

In this section, we discuss our analysis of the protocol. We have made several design choices to meet certain objectives as described in Section 3.2. However, as part of this thesis, we have not analysed all the objectives. We have tested the functionality of the protocol. The implementation works according to the design. In addition, we have done some preliminary performance analysis.

5.1 Functional Evaluation

In order to test the functionality of the protocol, we have conducted experiments using the cooja simulator. As discussed in Section 3, the protocol should be able to send and receive control messages, construct multicast distribution trees and send multicast data to interested routers.

In a complete multicast solution, a group membership protocol running on the router would be responsible for sending Join and Prune messages. Since we do not have a group membership protocol as of now, we have developed a command line shell that runs on nodes. On real hardware, the shell can be accessed over a serial USB connection. Even in cooja, it is possible to access the command line shell of a simulated node. The command line shell is available in contiki as an application and can be included while compiling your own application. Moreover, it can be customized to just run your own set of commands. For our testing, we have developed commands that do these four functions:

- Sending Join message
- Sending Prune message
- Sending multicast data
- Printing multicast table

Nodes	15 Zolertia[42] Motes
Radio Model	Unit Disk Graph Medium: Distance Loss
Transmission Range	50m
Interference Range	100m
Success Ratio TX	1.0
Success Ratio RX	1.0
RDC layer	NullRDC

Table 5.1: Parameters used for Testing Functionality

The parameters that we have used for testing are shown in Table 5.1. The default radio model in cooja is the Unit Disk Graph Medium (UDGM). In the distance loss version of this model, each node will have a transmission range and an interference

range. The nodes within its transmission range will always receive packets whenever it transmits. The nodes in its interference range will experience interference whenever it transmits. It is also possible to set the probability of successful transmission and reception, but for this experiment, we have set both of them to 1. The location of nodes on the X-Y plane are randomly chosen by cooja. Figure 5.1 shows the placement of nodes used in the experiment. The node 1 is the DODAG root and is responsible for setting up the RPL tree. We have used NullRDC as the radio duty cycling layer for this experiment. In case of NullRDC, the radio transceivers are always on.

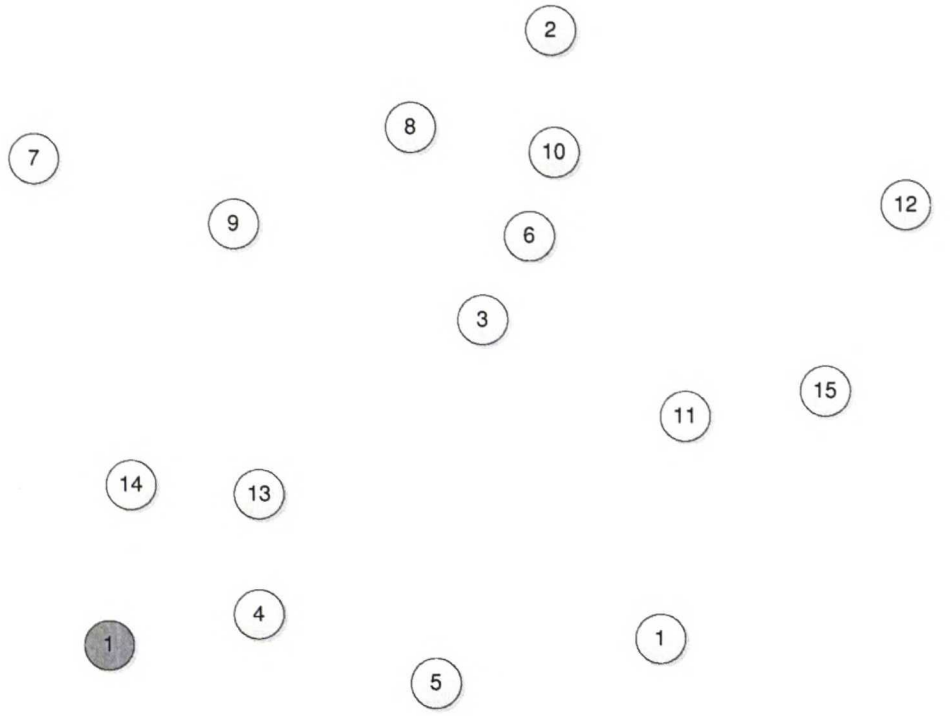

Figure 5.1: Network established by cooja

During the experiments, we tried different scenarios to test the four mechanisms as described in Section 3. The following tests were conducted.

- In order to test Join messages, we made nodes 12, 2, 7 and 15 join a channel with source as node 1. Later, we made nodes 7 and 2 join a channel with source as node 5. In both cases, the transmission of Join messages was successful and multicast trees for both channels were created. At each intermediate node, the states were added to the multicast tables. The merging of trees worked as expected. Any node that was part of both multicast trees had two states stored. For example, nodes 13 and 14 were part of both multicast trees and hence had two states stored in their table.

- To test data transmission we made the source nodes i.e node 1 and 5 send data on their respective group addresses. Whenever the data component in a router node receives data that has to be forwarded, we made them print a message along with the data. Thus, we were able to note the flow of data through the multicast tree. All the routers at the end of the multicast tree received the expected data messages.

- To test Update messages, we had made the nodes print a message along with its parent router address. At each node, we could see Update messages regularly transmitted to its parent in the multicast tree.

- Similar to Join messages, we sent Prune messages from nodes. We could see the expected changes in the tables in each of the intermediate routers. We also observed that the Update messages had stopped being sent from nodes that were no longer part of the tree.

- To test tree repairs, we changed the position of nodes so that they choose new parent nodes. However, before we conducted these tests, we also noticed that the tree repairs were already working. For example, the node 7 had 13 as its parent. Hence, initially in its multicast tree it selected node 13 as its parent. After some time, due to RPL mechanisms, node 7 changed its parent to node 14, and we could observe changes in the multicast tree immediately. In addition, we conducted tests where we explicitly changed positions of nodes and we could see that the tree repair mechanisms were working.

5.2 Performance Analysis

We have conducted some preliminary measurements to examine the performance of our protocol. As of now, we have just focused on two factors: packet delivery ratio and end-to-end delay. First, we will discuss the network setup used for the experiment, and in the subsequent sections, we will discuss the results.

Similar to the previous experiment, we have used the cooja simulator and compiled code for Zolertia motes. However, we have modified the code to accommodate the absence of a group management protocol. The network setup consists of 41 motes. Among the 41 motes, 1 is the DODAG root, 10 router nodes and 30 listener nodes. We will try to explain the difference between router nodes and listener nodes. In a real-world sensor network, not all nodes would be routers. Hence, we have made changes to differentiate between routers and listeners. However, the difference is only made for multicast data packets. In the experiments, only the 30 listener nodes will send Join requests to the sender. For the sake of simplicity, the DODAG root is made the sender and will transmit multicast data. When the sender transmits multicast data, the router nodes will definitely forward (broadcast) the multicast data towards its multicast children. The listener node will forward the multicast data only if it has more children and is not the end point of the multicast tree. However, note that the listener node will still forward multicast control packets and other non-multicast packets such as RPL packets etc. This design of

the experiment is to avoid unnecessary broadcasting of multicast data by all nodes, similar to real-world sensor networks, where the end nodes will not be routers and just consumers of multicast data.

Nodes	41 Zolertia motes (1 root, 10 routers and 30 listeners)
Radio medium	Unit Disk Graph Medium: Distance Loss
Transmission range	50m
Interference range	60m
Success ratio TX	1.0
Success ratio RX	1.0
RDC layer	NullRDC and ContikiMAC
Traffic	Constant Bit Rate (30 bytes application payload)

Table 5.2: Parameters used for Testing Performance

The parameters used for testing are shown in Table 5.2. The interference range has been reduced to 60m, whereas in the previous experiment it was 100m. We have used both ContikiMAC and NullRDC as the radio duty cycling layer. The experiment is performed with three different kinds of networks based on their Network Density (ND). If we consider a wireless sensor network as an undirected graph, then the network density can be derived from Equation 1. In a wireless network, if two nodes are in the interference range of each other, then there exists an edge connecting the two. Hence, in a complete graph, which means all the nodes are in the interference range of each other, the network density would be equal to 1. The listener nodes, upon booting and finding a route towards the DODAG root, will send a Join request towards it. After an initial delay of 75 seconds, the DODAG root, which is the sender node in this experiment will send multicast data. In the first few seconds of booting, there would be several RPL control messages being sent in order to construct routes between the nodes. Hence, we included an initial delay of 75 seconds so that the network becomes stable. We have developed a shell command to make the sender node send multicast data. This shell command sends a constant bit rate traffic of 30 bytes payload. The number of packets sent is 10. The 30 bytes is just application data and the whole packet would also include the headers. However, the actual size of the packet would vary because of header compression. In each network of a particular network density and RDC configuration, we carried out 4 experiments. Multicast traffic flows of 10 packets each were sent with the following intervals between two packet transmissions: 250 msec, 500 msec, 750 msec and 1 sec.

$$Network\ Density = \frac{Number\ of\ edges\ in\ the\ network}{Maximum\ number\ of\ possible\ edges} \tag{1}$$

5.2.1 Packet Delivery Ratio

Figure 5.2 and Figure 5.3 show the packet delivery ratio for NullRDC and ContikiMAC respectively. Both NullRDC and ContikiMAC have almost similar packet

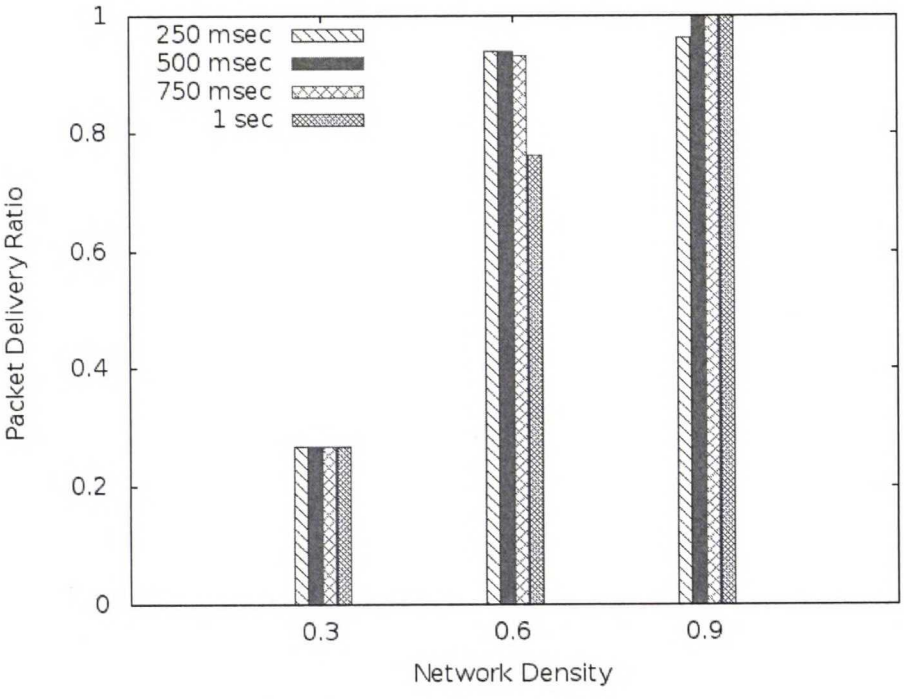

Figure 5.2: Packet Delivery Ratio with NullRDC

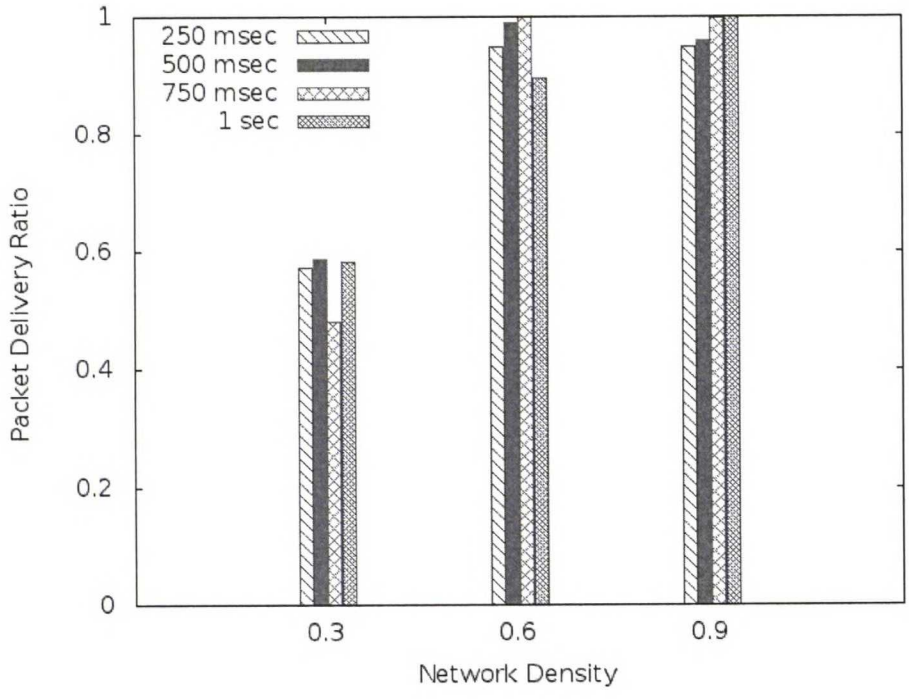

Figure 5.3: Packet Delivery Ratio with ContikiMAC

delivery ratios for ND = 0.6 and ND = 0.9. However, for ND = 0.3, NullRDC has higher packet loss rate than ContikiMAC. In case of NullRDC, the radio is kept on all the time. Hence it was expected to have better packet delivery ratio. The reason behind the unexpected behavior might be because of the way RDC layers handle broadcast packets and also because of the number of hops in low network densities. In the network with ND = 0.3, some of the listener nodes were more than 6 hops away from the sender node. As a result, there might be loss of packets in some of the intermediate nodes. Broadcast packets are handled differently by the two RDC layers. Since we use broadcast packets for transmission, this might make a difference. As mentioned before, in NullRDC, the radio is always on and hence the broadcast packet would be transmitted just once whenever the channel is free. In case of ContikiMAC, the radio uses periodic wake-ups to keep the radio on and listen for packets. Unicast packets are repeatedly sent until a link layer acknowledgement is received. Since there are no link layer acknowledgements for broadcast packets, the sender repeatedly sends the broadcast packet during an entire wake-up period to make sure that all the neighbours receive the packet. This behaviour of ContikiMAC probably has resulted in better packet delivery ratio than NullRDC.

5.2.2 End-to-End delay

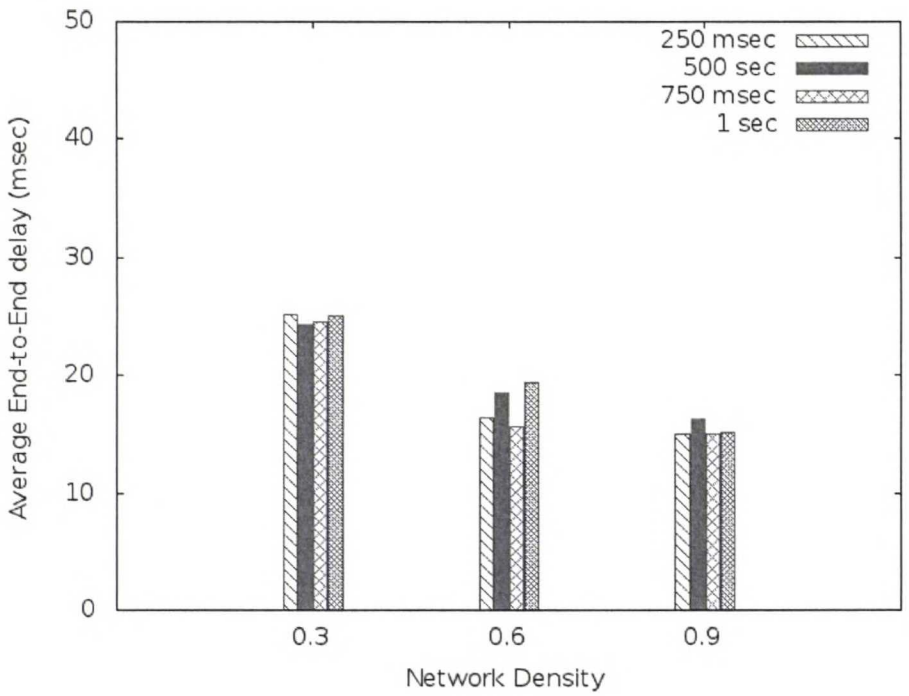

Figure 5.4: Average End-to-End Delay with NullRDC

Depending on the number of hops required to reach a listener node, the end-to-end delay would vary. We have calculated the average end-to-end delay and shown

it in our figures. The listeners in networks with lower network density will have more hops to be reached. Hence, they will have higher end-to-end delays. We can notice this result in Figure 5.4 and Figure 5.5. However, with NullRDC, there is a small difference in delays even between networks with different network densities. From the figures, we can also see that ContikiMAC has much higher end-to-end delay when compared to NullRDC. This can be explained by how ContikiMAC handles broadcast packets as discussed in the previous section. Because of this technique used by ContikiMAC, it takes longer time for broadcast transmissions[47]. In our figures, we have not included the maximum end-to-end delay i.e the total multicast transmission time required for data to reach all the listeners. In most cases, this would also be the time required for the data to reach the furthest node. As expected, it is much higher in case of ContikiMAC than in NullRDC.

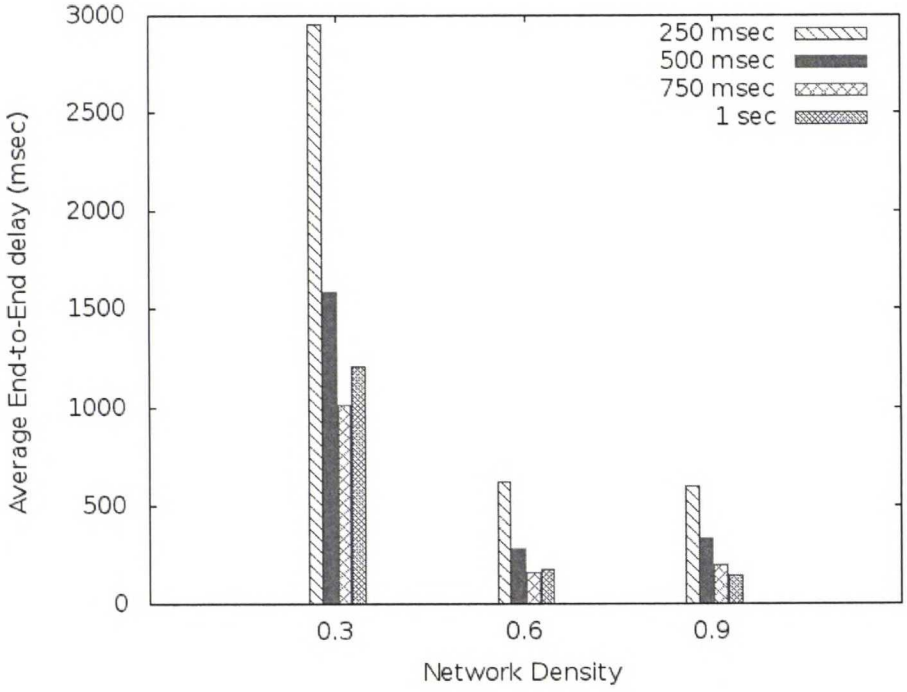

Figure 5.5: Average End-to-End Delay with ContikiMAC

5.2.3 Discussions

As discussed in Section 1, SMRF and MPL are multicast routing protocols that have been designed for LLNs. In the article on SMRF[7], the authors have compared the performance of MPL and SMRF. However, note that the authors refer to an older version of the Internet draft of MPL. They use a network of 21 nodes with different network densities and compare packet delivery ratio, end-to-end delay and energy. We have not conducted any tests on energy, but we can examine the other two parameters. Surprisingly, MPL and SMRF show better packet delivery ratios with

NullRDC, whereas in our experiments ContikiMAC showed better results. Both MPL and our protocol have similar packet delivery ratios, which is higher than that of SMRF. In the article, instead of average end-to-end delay, the total multicast transmission time per packet is compared. Our protocol exhibits much lower end-to-end delay compared to MPL, whereas SMRF has slightly better performance than our protocol.

Although we have done some preliminary performance evaluation, more detailed analysis is required based on other factors. Energy consumption is a crucial factor that has to be inspected. In our design, we use broadcast packets for data transmission, but the data does not flood throughout the network. Hence, we expect the overall energy consumption to be less. However, a point of concern is the heartbeat mechanism. The regular transmission of Update messages would probably cause an increase in energy consumed by the devices. We also have to test the protocol behaviour in other network scenarios (different transmission/reception success ratios, heterogeneous hardware devices, etc.). Furthermore, we can evaluate the protocol on a testbed made of real devices to validate the accuracy of our experiments.

6 Future work

In this section, we discuss some ideas to improve the protocol and also about work that has to be done in order to have a complete multicast solution. In addition, we have already discussed in Section 5.2.3 that further performance analysis of the protocol has to be done.

6.1 Group Membership Protocol

As discussed before, a complete multicast solution also needs a group membership protocol. We had also stated that MLDv2 protocol could be a basis to develop group membership protocol for our solution. While designing this protocol, we have to keep in mind the following points:

- Both the routers and leaf nodes will be using the group membership protocol. The group membership protocol running on the router will interact with the multicast routing protocol on the router.
- The router nodes upon receiving request from its leaf nodes to subscribe to a channel, will send a Join message to the source of the channel and add the state in its multicast table.
- The N variable of a multicast state does not change depending on the number of requests from its leaf nodes. The N variable is affected when it receives multicast control messages from other routers.
- The source node can either be a router or a leaf node. The router can also be a receiver. These scenarios should be considered.

6.2 Routing between DODAG trees

The multicast protocol that we designed can be used by nodes within a single DODAG tree having a single DODAG root. However, in real world applications DODAG trees would be connected to each other and also to the internet through their DODAG roots. The DODAG roots would act as border routers and can be connected to the Internet over a backbone link. A LoWPAN could have more than one border router and both of them can be connected to the internet. There could be different ways in which the DODAG roots are connected to each other and to the internet. In all these cases, there might be a necessity to have multicast communication between DODAG trees. Hence, a protocol has to be designed so that DODAG roots can exchange multicast information and thus support multicast between DODAG trees.

6.3 Improvements to the Protocol

During the course of this work, we came across certain ideas that could improve the protocol.

In our current design, the Update messages are transmitted regularly. The frequency with which the Update messages are sent can be preconfigured. We also configure the threshold value to specify how long a node has to wait for Update messages before it takes relevant actions. These values can be set by the application developer depending on the requirement. For example, if energy usage is a crucial factor, then the frequency could be reduced. However, the transmission of Update messages has a direct impact on the energy consumption of the node. Hence, this can be designed better to reduce energy consumption. In case of RPL, the frequency with which nodes exchange routing information is controlled by the trickle timer. Initially, they exchange messages with higher frequency, but then they slow down. Whenever an inconsistency is detected in the DODAG tree, the frequency increases again. A similar mechanism could be developed for our multicast protocol.

Some minor changes to implementation can be made to reduce code size. As discussed in Section 4, the protocol implementation consists of three logical components. These three components have been coded in two C files. Integrating these three components into a single C file will reduce code size.

7 Conclusion

The goal of the project was to develop a multicast routing protocol for 6LoWPANs. The multicast routing protocol has been designed and implemented on contiki OS. As part of the design process, we examined different multicast solutions developed for conventional networks. Among them, many features of the SSM protocol was found suitable to be implemented in LLNs. The SSM protocol and its parent protocol PIM, are designed to be independent of the unicast protocol used in the network. They use routing information from the underlying unicast protocol to build their multicast trees. However, in our case, we have used RPL as the unicast protocol. Although we have not explored any other unicast protocols, our protocol seems to be independent of the unicast protocol used since it is based on SSM. We have done several modifications to SSM so that it works in 6LoWPANs. Many unnecessary features of SSM have been removed. These features are useful in conventional networks but are not needed in LLNs. In constrained networks, we have the problem of disruptions in network due to movement of devices, weak signals, etc. Although RPL has been designed to consider such changes in the network, we had to develop additional features in our protocol to be aware of such changes and act accordingly.

The implementation has been tested to confirm functionality. We have also conducted preliminary performance analysis using the cooja simulator. In our analysis, we examine two factors: packet delivery ratio and end-to-end delay. The results from our experiments look promising. However, we still need to further analyse the protocol focusing on other factors such as energy consumption. The Update messages which we have used to take care of network repairs might be a cause of increased energy consumption in devices.

In addition to our routing protocol, a group management protocol has to be implemented in order to have a complete multicast solution. Based on further evaluation of our protocol focusing on energy efficiency, we could still improve it. As mentioned earlier, one of the main concerns that we have are regular Update messages. A possible solution to reduce their frequency is to introduce trickle timers similar to how RPL manages control messages. Although there is need for further development and improvement, our work so far suggests that a multicast routing protocol based on SSM is feasible to be used in 6LoWPANs.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, pp. 2787–2805, Oct. 2010.

[2] A. Walter-Krisch, "Heading towards 50 billion connections." `http://www.ericsson.com/oss-bss/blog/heading-towards-50-billion-connections/`, 2011.

[3] G. Mulligan, "The 6LoWPAN architecture," in *Proceedings of the 4th workshop on Embedded networked sensors*, EmNets '07, (New York, NY, USA), pp. 78–82, ACM, 2007.

[4] K. D. Korte, A. Sehgal, and J. Schönwälder, "A study of the RPL repair process using ContikiRPL," in *Proceedings of the 6th IFIP WG 6.6 international autonomous infrastructure, management, and security conference on Dependable Networks and Services*, AIMS'12, (Berlin, Heidelberg), pp. 50–61, Springer-Verlag, 2012.

[5] Z. Shelby, K. Hartke, and C. Bormann, "Constrained Application Protocol (CoAP)," Internet-Draft draft-ietf-core-coap-18, IETF Secretariat, Dec. 2013.

[6] J. Hui and R. Kelsey, "Multicast Protocol for Low Power and Lossy Networks (MPL)," Internet-Draft draft-ietf-roll-trickle-mcast-04.txt, IETF Secretariat, Aug. 2013.

[7] G. Oikonomou and I. Phillips, "Stateless Multicast Forwarding with RPL in 6LowPAN Sensor Networks," in *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pp. 272–277, 2012.

[8] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, "The Trickle Algorithm," RFC 6206, Mar. 2011.

[9] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "The PIM architecture for wide-area multicast routing," *Networking, IEEE/ACM Transactions on*, vol. 4, no. 2, pp. 153–162, 1996.

[10] S. Bhattacharyya, "An Overview of Source-Specific Multicast (SSM)," RFC 3569, July 2003.

[11] J. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile, "IEEE 802.15.4: a developing standard for low-power low-cost wireless personal area networks," *Network, IEEE*, vol. 15, no. 5, pp. 12–19, 2001.

[12] E. Callaway, P. Gorday, L. Hester, J. Gutierrez, M. Naeve, B. Heile, and V. Bahl, "Home networking with IEEE 802.15.4: a developing standard for low-rate wireless personal area networks," *Communications Magazine, IEEE*, vol. 40, no. 8, pp. 70–77, 2002.

[13] N. Sastry and D. Wagner, "Security considerations for IEEE 802.15.4 networks," in *Proceedings of the 3rd ACM workshop on Wireless security*, WiSe '04, (New York, NY, USA), pp. 32–42, ACM, 2004.

[14] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks." RFC 4944 (Proposed Standard), Sept. 2007. Updated by RFCs 6282, 6775.

[15] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks." RFC 6550 (Proposed Standard), Mar. 2012.

[16] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, (Tampa, Florida, USA), Nov. 2004.

[17] A. Dunkels, O. Schmidt, and T. Voigt, "Using Protothreads for Sensor Node Programming," in *Proceedings of the REALWSN'05 Workshop on Real-World Wireless Sensor Networks*, (Stockholm, Sweden), June 2005.

[18] A. Dunkels, "Full TCP/IP for 8 Bit Architectures," in *Proceedings of the First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003)*, (San Francisco), usenix, May 2003.

[19] A. Dunkels, J. Eriksson, N. Finne, F. Osterlind, N. Tsiftes, J. Abeille, and M. Durvy, "Low-power IPv6 for the Internet of Things," in *Networked Sensing Systems (INSS), 2012 Ninth International Conference on*, pp. 1–6, 2012.

[20] M. Durvy, J. Abeillé, P. Wetterwald, C. O'Flynn, B. Leverett, E. Gnoske, M. Vidales, G. Mulligan, N. Tsiftes, N. Finne, and A. Dunkels, "Making sensor networks IPv6 ready," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, (New York, NY, USA), pp. 421–422, ACM, 2008.

[21] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification." RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946.

[22] R. Hinden and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture." RFC 3513 (Proposed Standard), Apr. 2003. Obsoleted by RFC 4291.

[23] T. Narten, E. Nordmark, W. Simpson, and H. Soliman, "Neighbor Discovery for IP version 6 (IPv6)." RFC 4861 (Draft Standard), Sept. 2007. Updated by RFC 5942.

[24] A. Conta, S. Deering, and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification." RFC 4443 (Draft Standard), Mar. 2006. Updated by RFC 4884.

[25] R. Draves, "Default Address Selection for Internet Protocol version 6 (IPv6)." RFC 3484 (Proposed Standard), Feb. 2003. Obsoleted by RFC 6724.

[26] S. Thomson, T. Narten, and T. Jinmei, "IPv6 Stateless Address Autoconfiguration." RFC 4862 (Draft Standard), Sept. 2007.

[27] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," in *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006)*, (Tampa, Florida, USA), Nov. 2006.

[28] S. Deering, "Host extensions for IP multicasting." RFC 1112 (INTERNET STANDARD), Aug. 1989. Updated by RFC 2236.

[29] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "An architecture for wide-area multicast routing," *SIGCOMM Comput. Commun. Rev.*, vol. 24, pp. 126–135, Oct. 1994.

[30] J. Moy, "OSPF Version 2." RFC 2328 (INTERNET STANDARD), Apr. 1998. Updated by RFCs 5709, 6549, 6845, 6860.

[31] C. Hedrick, "Routing Information Protocol." RFC 1058 (Historic), June 1988. Updated by RFCs 1388, 1723.

[32] D. Waitzman, C. Partridge, and S. Deering, "Distance Vector Multicast Routing Protocol." RFC 1075 (Experimental), Nov. 1988.

[33] J. Moy, "Multicast Extensions to OSPF." RFC 1584 (Historic), Mar. 1994.

[34] A. Ballardie, "Core Based Trees (CBT version 2) Multicast Routing – Protocol Specification –." RFC 2189 (Historic), Sept. 1997.

[35] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)," RFC 4601, Aug. 2006.

[36] H. Holbrook and B. Cain, "Source-Specific Multicast for IP," RFC 4607, Aug. 2006.

[37] W. Fenner, "Internet Group Management Protocol, Version 2." RFC 2236 (Proposed Standard), Nov. 1997. Updated by RFC 3376.

[38] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, "Internet Group Management Protocol, Version 3." RFC 3376 (Proposed Standard), Oct. 2002. Updated by RFC 4604.

[39] S. Deering, W. Fenner, and B. Haberman, "Multicast Listener Discovery (MLD) for IPv6." RFC 2710 (Proposed Standard), Oct. 1999. Updated by RFCs 3590, 3810.

[40] R. Vida and L. Costa, "Multicast Listener Discovery Version 2 (MLDv2) for IPv6." RFC 3810 (Proposed Standard), June 2004. Updated by RFC 4604.

[41] H. Holbrook, B. Cain, and B. Haberman, "Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast." RFC 4604 (Proposed Standard), Aug. 2006.

[42] "Z1 Platform." `http://www.zolertia.com/ti`.

[43] "Z1 Features: Quick Hardware Tour." `http://zolertia.sourceforge.net/wiki/index.php/Z1`.

[44] A. Dunkels, "The ContikiMAC Radio Duty Cycling Protocol," Tech. Rep. T2011:13, Swedish Institute of Computer Science, Dec. 2011.

[45] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pp. 307–320, ACM, 2006.

[46] R. Musaloiu-E, C.-J. Liang, and A. Terzis, "Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks," in *Information Processing in Sensor Networks, 2008. IPSN '08. International Conference on*, pp. 421–432, 2008.

[47] A. Dunkels, L. Mottola, N. Tsiftes, Fredrik Österlind, J. Eriksson, and N. Finne, "The Announcement Layer: Beacon Coordination for the Sensornet Stack," in *Proceedings of EWSN 2011*, (Bonn, Germany), Feb. 2011.

[48] S. Deering and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," tech. rep., RFC Editor, United States, 1998.

[49] L. Bartolozzi, T. Pecorella, and R. Fantacci, "ns-3 RPL module: IPv6 routing protocol for low power and lossy networks," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, (ICST, Brussels, Belgium, Belgium), pp. 359–366, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2012.