TU Clausthal

EXTREMELY LONG AND COMPLICATED TITLE OF AN EXCELLENT THESIS PAPER (WRITTEN IN LaTeX)

ITIS RESEARCH PROJECT REPORT

presented by

ADITYA RAJ

Embedded Systems group
Department of Informatics
Technische Universität Clausthal

ES-P005

[draft created on April 12, 2017 at 23:59 (delete `drafting` option in `thesis-config.tex` to remove this line)]

Aditya Raj: *Extremely Long and Complicated Title of an Excellent Thesis Paper (written in LaTeX)*

STUDENT ID
123456

ASSESSORS
First assessor: Dr.-Ing. Andreas Reinhardt
Second assessor: Prof. Dr. Sven Hartmann

SUBMISSION DATE
15 April 2017

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

STATUTORY DECLARATION

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, wurden als solche kenntlich gemacht. Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsstelle vorgelegt.

Ich erkläre mich zudem mit der öffentlichen Bereitstellung meines ITIS Resarch Project Report in der Instituts- und/oder Universitätsbibliothek einverstanden.

Clausthal-Zellerfeld, 15 April 2017

Aditya Raj

[draft created on April 12, 2017 at 23:59 (delete `drafting` option in `thesis-config.tex` to remove this line)]

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

ABSTRACT

Short summary of the contents in English...

A great guide by Kent Beck how to write good abstracts can be found at:

https://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html

It is particularly important to only use the language that has been specified in its surrounding `otherlanguage` block to cater to proper hyphenation.

ZUSAMMENFASSUNG

Hier eine kurze Zusammenfassung des Inhaltes in deutscher Sprache...Sollten Sie die Thesis auf Englisch schreiben, ist dennoch eine deutsche Zusammenfassung notwendig. Im umgekehrten Fall kann die englische Zusammenfassung durch Auskommentieren weggelassen werden.

[draft created on April 12, 2017 at 23:59 (delete `drafting` option in `thesis-config.tex` to remove this line)]

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

CONTENTS

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

ACRONYMS

BS Base Station

WSN Wireless Sensor Network

SN Sensor Node

CTP Collection Tree Protocol

MAC Medium Access Acontrol

PC Processing Center

CRC Cyclic Redundancy Check

AM Active Message

PRR Packet reception rate

ETX Expected number of transmissions

RSSI Received Signal Strength Indicator

LQI Link Quality Indicator

SFD Start Frame Delimiter

RTS Request To Send

CTS Clear To Send

DVR Distance Vector Routing

OLSR Optimal Link State Routing

RREQ Route Request

RREP Route Reply

DSR Dynamic Source Routing

AODV Ad-hoc On Demand Distance Vector Routing Protocol

RISC Reduced Instruction Set Computing

UDGM Unit Disk Graph Medium

TCP Transmission Control Protocol

UDP User Datagram Protocol

FIFO First In First Out

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

LISTE DER NOCH ZU ERLEDIGENDEN PUNKTE

[draft created on April 12, 2017 at 23:59 (delete `drafting` option in `thesis-config.tex` to remove this line)]

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

1 INTRODUCTION

1.1 CONTEXT

In the past decade, Wireless Sensor Networks (WSNs) have shown tremendous capabilities in several application domains. This has been possible due to advancements in communication technologies and electronics, which has enabled researchers to use tiny sensors at low cost. Being limited in energy and computation power, these devices gather information from surroundings and send it to a central server, also known as Base Station (BS), for processing. Some of the application domains for WSN deployment include environmental or biodiversity monitoring, military surveillance, home automation, increasing agricultural yield by monitoring atmospheric and soil conditions, disaster-assistance and so on.

These battery operated devices with limited communication range (because of low energy budget) have greater advantage over everlasting energy budget devices, when infrastructural availability may not be possible due to remoteness of a geographical location or poor cost-benefit ratio to set up the whole infrastructure for a limited period of time. A basic requirement to enable such a self-sufficient data routing WSN system is to propagate the neighborhood information containing the Sensor Node (SN) identifier among the members of the WSN. Using the stored tabular information of the neighborhood, one could deploy gossiping algorithms, where each node randomly selects one of its neighbors for message exchange, or perform best neighbor selection based on several possible metrics. However, with an increase in number of SNs, the whole system may become collision-prone and therefore the overall throughput of the system can considerably decrease because of frequent message exchange collisions.

Heterogeneity can mitigate such kind of problems. The concept of introducing heterogeneity in a WSN involves deploying dissimilar energy and computational power SNs in the network. In order to save energy in long range transmissions, the communication range for the heterogeneous SNs are same for all the sensors in our network model. Implementation of such a protocol is quite similar to setting up a general homogeneous network model. The only difference towards realising heterogeneity is to add an additional layer to the routing protocol. Therefore, the overall set up phase requires all the steps necessary to set up a routing protocol at first place and further realise a

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

heterogeneity layer above it. The general idea for setting up a routing protocol the following concepts:

1. Routing protocols selection to find a route from source to destination
2. Multi-hop networking to transmit data (because of limited communication range SNs can not communicate to base-station directly)
3. Controlled channel access using Medium Access Acontrol (MAC) protocols (to determine which SN transmits at a given point of time)
4. Efficient neighborhood discovery and optimal intermediate SN selection for sending out the locally gathered data, will be the foundation for deploying heterogeneity in a WSN.

1.2 PROBLEM STATEMENT

Generally, we deploy SNs for *Event Detection* or *Spatial Process Estimation* scenarios. In a Wireless Sensor Network (WSN), for small data traffic or low processing complexity, communication and local computation, in any of the above mentioned deployment scenarios, is not a big concern. Problem arises when there is a need to do large amount of data processing operations such as continuous audio/video stream processing, or perform high computation tasks, like Fourier transform of input signal. Therefore, as the amount of data grows or the computational complexity increases, feasible and efficient options other than local aggregation and computation are preferred. An easier solution to this would be to increase the data processing capabilities and energy restrictions for all of the deployed homogeneous SNs. However, increasing the computation power for each of the device in the network will quickly exhaust the total energy budget of the network. Therefore such a protocol is hard to realise in remote locations where eminent power supply is not possible.

Thus to accommodate the design goal of minimal energy consumption and higher computational challenges for low power SNs, efficient data collection and information processing protocols are required. We here argue that in order to optimise the available resources, intra-network data processing can be deployed.

We propose to explore the possibility of getting the data processed by a nearby SN with higher computation power and eminent energy backup. We believe that heterogeneity could play a role in this direction, by pulling away the load of large amount of data processing

locally, by low power nodes, and instead agree to do the task utilizing it's own, much higher compute power. We will interchangeably refer these higher energy budget and higher computational power heterogeneous SNs with Processing Centers (PCs).

The problems we wish to address with this research project are summarised below:

1. Implement heterogeneity in a simplified way by connecting some of the wireless nodes to a more powerful device over usb serial connection.
2. Allow heterogeneity implementation as a plugin to general routing protocols.
3. Demonstrate benefits of adding heterogeneity layer over Collection Tree Protocols (CTPs).
4. Evaluate reliability, efficiency, robustness and end to end delivery ratio of a WSNs comprising of heterogeneous SNs.

1.3 MOTIVATION

Motivation for the project

Wireless SNs have gained a great amount of attention in recent years. The most suitable reason for their popularity is the ease of deployment in unreachable and unattained areas for sensing and collecting the data from the surrounding and reporting it to the Base Station. Several power-efficient protocols have been proposed in this domain. Energy efficiency is always thought as a major concern while choosing the protocols for a WSN. We mostly think in terms of homogeneously working nodes in the sensors neighborhood and further aim to realize the goal of reducing energy-consumption. From an out of box approach, we here explore the possibility of exploiting heterogeneity among SNs and use it as a leverage to solve the problem of energy consumption and efficient data processing.

Motivation for using the Platforms: Telosb and TinyOS

For research purposes, it is not a good idea to use application specific devices like Digital Signal Processors (DSPs), Field-Programmable Gate Array (FPGAs) or Application-specific integrated circuits (ASICs), which will lead to better computational efficiency, but reduced flexibility in terms of re-programmability and/or energy consumption.

Therefore, to address the challenge of large computation and low-energy budget, we decide to experiment with Telosb devices and use Tiny Operating System(TinyOS), for the research purpose. Telosb device is widely chosen by research community as it not only allows re-programmability but also can be operated on low power supply (2 Batteries of 1.5V each). Adding extra compute power in a Wireless Sensor Network (WSN), can be realized in two ways:

- Buying a new device with high compute power and using this as a heterogeneous SN placed across the network.
- Using the same Telosb device and attaching it to Raspberry Pi device, which adds extra compute power to the SN. We here also assume that heterogeneous nodes have higher energy capacity. However, this is a workaround to the normal practice of using a complete different device as a heterogeneous node.

First option is not a quite good approach for two reasons: 1. Most of the SNs that exist today, differ in connectivity or micro-controller architecture. Also, for many of the devices, TinyOS is not supported. 2. Time limitation to test a working communication interface of a Telosb device with completely new device because an entirely different instrument will have dissimilar communication bandwidth and protocol requirement.

Second option has three major advantages: 1. Communication interfacing with similar devices will not be a problem as they run on same bandwidth and other channel requirements. 2. Raspberry Pi or a serial connection from Telosb with a high compute power device such as desktops can be added as an extension to Telosb easily. 3. The entire focus of the research can be shifted to implement the heterogeneity solution and not re-inventing the wheels for setting up and running the communication link between two different devices.

Motivation for using CTP

An important question arises in this context is: Why do we need collection protocols?

The authors **[TEP:119]** have answered it as – *"Collecting data at a base station is a common requirement of WSN applications. The general approach used is to build one or more collection trees, each of which is rooted at a base station. When a node has data which needs to be collected, it sends the data up the tree, and it forwards collection data that other nodes send to it."*

CTP has become a de-facto standard in collection routing in WSN. It is generally used as a benchmark by researchers to evaluate new de-

signs or new protocol mechanisms [**moeller2010routing**], [**Hackmann:2008**], [**Alizai:2009**].

Moreover, CTP [**Gnawali:2013**] has been tested on thirteen different testbeds, seven hardware platforms include Telosb, six different link layers and multiple densities and frequencies, and detailed observations of a long-running WSN application that uses CTP. As claimed by authors, CTP has across all testbeds, configurations, and link layers delivered end-to-end delivery ratio ranges from 90.5%to99.9%.

Other main reasons why CTPs is hard to beat as compared to other routing algorithms are:

1. It is largely platform independent.
2. It is seen as a benchmark algorithm as it can be used on diverse set of platforms.
3. It is robust, reliable and efficient.

1.4 SIGNIFICANCE OF RESEARCH

We focus on the problem of sending data to one of the available PCs in two hop neighborhood. We believe that we can provide a new insight into tackling the problem of large amount of data processing through the heterogeneity concept, which was earlier possible only once the data reaches the BS. This will also imply a reduced number of packet transmissions from a SN to the BS because of the pre-processing of data by a PC prior to reaching the BSs.

Also, in an unstructured deployment of SNs, it becomes necessary to achieve the best performance out of the deployed PC in one or two hop neighborhood. Therefore, our second aim is to compare our heterogeneity protocol with CTPs and demonstrate how significantly a network can benefit by adding a heterogeneity layer on top of data collection layer. This will also establish the fact that the heterogeneity network will increase the overall energy budget of the WSNs.

1.5 ASSUMPTIONS AND LIMITATIONS

We assume the following scenario for the research project:

1. All sensors are not homogeneous. They are dissimilar in terms computational power and energy budget.
2. SNs are randomly distributed and are not configured with knowledge of their location.

3. SNs must send wireless queries to surrounding nodes to discover the network.
4. Implementation and testing of protocols is done on TinyOS [**TinyOS**] Operating System. The sensor node device used for testing and deployment is Telosb MSP430 [**crossbow:TELOSB**].
5. Telosb Devices attached to Raspberry PI will act as heterogeneous sensor nodes in the Wireless Sensor Network. Raspberry PI will provide raised computation power to the low power Telosb device.
6. There is very high energy budget for the heterogeneous high compute power devices.
7. Sensors have limited sensing and sending range(around 10metres).
8. The communication is based on the single-hop and the communication link is symmetric.
9. A Collection Protocol runs at back end for sensor data collection at Base Station.

1.6 STRUCTURE OF THE REPORT

Chapter 2 sets out with a background of the WSN structure, application space, describes general programming structure of nesC and TinyOS, and introduces the particularities in networking and routing. It further briefly presents the existing deployment of CTP. Chapter 3 presents a brief overview of the related work in the field of local data processing from two approaches: cluster-based data aggregation and heterogeneity. Chapter 4 describes the hardware and software platforms used for the heterogeneity implementation, as well as the design components of the heterogeneity layer. It follows up with routing design and data transfer design of heterogeneity. Finally, we summarise the heterogeneity process with the help of timer flow charts. Chapter 5 evaluates the heterogeneity model using Cooja simulator by running number of the experiments at various heterogeneity data transfer rates. Chapter 6 summarises the results and gives insights for future work.

2 BACKGROUND

A WSN is a bunch of wireless devices connected together in a certain order to monitor physical or environmental conditions. Each of these devices is connected to one or several other sensors forming a network. Generally, every sensor node comprises of a radio transceiver (to send and receive messages), a microcontroller, sensors and a power source. The size of these devices can vary from the size of dust particle to the size of a notebook. They are constrained in memory, computational power, communication bandwidth and energy budget and are often deployed together in one of the several possible topologies including star, ring, tree or multi-hop mesh network. The propagation of data between the hops of the network can be routing or flooding. We can see a multi-hop mesh organisation of these nodes in figure 2.1.

In this figure, each of these devices have a sensor field in which they can communicate with the devices sharing a common sensor field space. The small circles in the figure 2.1 represent the SNs in multi-hop mesh network. In this topology, the data acquired by a SN is routed to the sink with the help of the neighboring SNs. Finally when the data reaches the BS, it is processed and further sent to an external network. These devices generally adhere to IEEE 802.15.4 Zigbee protocol ([**ieee:802.15.4**]). This protocol is widely used for small low-powered digital radios and therefore find large number of application areas such as industrial Control and Monitoring, environmental and Health Monitoring, home Automation, entertainment and toys, security, location and asset tracking, emergency and disaster Response. Some of the features of this protocol include data rates of 250kb/s(2.4GHz) and 20/40kb/s(868/915MHz), 16 channels in the 2.4GHz ISM band, 10 channels in the 915MHz ISM band and one channel in the European 868MHz band, carrier sense multiple access with a collision avoidance channel access, fully handshaked protocol for transfer reliability, extremely low duty-cycle (< 10ppm) capability, availability of beaconless operation and support for low latency devices.

2.1 STRUCTURE OF WIRELESS SENSOR NETWORK

The seven layer OSI model [**zimmermann1980osi**], is not suitable for implementation in low powered sensor devices. Therefore, in order

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

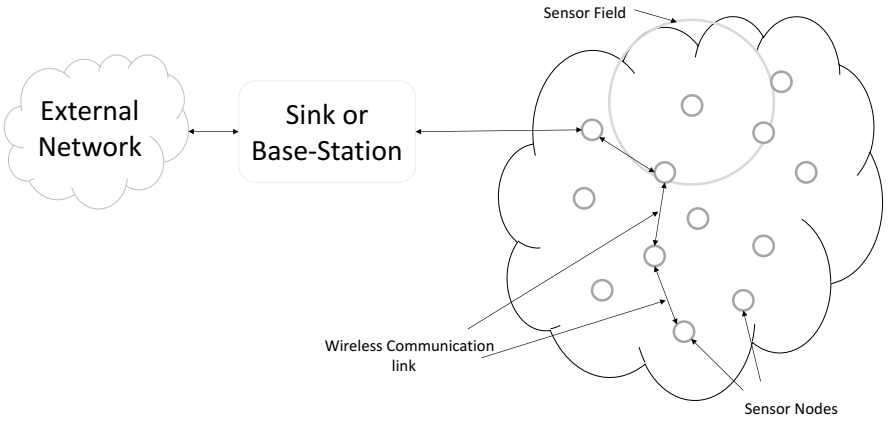

Figure 2.1: WSNs Organisation

to reduce the implementation complexity, the WSN protocol stack is divided into five layers as shown in the figure 2.2.

In this structure, each layer performs a set of tasks which is independent of other layers in the model. The physical layer deals with transferring a stream of bits over physical medium, signal detection and modulation, data encryption and connector cables compatibility with communication medium. The second layer, the data link layer provides services such as medium access control and error control, reliable data delivery, error detection and error correction. The third layer, the network layer is responsible for establishing communication paths between SNs and thereafter transmitting the packets along this path. The path selection phase depends on the one of the possible metrics: shortest path, energy efficiency, reliability and so on. The main tasks of this layer includes power conserving, partial memory, buffers, and self-organisation of SNs (because the SNs do no have universal ids). The protocols for routing layer can be separated into; flat routing and hierarchical routing or can also be separated into time driven, query-driven and event driven. The fourth layer, the transport layer provides transparent and reliable communications between end users. Two of the most widely used transport layer protocols are connection-oriented protocol, Transmission Control Protocol (TCP) and connection-less protocol User Datagram Protocol (UDP). TCP provides reliable communication service and ensures guaranteed data delivery whereas UDP provide an un-reliable service. In WSN domain reliable loss recovery is more energy efficient than TCP based communication. The final layer, the application layer is the mostly used layer for designing aWSN application. It deals with the processing of sensed information, encryption, the formatting and storage of data

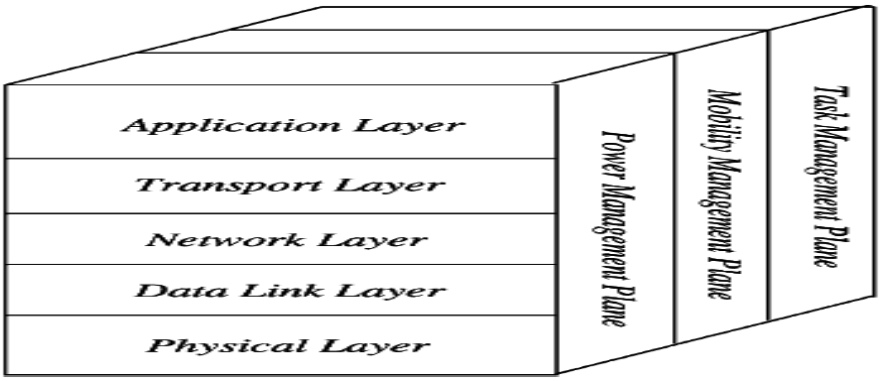

Figure 2.2: WSNs Protocol Stack

and is also responsible for traffic management. It often uses information from other layers to detect if the application can meet the required resources.

The three cross planes in the figure 2.2 are; power management plane, mobility management plane and task management plane. These layers help to manage the network and allow the SNs to work in conjunction with each other so that the overall efficiecy of a WSN increases.

2.2 APPLICATION AREAS

Due to recent advancements in WSNs, it are used in a large number of application scenarios. Some of them are listed below:

1. Disaster relief operations: SNs monitor changes in environment (temperature, humidity and so on) and thus assist in disaster relief operations. They are also deployed to monitor water level in rivers to prevent flood consequences [**cayirci2007sendrom**].
2. Bio-diversity mapping: Observing wildlife [**juang2002energy**]
3. Intelligent buildings: Smart homes which can control the functioning of gadgets based on environmental conditions [**han2010smart**].
4. Medicine and health care: Long-term monitoring of chronically ill patients [**pantelopoulos2010survey**].
5. Agriculture: Monitoring soil and environment conditions to determine the amount of additives like fertilizers, water for better yield [**baggio2005wireless**].

2.3 NESC AND TINYOS

This section will describe about the fundamentals of using nesC programming language ([**website:nesc**]) in TinyOS. In the first subsection we will explain about the structure of a nesC program and in second part we will describe about the abstractions involved in sending and receiving messages using TinyOS. We will also talk about the data structures offered in TinyOS and finally we will describe the limitations of using TinyOS.

nesC

As an extension to C programming language, it defines the structure of TinyOS applications in components. The components are further modularized into interfaces, modules and configurations. Let us go through them one by one:

1. Interfaces:

   The notion of interface in nesC is similar to the concept of *Interfaces* in Java or any other Object-oriented programming language. They define methods that can be implemented by a module. However, a major difference is: only events defined under interfaces must be implemented and commands can be called whenever required. For example Leds Interface provides these methods:

```
1
2             interface Leds {
3
4               // Turn on LED 0.
5               async command void led0On();
6
7               //Turn off LED 0.
8               async command void led0Off();
9
10              /**
11               * Toggle LED 0; if it was off, turn it on, if
                      was on, turn it off.
12               * The color of this LED depends on the
                     platform.
13               */
14            async command void led0Toggle();
15
16            ...
17            ...
```

Now to turn on the Leds on the Telosb, when required, we need to first use the interface called Leds in our application by stating the following:

```
use interface{
    Leds;
}
```

Then we have to call methods provided by Leds interface in our implementation. Leds interface provide methods to toggle(Leds.Ledotoggle()) or turn on (Leds.Ledotoggle()) the Leds on telosb board. Telosb led lights can be turned on by giving the command:

```
call Leds.Led0On();
```

We must note here that interfaces are prefixed with *call* keyword. Also, there are parametrised interfaces to support multiple instances of same interface. The parameters for these multiple instances of interfaces can be small small integers.

2. Modules and Configurations: Having discussed the interfaces concept, we will now look at the overall structure of a nesC program. A nesC program has two components: a file containing a module with its implementation, also known as *Modules* and another file with configuration including the implementation, also called as *Configurations*. Configurations, describe the wiring structure of components. In contrast to Configurations, Modules are implementations. Configurations connect the declarations of different components, while modules define functions and allocate state.

   Both modules and configurations can provide and use interfaces, but the main difference lie in their implementations. *Configuration* implementations refer to how the interfaces used in modules are wired to the actual component that provides it. While *Module* implementations are piece of executable codes that define how the provided interfaces should execute and return or how to use the interfaces to provide a new functionality. This will be clear in following example:

Module

```
1  module BlinkC @safe()
2  {
3      uses interface
           Leds;
4      uses interface
           Boot;
5  }
6  implementation
7  {
8      event void Boot.
           booted()
9      {
10         call Leds.led0
               Toggle();
11     }
12 }
```

Configuration

```
13 configuration
       BlinkAppC
14 {
15 }
16 implementation
17 {
18   components BlinkC,
         LedsC;
19   BlinkC.Leds -> LedsC
         ;
20 }
```

In the above code, the module section uses interface Leds and the implementation section calls *Leds.ledoToggle()* to turn on the Leds on Telosb. Now, in the configuration section we need to instantiate LedsC component and further wire it to Leds interface used in module section so that the TinyOS points to the actual implementation of Leds, provided by LedsC component.

TinyOS

TinyOS ([**website:TINYOS**]) is the de facto standard in the field of WSNs. It provides network protocols, device driver for different sensor platforms, data capturing tools, single-hop networking, ad-hoc routing, timers and many other features, which can be easily used, with suitable modifications, for different application domains. It follows event-driven model which helps to run concurrent applications using small amount of memory. TinyOS uses Event driven models because they are faster than stack threaded design. This is because of two main reasons: 1. They do not require in-advance memory reservation to save execution context. 2. Hardware, is always split-phase rather than blocking. Because of non-blocking operations they perform the operations rapidly. In idle state, it goes in sleep state to save energy.

A First In First Out (FIFO) based TinyOS scheduler schedules operations of components. This is also power efficient because the device can go in low power state when the queue is empty. Components are a collection of *Command Handlers*, *Event Handlers* and *Tasks*. They con-

tain the declarations of commands and events which are provided by the interface it uses or provides. After the declaration in Modules section, the interfaces in an application can be wired to the components which actually implements them. Let us look at these three components one by one:

1. Commands: Commands are non-blocking requests made to low-level components and therefore must return with the exit status whether it was successful or not. Commands can also schedule tasks for later execution. This will involve event handlers which can in turn execute other low level commands.

2. Events: In an embedded system application, we primarily focus on handling events. In general, events are generated or triggered in response to some action that has been initiated by commands. Interfaces contain commands and events, and therefore when we use an interface, we must implement all the events, so that appropriate actions are defined when events are fired.

3. Tasks: Tasks enable components to perform general-purpose "background" processing in an application. They are non preemptive. This means that only one task runs at any time, and TinyOS does not interrupt one task to run another. Once a task starts running, no other task runs until it completes. This means that tasks run atomically with respect to one another. This has the nice property that we don't need to worry about tasks interfering with one another and corrupting each other's data. However, it also means that tasks should usually be reasonably short. If a component has a very long computation to do, it should break it up into multiple tasks. A task can post itself.

To understand these components, we will consider the structure of AMSenderC component:

```
provides {
    interface AMSend;
    interface Packet;
    interface AMPacket;
    interface PacketAcknowledgements as Acks;
}
```

In the above example, if *AMSend* interface is wired to *AMSenderC* then it will be called as implemented in *AMSenderC* component. Now, if we invoke *AMSend.send()* method (provided by *AMSend* interface) with a network packet (which is of type nx_uint8_t or nxuint16_t)

then it will schedule a task to send the packet which in turn will invoke an event handler *AMSend.SendDone()* to acknowledge, if the sending operation was successful or not. Other aspects related to networking design will be covered further in 2.3

There are following limitations in TinyOS:

1. Dynamic memory allocation is not allowed. This prevents run time memory allocation failures and memory fragmentation.
2. Function pointers are also not allowed because TinyOS needs to know the complete execution graph of the program in advance.

TinyOS Networking Architecture

The maximum size of a TinyOS packet is 128 bytes including its headers and Cyclic Redundancy Check (CRC), which also matches the 802.15.4 specifications. An increase in packet size can lead to unpredicted errors in TinyOS. The core of communication abstraction comes from Active Messages (AM), a single-hop unreliable packet. They have a destination address and provide synchronous acknowledgement. The interfaces to send and receive messages in TinyOS are listed below:

- AMSend: This interface is used to send a message by calling the AMSend.Send command (provided by AMSend interface). On completion of successful or unsuccessful sending SendDone event is signalled. In general WSN application development scenario, we also using this event as an indicator to send the next packet.
- AMReceive: On reception of a message sent via AMSend interface, AMReceive.receive event of the corresponding AMSender id is signalled.
- AMSnoopingReceiver: This interface signals event AMSnoopingReceiver.receieve even when the packet is not addressed to a particular SN. This allows us to snoop packets in the device communication range.

To send data, we need to instantiate a member of AMSend with an am_id_t type, which is a small integer, and further wire it to AMSend interface. We then need to call AMSend.send() method from the module implementation, which will trigger sending of Active Message (AM) packet. The packet can either be sent to all the sensor nodes which is done by broadcasting at AM_BROADCAST_ADDR (a constant defined as oxFFFF and is reserved in TinyOS for broadcasting

purposes) or to a specified TOS_NODE_ID (every SN has a specific address which can be obtained by calling TOS_NODE_ID constant).

After a SN receives an AM, it calls events that are responsible to receive the messages of the defined *am_id_t* type of Receive Component. This concept of multiple instances of same Component, gives us the flexibility to have multiple instances of Receivers (abstracted as *AMReceiverC*). With multiple AM ids of AMReceiverC, the device will only call the event associated with the matching receiver id. This concept is predominantly used in TinyOS to distinguish messages of different types and we will also use this in implementation of the concept *heterogeneity* to distinguish one hop and two hop messages received.

2.4 ROUTE FINDING AND SELECTION

Route discovery, route selection and route representation are key requirements for a wireless multi-hop routing problem. Therefore, it becomes quite important to decide appropriate protocols for finding a solution to multi-hop routing scenarios. We will cover the fundamentals to find a solution to the multi-hop routing problem by sequentially enumerating through the mentioned phases, as shown in the figure 2.3:

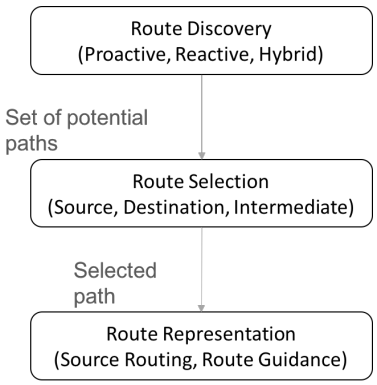

Figure 2.3: Route Finding Phases

1. Route Discovery: Three ways exist to find a route from source to destination:

   a) Proactive: Protocols like Distance Vector Routing (DVR) ([**RFC1058**]) and Optimal Link State Routing (OLSR) ([**RFC3626**]) use pro-active techniques to find a route to the destination. They maintain routing tables to store information about how to reach the destination. Although this technique is

quite good for high traffic but it has huge memory requirement to store tables for the entire network.

b) Reactive: Protocols in this category, e.g. Ad-hoc On Demand Distance Vector Routing Protocol (AODV) ([**RFC3561**]) and Dynamic Source Routing (DSR) ([**RFC4728**]), perform on-demand route discovery by sending a Route Request (RREQ), containing the desired destination address for data sending, to neighbors and receive a Route Reply (RREP) from the destination. Route discovery can be done in two pays: Firstly, doing it on fly. Under this technique route is searched whenever required. Secondly, it can be done hop-by-hop. In this technique each SN decides the next suitable neighbor to forward the packet. SNs accomplish this task by exchanging periodic routing updates through beacons containing necessary information for neighborhood discovery.

c) Hybrid: Protocols combining feature from both, proactive and reactive protocols.

2. Route Selection:

   For proactive protocols, route discovery phase is sufficient for route selection. Periodically updated routing tables provide the intended nodes with the information about how to reach destination nodes. However, this is not the case with reactive protocols. The selection phase can be handled by either of the three terminals: source, destination or the intermediate nodes.

   The decision to make source or destination based route selection can depend on the application requirements. Hop count, end-to-end delay/jitter, interference level, packet loss rate, link residual capacity, load balancing and so on can be among the several possible metrics to select the net hop route. In general, number of hop counts to destination is often the preferred metric as it is easy to compute and also it provides us the shortest path to destination. For intermediate nodes, the route selection is done by every node on way to discover the best next hop.

3. Route Representation:

   Two approaches exist to achieve route representation: Either storing the exact route via routing tables/source routing or provide a route guidance. To implement the source routing approach, we need to include the exact route from the source to destination in the RREP packet header. The sender then forwards the packet to the next recipient specified in the packet header. This process continues until the packet reaches the destination.

In contrast to source routing, the route guidance method involves intermediate nodes to find the best next hop.

In order to make the routing more efficient and to save energy by avoiding route re-computation to destination every now and then in heterogeneity, we will implement the idea of source routing in data transmission phase. In this mechanism we will let the sender know about the exact route to destination via which the data transmission phase should take place. This is discussed in more detail in chapter 4.

2.5 DATA COLLECTION

Data collection is primarily used to collect sensor data captured by different SNs in a WSN. The nodes collect information from their surroundings and forward this information to BS. Data collected from deployed sensors can be further evaluated or analysed at the sink node or BS. The fundamental idea is: generation of data at multiple nodes and collection at one node (called Root node or BS). This pattern of flow of data also appears to be converging at one point. Efficient collection and reliable sending is a major challenge in the converging flow of the data. This requires quick adaptation to changes in the network without frequent beacon exchanges (because these exchanges consume considerable amount of energy). CTP authors ([**TEP:119**]) have claimed to solve the issues like reliability, robustness and high data delivery ratios.

Most trivial way to realise this form of data collection is via tree establishment by following the below mentioned points:

- Root nodes advertise themselves with hop count value 0 to the neighbors.
- Other SNs collect information like hop distance, node id and more from their neighborhood.
- Each SN selects one of its neighbor as parent based on certain metrics (for examples refer to 2.5).
- The parent nodes handle and forward the packets received from their children.

However there are following issues which also need to be taken care while establishing a tree.

- Uni-directional links: A SN determines the closest parent as the one which can only send packets but can not receive them.

- If a node accidentally receives one beacon from a far away node
- Mobile nodes: Node selected as parent is not in range because of it's mobile nature

Therefore, while selecting a parent we also include parameters to estimate link quality. Some of the ways to measure link quality are mentioned below:

1. Packet reception rate (PRR): Send a predefined number of packets and measure what fraction of these are received at regular intervals.
2. Received Signal Strength Indicator (RSSI): Analyze received packets with regard to their signal strength (serving as an indicator for the node distance).
3. Link Quality Indicator (LQI): Correlation between bits in the Start Frame Delimiter (SFD) to indicate potential channel issues (reflection, scattering, etc).
4. Expected number of transmissions (ETX): Estimate how many re-transmissions are needed on average when using a particular link.

2.5.1 *Collection Tree Protocol*

CTP is a DVR protocol. DVR is based on the idea of manipulating the vector distances to other SNs in the WSN. Under this routing methodology, each SN maintains a routing table consisting of following two information to reach each of the possible final destination SNs.

1. next node: Direction in which packet should be forwarded
2. cost: Hop counts from the destination

Every entry in the next node column of the routing table must be an adjacent SN, so that the intended sender can directly forward it's packets to this adjacent member. The table is maintained by periodic exchange of the vector consisting of the adjacent node routing table's cost column. Every SNs updates it's cost column by keeping the minimum of cost value it knows previously and the recently arrived cost vector. To avoid loops in the routing protocol, CTP uses datapath validation technique. This technique gets rid of the routing loops by triggering routing updates when it encounters a data packet to be forwarded has lower ETX value than it's current ETX value or vice-versa or the ETX value of a node drops significantly (in CTP this threshold is

1.5). CTP also uses adaptive beaconing mechanism to minimise power consumption and update stale information. The periodic timer to update stale information goes on increasing exponentially but the timer is reset as soon as the route update request is received.

2.5.2 *Implementation*

In CTP, ETX metric is used to establish a tree topology for data collection. Every node broadcasts beacons indicating its ETX to the sink. Broadcast intervals are exponentially increasing (up to 512 seconds), in static network conditions. Further, nodes select their parent based on the ETX metric. Nodes compute their ETX using the formula:

$$\mathrm{ETX}_{\mathrm{ToSink}} = \mathrm{ETX}_{\mathrm{Parent}} + \mathrm{ETX}_{\mathrm{Node-ParentLink}}$$

In this protocol parent can also be updated during run time when link with better ETX becomes available. A SN participating in CTP collection, sends its data as uni-cast messages with link-layer acknowledgments enabled. TinyOS developers have included CTP implementation in their operating system libraries. This can be found at **[tinyOS:ctpImplementation]**. In the following section we will describe the main role taken by the software components of CTP framework as shown in figure 2.4.

1. Link estimation: This component provides the ETX information for one hop communication between two SNs. This value is updated based on the adaptive beaconing strategy as mentioned in the above section. It also combines information from physical, data link, and network layers to provide accurate link quality estimates. For every 5 transmissions by the ETX estimator, it produces an ETX value of 6, if the number of successful transmissions is 0. The implementation for this module can be found in **[ctp:linkEstimation]**.

2. CtpRoutingEngineP: This component provides methods to select the next hop for data transmission based on the ETX values obtained from Link Estimator. It also stores the minimum cost route to the root node. A root node advertises it's ETX as zero and the ETX of a SN is calculated by adding the ETX of it's parent node plus the ETX of the link to it's parent node.

   CTP beacons are sent at specific intervals determined by Trickle algorithm. The algorithm allows to gradually reduce the beacon sending rate which further helps in saving energy and bandwidth. However, data path validations can reset this interval to

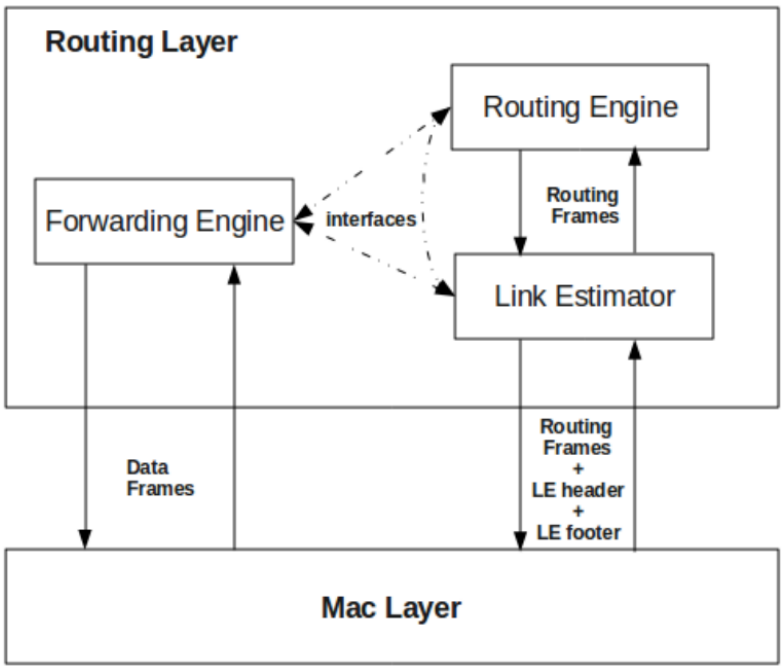

Figure 2.4: Message flow and modules interactions, from [**colesanti2010performance**]

make the SNs react to topological and environmental changes quickly.

3. CtpForwardingEngineP: It is responsible for forwarding the packets received from child nodes as well as the data generated by the SN itself. It also attempts retransmissions when necessary (in worst case up to 32 times). It also informs about the routing inconsistencies like routing loops to routing engine.

CTP for Heterogeneity

We have re-used existing metrics provided by CTP for finding solution to best next hop discovery. This can be done by tweaking the routing engine component in CTP (which establishes a collection tree to collect data at sink). Moreover, the information provided by CTP is a reliable solution to find neighbors and estimate routes because the protocol considers information from link layer and routing layer to estimate the link quality. We will discuss in detail about the modifications and usage of CTP in heterogeneity in chapter 4.

3 LITERATURE REVIEW

Several works have been proposed in the direction of extending resource power and achieving higher average energy budget of the sensor nodes across the SN. Most of the solutions are scenario specific and do not give good results under altered conditions and assumptions. Pottie and Kaiser [**pottie2000wireless**] in their paper have stated that transmissions of about 100 metres range is equivalent to 3000 instructions. Therefore, we need protocols for data aggregation or local processing to minimise energy expenses in performing long range transmissions. Local processing of data can be based on two approaches: cluster based mechanism or addition of heterogeneous processing nodes in the network. In this section we will review the existing protocols in these fields and motivate our research work.

3.1 CLUSTER-BASED

The concept of clustering to solve the problem of local data processing requires the selected leaders or cluster heads to cater for the communication and processing overhead. Based on the this concept, the paper LEACH [**Heinzelman:2000**], was proposed. Though, this algorithm uses randomized cluster head rotation to address the problem of evenly distributing energy load among sensor nodes, yet there are problems in cluster election and cluster formation phase. Also, there is a elevated energy consumption between cluster heads and BSs as the cluster heads are required to communicate directly with BS. In order to directly communicate with the BS, the cluster heads will have to send many packets using their high transmission power. Therefore amount of energy dissipated in performing high power transmission will still be a major concern even if we address the problem of fairly assigning slots to the available senders. Improvements to the cluster based algorithms were proposed. LEACH-C [**Heinzelman:2002**], which is one of the proposed improvements to LEACH, improves the cluster formation by using a centralized scheme to distribute cluster heads through out the network. More advanced version of LEACH-C was proposed by Zhao in paper [**zhao2004energy**]. Their algorithm does not require centralised allocation of cluster heads by BS and distributes cluster heads more uniformly across the network.

Projects on hybrid algorithms for clustering like in a paper by Jung [**Jung:2009**] takes into account several factors like residual energy

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

and distance for adaptive static and dynamic clustering formation for high and low data traffic rates respectively. Many authors like Ye, in the paper EECS [**Ye:2005**] argue distributed interaction, load-balanced clustering and low-control overhead for data collection. In this mechanism, the larger the distance between cluster head and base station, the smaller member size the cluster head should accommodate to compensate for the penalty of long range transmission. Chain based algorithms like PEGASIS [**Lindsey:2002**] and CHIRON [**Chen:2009**] improve the energy efficiency of the network by allowing each node to play the role of head node in turn but still transmission to distant nodes in long chain remains a problem. Since there can be only one node and multiple transmissions are not possible, latency remains an issue for chain-based algorithms. Therefore, it can be concluded that clustering and chaining algorithms can not be used to address the problem significantly.

3.2 HETEROGENEITY

Adapting heterogeneity to a routing protocol is another way to approach the problem of local data processing with minimal energy consumption. In the paper authored by Sharma and Mazumdar [**Sharma:2005**], the focus is on establishing heterogeneity by using wired connections between certain nodes to reduce the overall energy consumption. However, this idea limits the usability of a network for long-term scenarios. in an another work using the Mica2 motes and stargate devices, Hu et al. [**hu2009design**] have built a hybrid network for detecting cane toads in northern Australia. They have proposed to build a WSN with low-power motes with higher processing capabilities. This again puts a limitation to the idea as this would lead to faster draining up of energy because resources consume energy, even if they are not being used. Therefore, they can not be used for autonomous deployments.

Use of mobile agents to carry the code and state information from one device to another has also been studied to address heterogeneity issue. In a paper titled AFME [**muldoon2008agent**] and in a similar paper titled: MAPS [**Aiello:2011**], authors have explored in the direction of using mobile agents. However, the object-oriented designs of mobile agents are quite slow and extremely difficult to implement.

To increase the network lifetime, Rhee and et. al. [**Rhee:2004**], have proposed the use of functional device heterogeneity. The paper is based on the idea of Endpoints, nodes which can not relay data on behalf of others and rather act only as destination for the communication, and routers, nodes that operate with higher duty cycles as they can relay data on behalf of others. In the paper by Yu [**Yu:2007**],

researchers have supported the idea of optimizing the number and location of processing nodes, which lies more in the domain of a Mathematical Optimization.

From the theoretical energy analysis, Reinhardt and et. al. in their paper [**reinhardt2013exploiting**], have concluded that heterogeneous SNs are beneficial to deploy in a network with three major application scenarios: cryptography, compression and high-data rate processing within the WSN. From the evaluation, they have come to a conclusion that: "Energy savings can be achieved by deploying processor nodes, as their greater energy consumption is counterbalanced by reduced execution times and less traffic in the network."

In a similar work by Reinhardt and et. al. [**reinhardt2008designing**], they have argued that platform heterogeneity with task migration concept can save the overall energy budget of a WSN. They have reasoned this with task migration concept which can play a part in the WSN when the energy demand for transmission of data plus remote processing is less than local processing of data plus estimated energy demand of processing and reception of data.

Our paper focuses on abridging the research gap on simplified deployment of heterogeneity in a WSN which is based on the idea of energy savings as proposed in paper [**reinhardt2013exploiting**] by minimising the number of transmissions required to deliver the data to the BS. Our aim is to address the drawbacks and limitations of heterogeneity concept and further propose a simplified mechanism to provide the heterogeneity layer as an additional plug in on top of the selected routing protocol. Since heterogeneity layer has to depend on other routing algorithms to get the data delivered to BS, we simulate heterogeneity on top of CTP algorithm, which is considered robust, reliable and efficient for diverse number of platforms. In the paper [**pecho2010simulation**], the authors have concluded that CTP is designed for low data rates. Our work on heterogeneity also aims to extend CTP for high data transfer rates. therefore, the heterogeneity layer should also be seen as an extension to increase the overall flow of wireless data traffic at minimal energy expenditure.

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

4 IMPLEMENTATION OF HETEROGENEOUS WSN

The SNs in the heterogeneity network model are not centrally operated to avoid single point of failure. Therefore they act independently to co-ordinate and communicate with their neighbors to resolve contentions among the SNs for heterogeneity layer access. This approach requires frequent message exchanges among the nodes. Therefore, we need an efficient routing methodology to minimise the beacon exchange count and further initiate the data transfer via heterogeneity in a contention free slot.

In order to realise the benefits of deploying heterogeneity in a network, we simulate heterogeneity along with CTP algorithm. This chapter describes the simulation methodology and programming structure of heterogeneity concept. In the first section, Platforms used 4.1, we describe the hardware and software components used for simulating the network model. In the second section, Design Components 4.2, we provide an overall picture of heterogeneity model and further describe the TinyOS components and interfaces involved in the design phase. This also explains the importance of CTPs module for our routing model. In the third section, Control Plane Design 4.3, a detailed analysis of the routing mechanism in heterogeneity is provided. It describes three aspects of heterogeneity: 1. how the protocol finds the route to destination heterogeneous node 2. the criterion to decide one of the possible routes found by route discovery phase 3. how the final selected route is represented in the heterogeneity model. As a follow up to the routing model, the subsection, Routing Implementation, conforms the routing model requirements with heterogeneity model. This subsection also contains the packet contents of message exchanges in heterogeneity. The packets in our network model serve as an important aspect to realise information exchange among SNs. In the fourth section, Heterogeneity Specific Implementations 4.4, we provide a detailed explanation of data structures involved in running the heterogeneity model. The explanation focuses on two major aspects: 1. how these data structures are linked to each other and 2. how their interdependency act as an important tool to regulate state change in heterogeneity. In the fifth section, Data Plane Design 4.5, data transmission in one hop and two hop neighborhood through heterogeneity is explained. In the final section 4.6, we present the flow diagrams to sketch the life cycle of heterogeneity model.

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

4.1 PLATFORMS USED

This section is divided in two parts: hardware components and software components.

4.1.1 *Hardware Components*

The hardware used for implementation are Telosb motes [**datasheet:Telosb**]. This sensor platform was originally developed by the University of California, Berkeley by TinyOS developers. It features the 8MHz Texas Instrument MSP430 (the MSP430F1611) microcontroller with a 10 kBytes internal RAM and a 48 kBytes program Flash memory, IEEE 802.15.4 CC2420 radio chip, data transfer rate up to 250kbps, integrated onboard antenna, 1MB external flash for data logging, programming and data collection via USB, sensor suite including integrated light, temperature and humidity sensor and runs on TinyOS 1.1.10 or higher.

MSP430 Microcontroller

The MSP430 [**website-MSP430**] is a low-cost microcontroller generally used for low powered embedded devices. It has a 16-bit Reduced Instruction Set Computing (RISC) CPU with an instruction cycle time of 125nS. The device is highly optimised for low energy consumption and high code efficiency. It supports six different low-power modes to disable unnecessary running of clocks and CPU. It is capable of wake-up times below 1
muS, which allows the microcontroller to stay in low power mode for longer period of time and thus maximising the available energy budget. The power consumption in active mode is about 330µA at 1MHz with 2.2V. In idle mode the microcontroller needs less than 1µA. The supply voltage should be in the range of 1.8V to 3.6V.

CC2420 Radio

It is a 2.4GHz IEEE 802.15.4 complaint RF transceiver [**TI:cc2420**]. It is specially designed for low power embedded devices. Some of the important features include: extensive hardware support for packet handling, data buffering, burst transmissions, data encryption, data authentication, clear channel assessment, link quality indication and packet timing information.

4.1.2 *Software Components*

We have compiled our heterogeneity code in TinyOS which uses nesC programming language. In addition to this, we have also used COOJA simulator [**cooja:Contiki**] to simulate the compiled code in a virtual WSN environment. Cooja is a java based simulator for emulating SNs of certain platforms at hardware level. This allows faster and precise inspection of the model behaviour for the compiled TinyOS code.

4.2 DESIGN COMPONENTS

In the first subsection, we will discuss heterogeneity implementation possibilities with a focus on energy and computational power. In the second part, we will briefly explain about the working of heterogeneity layer and in the final subsection, we will talk about the general TinyOS components required for implementing heterogeneity.

4.2.1 *Heterogeneity Possibilities*

From the literature survey in chapter 3, we have explored several possible dimensions of heterogeneity implementation including increased energy, computational power, memory, wireless range, etc. for a particular group of SNs. With CTPs being the most widely used protocol, we implement it as the baseline for collecting data generated at a SNs and for the sake of simplicity, we rely on the parameters energy and computational complexity throughout the remainder of the report. Also, we depend on these parameters to perform a comparative analysis of heterogeneity with CTPs.

4.2.2 *Outline*

The heterogeneity set up requires CTP implementation (provided in [**tinyOS:ctpImplementation**]) at back end. CTP is responsible for building and maintaining minimum cost trees to nodes which advertise themselves as roots based on ETX parameter. Messages are sent and received via CTP send and receive interfaces. On top of the CTP layer, we implement the heterogeneity layer. The tasks performed by this layer are sequential and event-driven. SNs respond to different events and thus change their states on subsequent events reception or triggering. We will discuss more about states and flow diagrams in section 4.6. These states are stored as global variables or well-defined structs in the implementation. In summary, the WSNs operates in an endless loop in the following order:

1. Beacon based PC announcement in one and two hop neighborhood: Each PC participates in the periodic advertisement of their computational power availability to their one and two hop neighbors.

2. Selective Request To Send (RTS) response by SN to the beacon announcement: The notion of RTS concept used in heterogeneity layer is equivalent to sending a request packet to check the PC availability before the actual transmission of data. The packet requests a unique time slot for the intended sender and thus resolves the conflict of multiple data senders to one PC.

3. Clear To Send (CTS) response by PC: Based on the earliest RTS request, a CTS response is generated by the PC for the intended sender. The notion of CTS response implies that the PC will have to send a clear to send signal before the intended SN can initiate the data transmission. Until the intended sender gets a CTS response, it participates in CTP collection for sending it's data.

4. Data transfer via heterogeneity layer: On reception of CTS response by the intended sender, the data transmission phase starts for the time period the intended sender requests.

5. Data collection via CTP from PC to BS: On completion of data sending by a SN to PC, the final computed data is added to the data collection queue and thus it reaches the BS via CTP.

We will have a closer look at the detailed implementation of the heterogeneity set up in the coming sections and subsections.

4.2.3 *TinyOS Components*

In this subsection, we will discuss about the interfaces and wiring required for implementing heterogeneity.

1. General Interfaces: Following are the commonly used interfaces required for the implementation:

   - Boot: It is required to boot a SN. the boot interface signals Boot.booted() event on successful booting of the device.
   - SplitControl: It is used to start and stop services of the radio transceiver. It is wired to ActiveMessageC component.
   - StdControl: This interface is provided by Collection protocol. StdControl interface controls the state of CTP routing by setting/resetting the routing state once the routing layer of CTP starts or stops respectively.

- Leds: This is used for controlling the led lights on the Telosb. We use light patterns to indicate transmissions and receptions in our application. The interface is wired to ledsC component and provides methods such as LedsToggle, LedsOn or LedsOff for the leds.

2. CTP-based Interfaces: These interfaces are provided by CTP to send and receive messages via the collection tree.

   - RootControl: This interface is used to advertise a particular SN as a root node of the CTP tree.
   - Send: It is used to send messages via CTP protocol through the collection tree. This interface is wired to CollectionSenderC component provided by CTP.
   - Receive: It is used to receive messages sent via Send interface in 2. The messages received via this interface are exclusively sent via CTP protocol. Messages sent via other sending interfaces, which are not wired to CollectionSenderC, do not get routed via CTP. This distinction helps in implementing an abstraction layer over CTP on which we will build our heterogeneity model and thus keep our data exchange model separate from CTP communications.
   - CtpInfo: This interface provides methods to obtain ETX information about the current SN. It is also wired to CollectionC component of CTP so that we could obtain the ETX metric used in setting up the collection tree. This metric is also updated periodically by CTP and therefore keeps the heterogeneity model remians up to date with changes in ETX metric.
   - Timer: This timer is fired to perform routing via CTP. Only when this timer is fired, the SN announces itself as a part of CTP and further acknowledges itself as a member of CTP tree.

4.3 CONTROL PLANE DESIGN

This section describes the advertisement phase of a PC and PC selection phase of an intended sender. We will first describe the routing model and then explain the routing implementation phase of our design.

4.3.1 *Routing Model*

In this subsection, we will look at the route discovery, route selection, route representation and data forwarding phases of heterogeneity.

1. Route Discovery: To derive the route discovery mechanism implemented in our design, we will continue from the previous description on route discovery in section 1 (in chapter 2).

   In our heterogeneity model, we perform the route discovery phase by maintaining an array of known PCs in at most two hop neighborhood. Therefore, We can say that the route discovery is done proactively because we know exact route from sender to PC in advance. A sorting algorithm periodically sorts this array for the optimal selection of PC at constant run time ($O(1)$) (explained in detail in sections 2 and 4.4). Further, to ensure collision free data transmission, a RTS packet is sent to the first member of the PC array to verify if the recipient is free.

   The PC reply packet carries routing information to stimulate data transmission by sender. We can utilize the concept of message pools to avoid the PC route reply packet transmission. The concept of message pool can be exploited in two ways: either we maintain separate queue for each of the possible senders or accept data from all possible senders in one queue at any point of time. Although these methods solve the problem of sender fairness, yet these pools can be memory intensive in high traffic scenario. Also, to implement the latter scenario, we will first have to sort similar data in the message pool and then process them in ordered way. One more drawback of pooling approach is frequent back-offs while sending stream of data because multiple nodes will try to to get channel access while sending to one PC and hence would not get clear channel assessment very often. This can finally lead to low data delivery ratio. Therefore, instead of this pooling approach, we have chosen to reserve the processing center for a particular SN and only after the reservation confirmation, the intended sender can send their data for processing. We also assume here that each of the SN can send only one type of data for the allotted reservation period. A PC confirms its own reservation via RREP packet to the sender on first come first serve basis. It waits for a certain amount of time to receive data. In case of no reception after passage of allotted time period, it again starts flagging itself as an available PC to the SNs in one or two hop neighborhood.

2. Route Selection: In the previous discussion on route selection in 2 (in chapter 2), we have explained the possible ways of route

selection. Now, we will discuss how we achieve this in our heterogeneity design.

We use CTP to get ETX information of the SN in order to select the best PC for data transmission. As claimed by the authors, CTP maintains an updated ETX estimate of SNs on periodic basis. We use these updated ETX values for PC selection. To extract the ETX values, the interface 'CtpInfo.nc' is wired to CTPRoutingEngineP component, which is also provided in the TinyOS CTP implementation. This interface provides following useful methods for neighborhood discovery and route selection:

a) getParent() -> Gets the parent of the node in the tree.

b) getEtx() -> Gets the ETX for the current path to the root through the current parent.

c) triggerRouteUpdate() -> Informs the routing engine that sending a beacon soon is advisable.

d) numNeighbors() -> Gets number of neighbours.

e) getNeighborLinkQuality() - Gets the Link Quality of a neighbor.

f) getNeighborAddr() -> Gets the address of the neighbor.

For the route selection process, we propagate the ETX information, obtained by **getEtx** method, during the beacon broadcasting phase. This value is then used through out in our design and is the building block of optimal PC selection. We therefore believe that re-using and keeping an updated copy of this information in the network enhances the network reliability.

We have also added an extra method in CTPInfo interface to acquire addresses of all the SNs in one hop neighborhood. This was done with the help of **getNeighborAddr** method. This method returns the address of all the SNs from it's routing table. As mentioned in previous section on route discovery, each SN in CTP maintains a routing table of size number of neighbors containing neighbor address and other relevant fields. We therefore exploit this routing table information by iterating through all the routing tables for the number of neighbors in one hop neighborhood (provided by CtpInfo numNeighbors method). Further, we store all the neighbor addresses in an array and return it. Although this method is currently not used in the model, but it can be quite useful in improving the network reliability and efficiency by precisely determining the SN density around each of the SNs and further placing a PC around it.

3. Route Representation: We have implemented the idea of route guidance in search phase. However, to make it more efficient, we have blended in the idea of source routing in data transmission phase. This approach will inform the sender about the exact route to destination through which the data transmission phase should take place. In this way we will not waste energy in re-computing the route to destination every now and then. There is also an added benefit to this approach: we do not need to maintain long routing tables as we need only the routing information from the SN to the PC in one or two hop neighborhood. We therefore do not add routing information for PCs located at a distance more than two hop and hence avoid maintaining long routing tables and thus leave off the distant PCs.

4. Auxiliary Components: Apart from the above three routing components, we have also looked into the following aspects:

   a) Route Maintenance
   b) Route Refreshing
   c) Route Failure Handling
   d) Route Invalidation
   e) Restricted Flooding
   f) Data Aggregation

   The first four auxiliary components have already been taken care of by re-using the information from CTP protocol because the protocol sends beacons every 5ms to update stale route information, if any, in the WSN. Also, we have restricted flooding to two hop neighborhood in the route discovery phase. In other phases, we perform unicast transmissions. Moreover, the concept of heterogeneity relies on the notion of data aggregation technique as we send only same type of data to the processing center. This reduces the number of packets flowing through the network which in turn would reduce the overall energy budget of the SNs.

4.3.2 *Routing Implementation*

The routing implementation is summarised in following points:

1. As discussed before, route discovery phase begins with beacon advertisement of PCs. Each node maintains a local copy of the advertising PCs and use it for route computation phase. Table 4.1 shows the packet contents of beacon advertisement message.

Table 4.1: Beacon Advertisement

Field names:	PC ID	PC ETX	Relayer ETX
TinyOS Data Type:	nx_uint8_t	nx_uint16_t	nx_uint16_t
Size(In bytes):	8	16	16

2. In the next step, a route is selected based on ETX values propagated during beacon advertisement.

3. Based on PC availability at different moments, we can expect PC availability issues. Therefore, we can not rely on static routing table and rather need to determine if the processing centre is available before data transmission. If no PC is found free, the data transmission takes place via CTP. For determining the availability, the sender sends a RREQ to PC and in response the PC makes a RREP to the sender if it finds itself free. Table 4.3 and shows the contents of RTS and CTS packets respectively.

Table 4.2: RTS Request Packet

Field names:	Duration	Sender Address	Relayer Address	PC Address
TinyOS Data Type:	nx_uint8_t	nx_uint8_t	nx_uint8_t	nx_uint8_t
Size(In bytes):	8	8	8	8

Table 4.3: CTS Response Packet

Field names:	Duration	Sender Address	Relayer Address	PC Address
TinyOS Data Type:	nx_uint8_t	nx_uint8_t	nx_uint8_t	nx_uint8_t
Size(In bytes):	8	8	8	8

4.4 HETEROGENEITY SPECIFIC IMPLEMENTATIONS

In the following subsections, we will describe heterogeneity specific design data structure and further point out two important elements of this design: Periodic Member Updating and Periodic Sorting.

4.4.1 *Data Structures*

1. Enums: They are used as constants for defining timer intervals, debug parameters, ids of the PCs and BS, random data transfer duration selection for each SN and many other parameters used in the simulation.

2. Queues: TinyOS Queue interface is equivalent to a Queue data structure. Important commands in this structure include commands to en-queue an element, de-queue an element and retrieve queue size. In heterogeneity, we have implemented following four queues:

   a) CTSQueue: This queue stores RTS requests from the intended data senders to the PC. This has been defined as a queue to store multiple RTS requests to the PC and further send out the CTS response to one of them based on a specific criterion. Although in the current implementation we have sent out the CTS response on first come first serve basis, and thus the idea to store multiple RTS requests may not seem much useful but for future testing and comparative studies, we can choose an optimisation algorithm to send out the CTS response based on parameters like: fairness, ETX or other LQI. This has been done to make the heterogeneity layer extensible to other WSN application domains.

   b) DataStoreQueue: This queue holds the data transfer information of the SNs which are participating in data transfer phase of heterogeneity. Every member of this Queue is a struct of PC id, Relayer id, Sender id, and the data transfer duration approved by PC. As soon as the recipient confirms its id with the CTS response, it en-queues itself in the queue and calls data transmission phase to initiate the data sending operation.

   c) DataGenerationQueue: This queue stores data for transmitting it through collection tree formed via CTP. Principally, each SN generates data on a periodic basis, which has to be de-queued and further sent out via CTP. However, if a SN is granted CTS permission to send data to the PC, the node stops sending it's data via CTP and instead de-queues data from the DataGenerationQueue and forwards them to the PC via heterogeneity layer.

   d) CTPCollectionDataQueue: This queue holds the processed data collected from the heterogeneity layer. On receiving the data termination beacon from the sender node, the PC processes the incoming data and enqueues it in this queue. Further it gets collected via CTP.

3. Structures: Following structures are used in the application:

   a) beaconBroadcastMsg: PCs broadcast out their availability to at most 2hop neighbors. This message is forwarded as

a network packet containing ETX and id of PC. Later the relayer adds its own ETX to the packet, when the message is finally forwarded out to two hop neighbors of PC.

b) processingCentre: This array of structure holds the information of the processing centers in it's 1hop or 2hop neighborhood as an individual member. The information includes ids and ETX of PC and relayer. This information is updated after a SN receives broadcasting beacons from the PC in one or two hop neighborhood. If the Process centre is in the immediate one hop then the relayer is by default set as 0. SNs scan this array to find out the most suitable processing centre in it's neighborhood for sending RTS request. This scanning also filters out the nodes indicated as busy in busySensorNode array (3d).

c) PreambleMsg: This is sent out as a network packet as RTS in response to PC availability indicated by beaconBroadcastMsg broadcasted by PC.

d) busySensorNode: Every node maintains a copy of busy SNs in it's neighborhood. This array of struct holds, ids of busy PCs, relayers, senders and time period allotted for the intended sender to transfer data, as an individual member of the struct array. This information is kept up-to date via a periodic timer which keeps on decrementing the time period allotted at every firing interval. After the allotted time period reduces to zero, the structure member is remove from the array of structs.

e) dataSendStruct: In the data transmission phase, the sender sends out the data and packet counter (for debugging purposes) to the PC which has sent the sent a CTS response.

4.4.2 *Periodic Member Updating*

We rely on periodic CTP control beacons to update the stale ETX information for each of the member node. At heterogeneity layer, we update 3b on periodical beacon announcements by PCs. These beacons indicate the availability of PC in at most 2hop neighborhood. We also understand that this periodic update is not necessary so often if the PC is already included in the array. Therefore we also put a minimum number of beacons received on updating a stale member of the struct. This approach resolves the trade-off between accurate data availability and low cost. The heterogeneity layer updates the 3b on two occasions:

1. It receives a beacon from an unknown PC. As a result of the reception, the new member is appended in the array.
2. It receives a beacon from a known PC. In this case, the member is only updated after the enumerate E_MEMBER_UPDATE_PERIOD reaches it's maximum count.

4.4.3 *Periodic Sorting*

Periodically, the members in 3b are sorted in the descending order of priority. The sorting depends on the following parameters:

1. Lowest ETX value within 1hop neighborhood (the relayer value is zero)
2. Lowest ETX value within 2hop neighborhood (the relayer value is non-zero)

This sorting helps in selecting the best PC in $O(1)$ run time. This sorting is useful in situations where there are short data transfer slots and frequent re-assignments of PCs to the SNs. However, in situations where we have longer data transfer durations implying sporadic re-assignments of PCs, it is worth removing the timer and instead calling this sorting method on-demand (when RTS has to be sent to PC).

4.5 DATA PLANE DESIGN

On reception of CTS packets from the PC, DataSendingTimer is fired to trigger the data sending phase for intended sender. During this period, the PC marks itself as busy, stops advertising itself as PC in two hop neighborhood and does not further respond to any RTS requests. DataSendingtTimer sends out three beacons preceding the actual data transmission to indicate the beginning of data transmission phase and at the end of data transmission phase it again sends out three beacons to indicate the termination of data transmission phase. This process is necessary in the data transmission implementation to precisely determine the actual start and end time of data transmission phase. On successful reception of at least one of the termination beacons, other received termination beacons are simply ignored. We also need to send out more than one beacon to let the PC know that data sending has been terminated because the PC has to precisely know when it has to en-queue the computed data in I_CTPCollectionDataQueue. Also sending out only one beacon has more sending or receiving failure probability due to multiple channel back-offs or receiving event errors at PC. Therefore, this failure can discard the entire data which

was received but not processed and hence not en-queued for collection via CTP after the data transmission had completed.

This will be discussed in more detail in following two subsections. The first part covers the case for data sending in one hop neighborhood and the second part covers the data sending operation via a relayer to a two hop PC.

4.5.1 *One-hop Data Transmission*

Data sending in one hop is straight-forward. We use AMSend interface to send data to the PC. In this case the PC is set as the destination address in the *AMSend.send* method. The relayer address is set to zero because we do not need a relaying point to forward our data. The sender sends data via a specific instance of AMSend interface (instantiated via an integer am_id_t) and the PC receives them via AMReceieve event parametrised to same am_id_t as that of sender. This transfer takes place for the duration of transfer requested by intended sender minus a small safe margin value to compensate RTS and CTS beacon exchange period. Finally, on reception of termination beacons, the PC does some complex computation on the received data and en-queues it in I_CTPCollectionDataQueue for data collection via CTP.

4.5.2 *Two-hop Data Transmission*

Data sending in two hop is more complicated than one-hop transfer. On reception of CTS response, both the sender and relayer mark themselves as busy. The relayer on reception of data always checks who the data is intended to from the packet contents. It extracts the PC address from the packet contents and forwards the data to the extracted PC address. Now, for the PC, which is also the two hop recipient, the data appears to be forwarded as one hop traffic from relayer. The PC further en-queues the collected data in I_CTPCollectionDataQueue on reception of termination beacons.

4.5.3 *Data Processing Phase*

The incoming data from sender is processed on each packet reception. On reception of data from the intended sender, the PC serially forwards it to a heterogeneous device (such as Raspberry Pi or a computer) over usb connection. We had motivated the need of heterogeneity (in chapter 1) for running memory intensive and/or computation intensive algorithms like data compression, Fourier trans-

form, data analysis on the PC. We can implement these algorithms on the attached peripheral and forward the incoming data serially to the heterogeneous device. We would like to mention here that data processing power of a PC entirely depends upon the computational capabilities of the peripheral component.

4.6 PROCESS MODEL

Our model works on the systematic calling of timers when required. Some of these timers follow periodic patterns while others are called when certain conditional elements are encountered. For the conditional timers, the timer is called once by *Timer.StartOneShot* method and this calling repeats in a loop until the condition to execute the loop holds. In the following subsections we will describe these timers and conditional elements to keep these timers running.

4.6.1 *System Overview*

The work flow of heterogeneity model is illustrated using the figure 4.1. This work flow is further briefly discussed in following points:

1. BeaconTimer: This is a periodic timer fired by PCs. On firing of the timer, a PC advertises itself as the available processing device centre in two hop neighborhood.
2. This advertisement is updated and sorted locally by each SN periodically by SortProcessingCentreBasedOnETX timer.
3. Intended sender nodes use the best available PC and sends RTS request using PreambleSendingTimer.
4. The PC replies to the RTS via CTS messages.
5. SNs snoop the CTS response and update their Busy PCs and relayers struct array.
6. On receiving CTS response by the intended sender, DataSendingTimer is fired and data transmission phase starts.
7. On completion of data transmission, CTPCollectionTimer collects this data and sends its via CTP to BS.
8. In case of no PC availablity in one or two hop neighborhood, the SN participates in the CTP forwarding of data via CTPCollectionTimer.

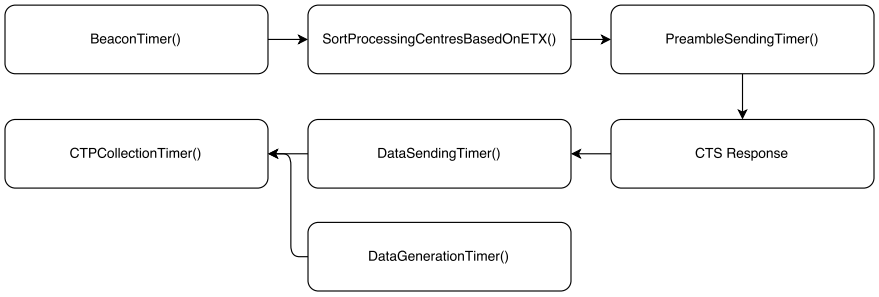

Figure 4.1: Timer Work Flow Summarised

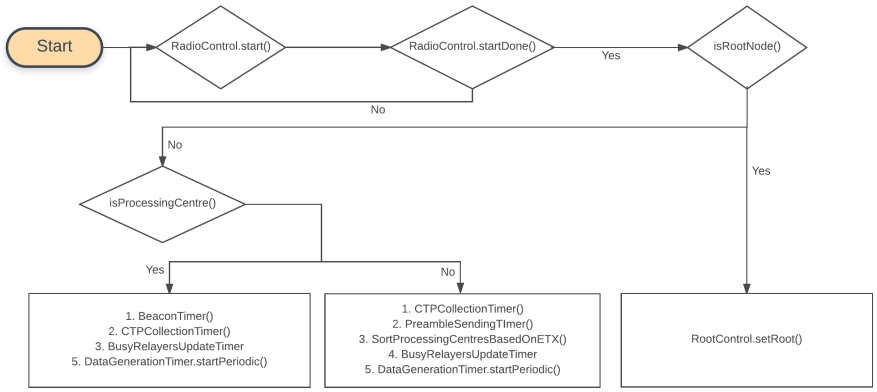

Figure 4.2: Timer Interfaces in Heterogeneity

In subsection 4.6.2, we will differentiate the timers for different types of SNs. And in the further subsections we will discuss these timer

4.6.2 *Timers Flowchart Diagrams*

In figure 4.2, we have illustrated a flow chart for the lifecycle of a SN participating in heterogeneity design. Initially, every SN turns it radio transceivers on and then, only when the RadioStart.startDone event signals without any errors, the SN proceeds further to start certain timers. The selection of timers is based on two methods:

1. isProcessingCentre: This method returns True, if the address of the current SN (obtained by TOS_NODE_ID) matches a list of PCs defined as enum constants.

2. isRootNode: This method returns True, if the address of the current SN (obtained by TOS_NODE_ID) matches a list of BSs defined as enum constants.

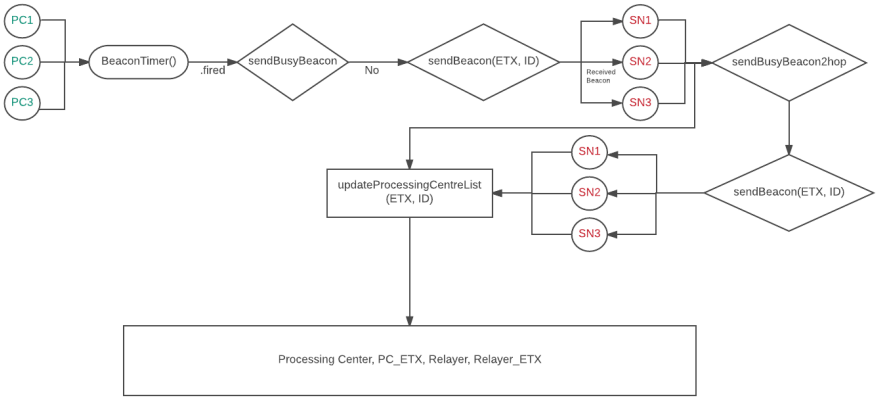

Figure 4.3: Beacon Timer

Based on the return values, appropriate timers are triggered. We will look at these timers in detail in coming subsections.

BeaconTimer

It is a periodic timer fired by PC at regular intervals. Figure 4.3 explains this concept in more detail. Firstly, each of the PCs checks sendBusyBeacon flag. If the flag is found not busy, a beacon, consisting of ETX and PCid, is broad-casted in one-hop neighborhood. On reception of these beacons, the SN first update their local copy of processing centre struct array, which consists of PC id, PC ETX, relayer id and relayer ETX and then they forward it to their one hop neighbors. In the next subsection, we explain the PreambleSending Timer diagram which uses these updated local copy of PC to send RTS packets.

PreambleSendingTimer

Each SN maintains an updated list of PCs in two hop neighborhood and further sends a RTS packet to request data processing by a PC, based on the current state (occupied or free) of the SN. We will explain this concept with the help of figure 4.4. The model requires the sorted data from *SortProcessingCentresBasedOnETX* timer. This timer periodically sorts the data in processing centre struct array based on lowest ETX value and neighborhood distance. The PreambleSendingTimer then uses two arrays to decide whom to send RTS: 1. sorted array of PCs and 2. busy PCs and relayers struct array (initially this list is empty, which means no PC is busy initially). If a PC is found by the method getProcessingCentreForTransmission using the above

mentioned arrays, a RTS packet is sent by setting transmitter address, receiver address, relayer address and duration of transfer desired as the packet contents. The recipient verifies if it is a relayer by extracting the relayer address from the packet contents (for a relayed transmission the packet content will have non-zero relayer address). In relayed transmission case, the packet is further forwarded to destination PC. On reception of RTS request by the PC, it queues the request in CTSQueue and sends the CTS response to the first RTS sender. The sending response is broad-casted so that spectator SNs update their busy PCs and realyers struct array. This prevents other senders from not selecting an already occupied PC. After getting the CTS response, DataSendingTimer is fired which initiates the data transmission phase. The flowchart for DataSendingTimer is explained in next subsection.

DataSendingTimer

DataSendingTimer is triggered only after RTS response has been made by PC. Figure 4.5 elaborates the data sending concept of heterogeneity model in detail. Once data sending timer is fired, the intended sender verifies whether the time allotted to it is still remaining. If this condition holds, data is sent to the PC or one hop relayer depending on whether it is one hop transmission or two hop transmission respectively. The AMSend.send signals send.sendDone event after successful or unsuccessful data transmission. We utilise this signalling event to again call data sending timer for sending the next packet.

CTPSendingTimer

Data received by PC is computed and added to CTPCollectionDataQueue. CTPTimer fires periodically and collects this data to route it via CTP to BS. If there are no elements in this queue, the timer uses DataGenerationQueue to route data via CTP. DataGenerationQueue contains data acquired periodically by the SN. For demonstrating heterogeneity, we have generated random numbers by periodically firing DataGenerationTimer. This concept is shown in flowchart 4.6.

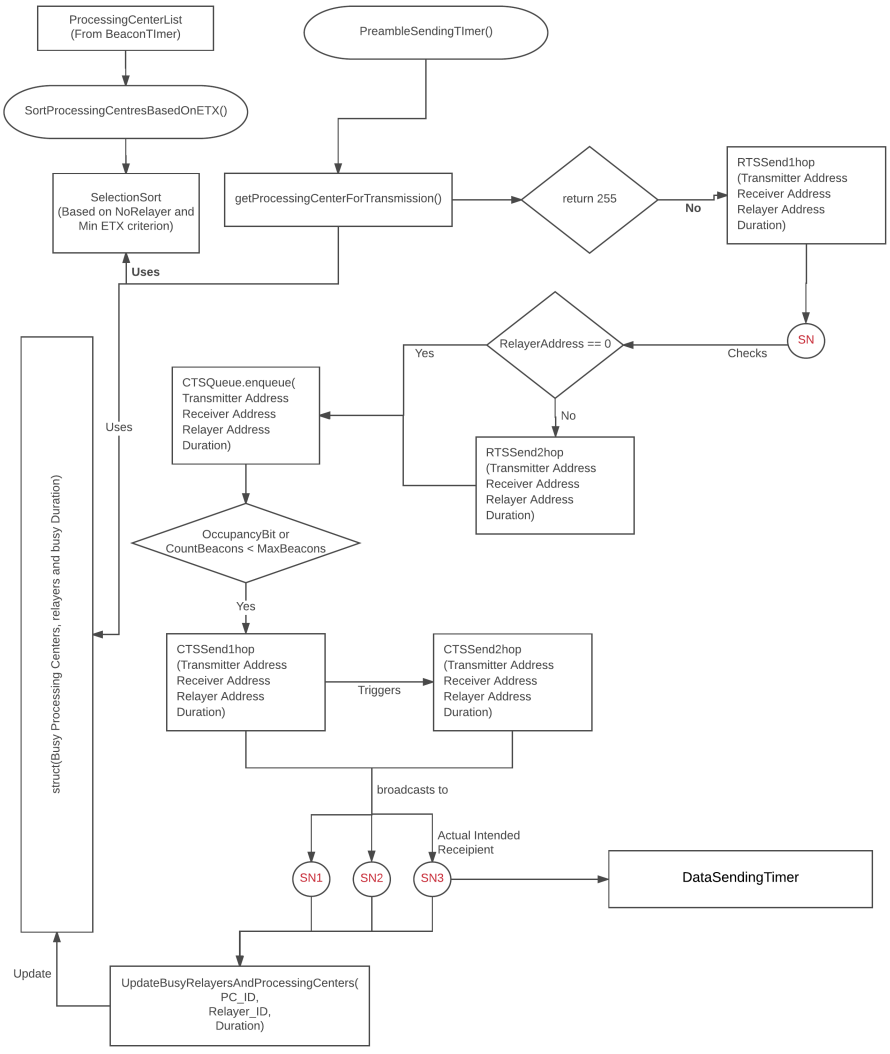

Figure 4.4: Beacon Timer

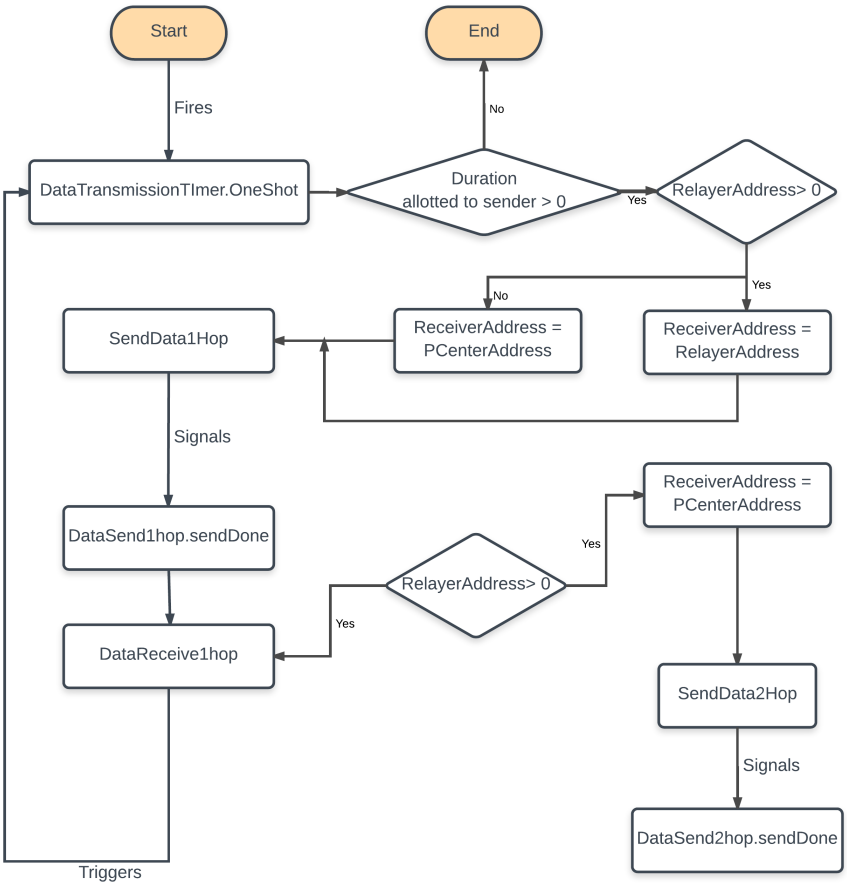

Figure 4.5: Data Sending Timer

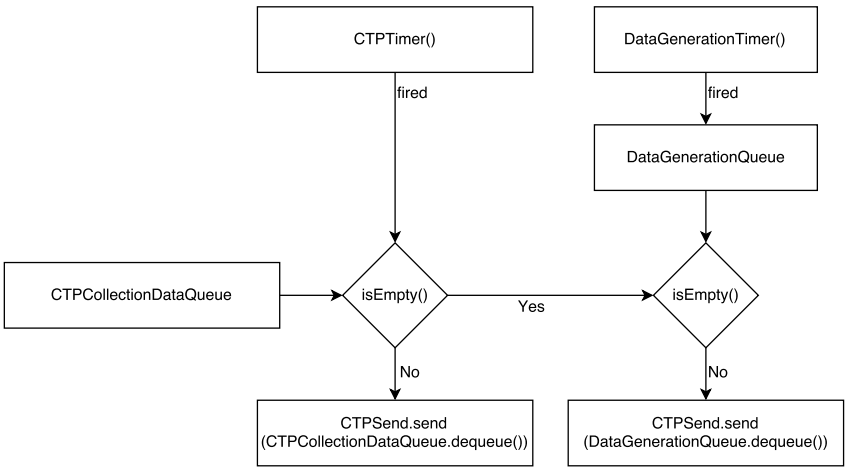

Figure 4.6: CTP Sending Timer

5 EVALUATION

This chapter evaluates the implementation of heterogeneity model as described in chapter 4. We will base our test data obtained from Cooja simulator log outputs. Cooja is a network simulator, which allows execution of TinyOS compiled codes in a virtual Telosb WSN set up. Using the simulator, we can debug the model using print statements. These statements are displayed in Cooja *mote output* window along with the time-stamp and sender id of debug statements. Cooja log files can be saved as *.txt* files and therefore could be further used to obtain relevant visualisations and numerical calculations using statistical programming language such as *R*.

In the first section of this chapter, we will explain the structure of test results obtained from the heterogeneity model along with the result analysis and in the second section we will evaluate our model for the five goals: reliability, robustness, fairness, efficiency and hardware independence.

5.1 STRUCTURE AND ANALYSIS OF TEST RESULTS

To compare the analytical results with simulations (using Cooja), we use the network topologies as shown in figures 5.1 and 5.2. Since, the size of the topology, the number of nodes deployed and number of nodes present in the PC neighborhood can have significant impact on the protocol behaviour, we have focused on two important scenarios:

1. Frequent one hop data transmission: Demonstrated using 5.2.
2. Increased chances of two hop transmission: Demonstrated using 5.1

Figure 5.2 describes the tree arrangement of SNs in which the parent nodes three and five act as the PCs. We have also added eight as the shared PC between the children of other two PCs. The reason for choosing this kind topology is to see how heterogeneity behaves in cases of frequent one hop data transmissions. We use another arrangement of SN in an ellipsoidal pattern (as shown in figure 5.1). In this topology, we have placed all the SN in a ring like arrangement in which a SNs has exactly two neighbors in the one hop neighborhood. This network of SNs provides us more opportunities to observe more two hop data transmission requests than the tree topology because in

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]

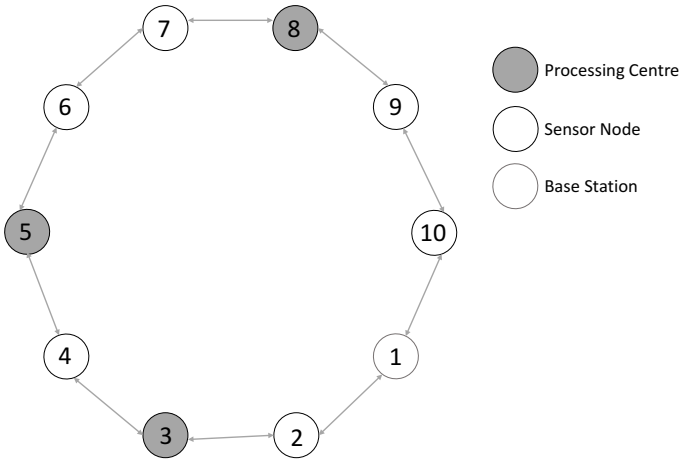

Figure 5.1: Ellipsoid Topology

this topology there can be maximum two nodes competing for one hop data transmission request where as in the tree topology we have more than two nodes competing for each of the PCs.

In the next subsections we will focus on the results obtained by simulating the heterogeneity model on these topologies in Cooja simulator at different heterogeneity data transfer rates.

Sender Enqueue-Dequeue Log

It stores addresses of sender, relayer and PC participating in the data transmission phase of the heterogeneity model. This information is printed using Cooja simulator. The simulator also provides the timestamp for each of the debug statements. This feature is used to calculate the duration of data transfer between participating SNs and further plot a data transfer schedule to visualise the occupancy of available PCs during the data transmission phase [**bitencourt2012simulation**].

The figures 5.3, 5.4 show the data transfer schedules of the intended senders in ellipsoid (5.1) and tree (5.2) topology respectively for approximately one hour simulation run time. It was difficult to plot more than one hour of simulation run time because of memory constraints. The Cooja simulator runs out of memory on simulating large number of nodes more more than one hour in our experiment.

The simulation parameters used to plot these diagrams are explained as follows:

1. In the virtual environment scenario using Cooja simulator, we have selected Unit Disk Graph Medium (UDGM) of data transmission. In this model of data transmission, all the SNs outside

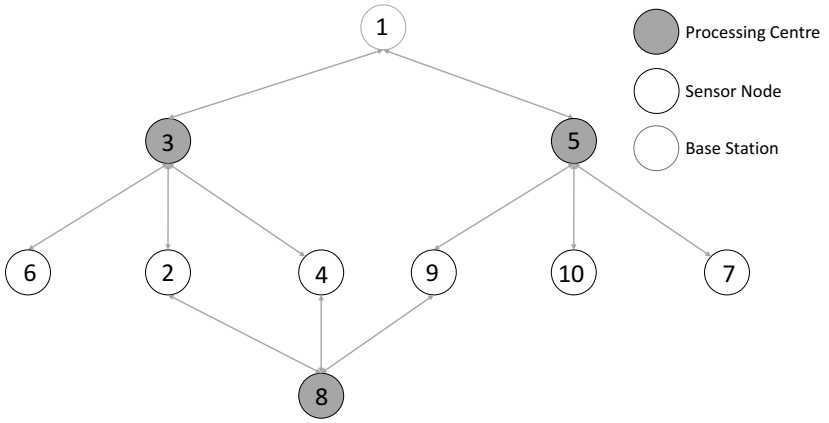

Figure 5.2: Tree Topology

the 100units radius of a SN do not receive any packets at all whereas the SNs within the sensor radius receive all the packets.

2. In both the topologies, SNs with addresses three, five and eight are thePCs which advertise themselves at a periodic interval of five seconds. SN one acts as root node and remaining others are general SNs.

   In tree topology, PC three acts as parent for SNs six, two and four whereas the PC five is the parent for SNs nine, ten and seven. Also, the PC eight is added at the bottom to share processing requests from two, four and nine.

3. In both the topologies, the SNs can reach only it's one hop neighbor directly. For example: In ellipsoidal topology SN five can only communicate with SNs six and four directly.

4. The intended SN requests the available PC for a random transfer duration (in our model a random number between 12-20 seconds).

Enqueue-Dequeue Log Analysis

In this subsection, we will analyse the plot results of Enqueue-Dequeue log file for evaluating our heterogeneity model. Using the log file, we have plotted the data transfer schedule diagrams, shown in the figures 5.3 and 5.4. In these diagrams, Y axis shows the list of intended senders and X axis shows the time period of simulation. Each PC is represented with unique color. The width of the colored boxes in both of these plots denote the time slot allotted to one of the intended

Figure 5.3: Data transfer schedule in heterogeneity: Ellipsoidal Topology

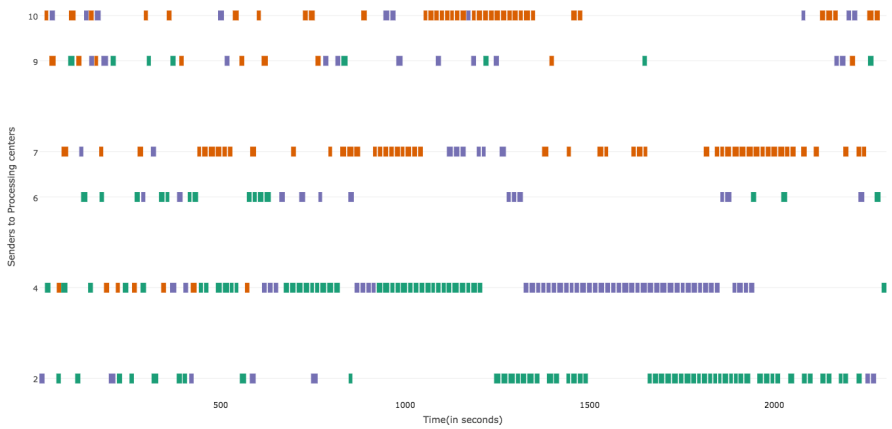

Figure 5.4: Data transfer schedule in heterogeneity: Tree Topology

senders. The non-uniform width of each of the colored slots represents the PC allocation based on the random duration of PC demand, between twelve to twenty seconds, by the intended sender.

From the following diagrams, we can infer the following regarding evaluation measures:

1. Sender Fairness: From the box plot diagram 5.5, we can infer that PCs are more evenly distributed among intended senders in Tree topology than Ellipsoid topology because the contending senders have more fair chances to occupy PC in former topology than the latter pattern. Also, for ellipsoid topology, most of the senders are either in two hop neighborhood of PCs or cannot occupy distant (more than two hop) PCs before the PC's immediate

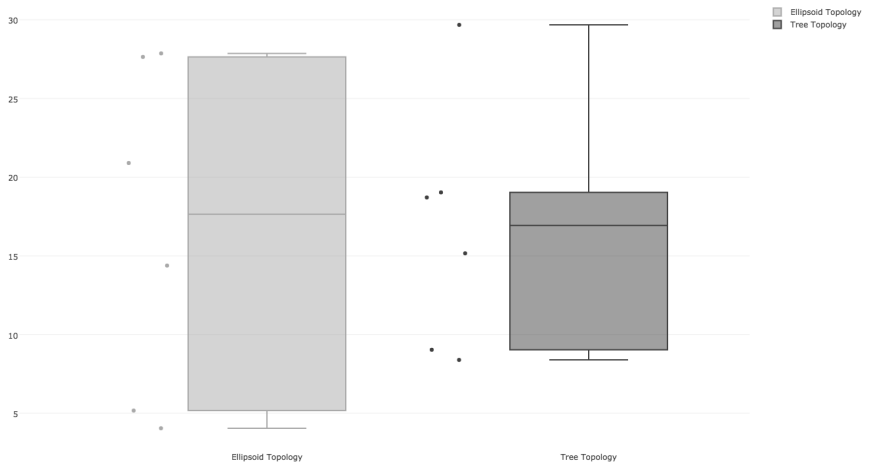

Figure 5.5: Sender Fairness: Ellipsoid vs Tree Topology

neighbor RTS. Therefore, these senders have limited options of selecting a PC for data transmission.

2. Relayer Fairness: The two hop requests in both the topologies are not so frequent because of less chances to get CTS response for a RTS packet from a far away sender than the near by sender. This can be deduced from the figure 5.6, in which the graph shows maximum for both the topologies when relayer address is zero (which means that the relayer is not used and rather one hop sending is used). There are two reasons for this phenomenon: 1. The time taken to reach two hop RTS by a two hop sender is greater than the time taken to deliver one hop RTS by one hop sender. 2. For a two hop request, the sender must first send in its RTS request, a blocking signal for the relayer. This blocking is only confirmed when the relayer, which has to be blocked for the requested time slot, has not sent its RTS request prior to the contending sender. The probability of occurrence for such an incidence is low because the PC exists in the immediate neighborhood for the relayer than the contending two hop sender.

   Also it can be inferred from the diagrams that relayers are used more often in ellipsoid topology as compared to tree topology. This happens because for each PC, in ellipsoid topology, there can be maximum two contending SNs where as in the given tree topology there are more than two senders in one hop contending for a single PC.

3. PC Occupancy: We ran the simulation four times for around seventy minutes and took the average of occupancy duration

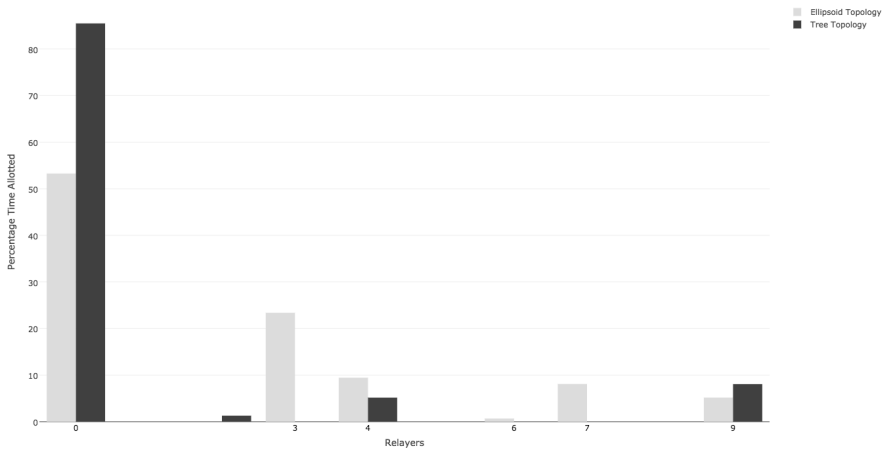

Figure 5.6: Relayer Fairness: Ellipsoid vs Tree Topology

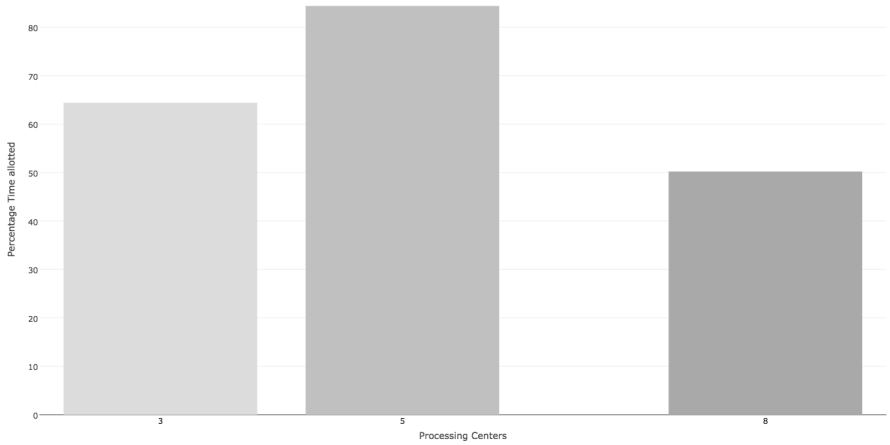

Figure 5.7: Processing Centre Occupancy

for each of the PCs. This reading is plotted in the figure 5.7, where on X axis the PCs are shown and on Y axis we plot the percentage of time allotted out of the total simulation run time. It can be observed from the diagram that the PCs could not be kept occupied 100 percent of the simulation run time. This is because of the numerous RTS and CTS beacon exchanges and contention in the channel to make both CTP and heterogeneity layer work for data transmissions. However, with the achieved PC occupancy for about 50 percent and more for each PC, we are still able to achieve better performance than CTP in terms of number of packets delivered at BS. This is discussed in detail in the next subsection (5.1).

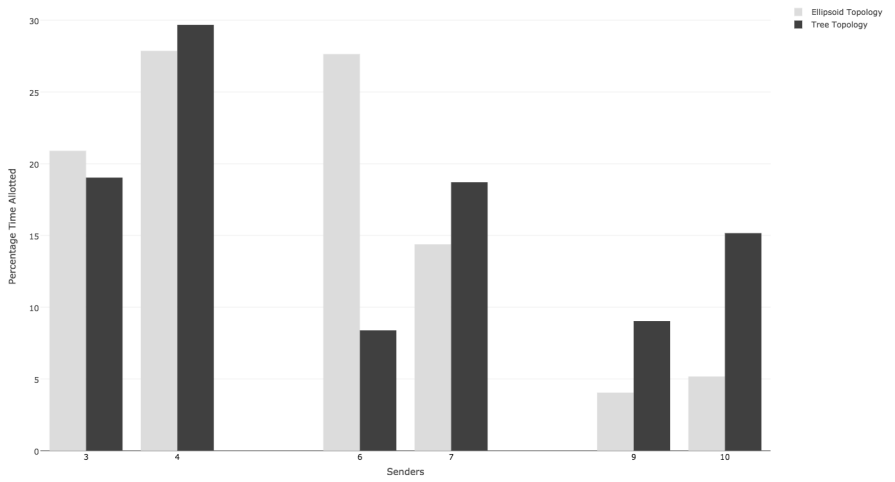

Figure 5.8: Comparison: Ellipsoid vs Tree Topology Sender Fairness

PRR log

This log stores list of packets sent and received by the intended SN and PC respectively. On every packet transmitted/received, total number of packets sent/received counter is incremented. After the sender runs out of allotted time, the counter is forwarded to the serial output and further displayed in the Cooja mote output window. The log file is further saved as *.txt* file to analyse the packet reception ratio in heterogeneity.

PRR Log Analysis

We use the above described log file to analyse following two scenarios.

1. Analyse how PRR changes on changing the heterogeneity data transmission period.
2. Perform a comparative analysis of number of packets received by CTP to number of packets received by heterogeneity at different data transfer rates

Scenario 1: The figure 5.9 represents the data delivery ratio of heterogeneity layer to the PC at different data sending rates. This diagram presents an interesting overview of heterogeneity data sending rate versus PC PRR. It can be concluded from the graph that if we send the packets at lower transmission intervals, say at 50ms or less, we loose more packets however the total number of packets received at the PC increases, whereas if we increase the heterogeneity

data transfer intervals, we see a rise in PRR but overall number of packets received falls down. The increase in PRR from 91 percent to 95 percent is mainly due to reduced heterogeneity data traffic on increasing the transmission interval. With lower transmission interval in heterogeneity layer, the total number of sent packets increases and therefore the sender node needs frequent channel access to transmit the data. However, the channel also has to be accessed by CTP layer for route maintenance and updating and packets transmission. This is the main reason for packet loss. Also, the rise in PRR is primarily due to the fact that on increasing the periodic interval of transmission channel access, we reduce the channel access contention.

The graph also reaches the saturation point of around 95 percent after crossing the 120ms transmission interval checkpoint. The flattening of curve beyond 120ms data transfer interval is mainly because of the following reasons:

1. CTP layer at the back end contends with the heterogeneity layer to send its packets to the BS for data transmission or to maintain the CTP tree with frequent beacon exchange.

2. Given that a collection layer at the back end is prime necessity for periodic data collection of processed or unprocessed data, it is not possible to circumvent the collection layer to improve PRR.

3. Re-transmission of undelivered packets is not implemented in heterogeneity. This is because we do not aim to achieve 100 percent PRR and rather we aim to demonstrate the advantages of local data processing and thus reducing the overall network burden.

Second scenario: This study will demonstrate that a considerable amount of energy and time is saved by performing local computation at heterogeneous node. With this analysis, we also aim to show that heterogeneity reduces the burden of CTP by a reasonable factor in carrying over the data generated at a SN via the collection tree to the BS. In order to perform the evaluation, we need to uniquely mark the data sent by heterogeneous processing point to distinguish it from unprocessed data and further on reception of a data at BS, the counters for processed or unprocessed data is incremented based on the set or unset marker value respectively.

We show this comparison, in figure 5.10, by plotting number of packets processed by heterogeneity to the number of packets collected via CTP in twenty five minutes of Cooja simulation run time at several heterogeneity data transfer intervals. We represent data

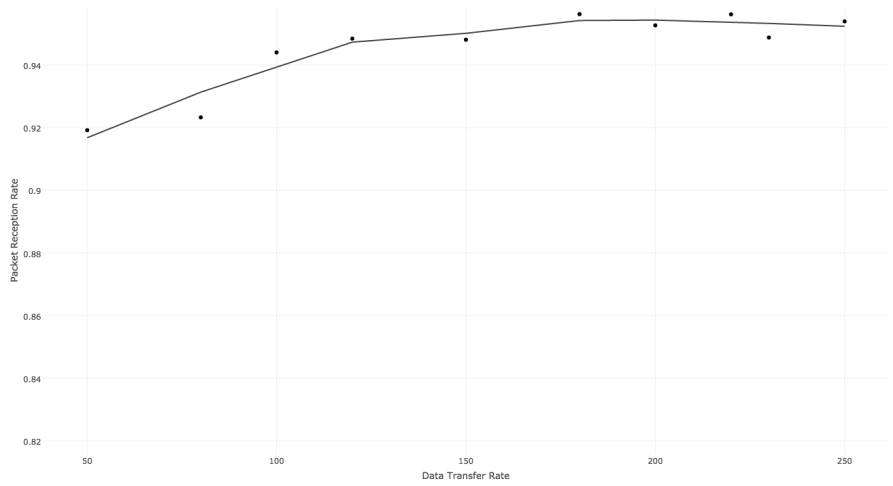

Figure 5.9: PRR for different heterogeneity data transfer rates

transfer rate at collection interval of 2000ms on X axis and PRR for the receiver (which is the PC) on Y axis. From the graph it can be inferred that the heterogeneity layer receive more data at lower data transfer rate. This is because of frequent call to packet sending interface to send the generated data from the intended data sender node. In the diagram, the plot also decreases gradually on increasing the data transmission period because of increased heterogeneity periodic data transmission interval.

In the figure 5.10, the ratio of heterogeneity processed data to CTP unprocessed data at 50ms data transfer interval is approximately 7 : 1. The selection of 50ms data transfer interval has one big disadvantage: only 91 percent of the data sent by sender node reaches the PC (as shown in figure 5.9). We can therefore make a trade-off between data delivery ratio and amount of data processed by operating heterogeneity layer at data transfer rate of approximately 120ms. This will not only give us about 95 percent PRR at PC but also deliver around 3.5 times more packets than the CTP layer. This approach will also reduce the total number of packets flowing through the collection protocol (after the data has reached the destination PC). This is because the data which has to be earlier sent to the BS for processing is already now processed and instead of transmitting all the packets to BS, we send only a few processed packets.

5.2 A PRACTICAL OVERVIEW OF HETEROGENEITY FOR ENERGY EFFICIENCY

To examine the energy efficiency of heterogeneity from a practical overview, let us assume that the PC is five hops away from the BS and

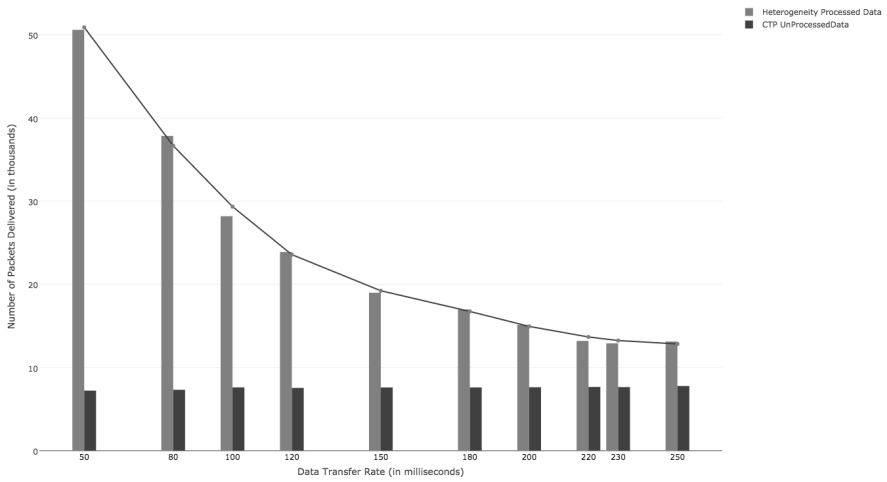

Figure 5.10: Comparison for number of packets delivered at BS by heterogeneity vs CTP at varying data transfer rates

the sender node is in two hop neighborhood of PC. If only the CTP layer is active then the data generated at the SN must travel seven hops to reach the BS for getting processed. By the definition of a heterogeneous processing point, we can also assume that computation time is approximately same for both, heterogeneous node and BS. We can see that the usual flow of large amount of data via CTP not only burdens the entire network but also drains energy of the entire collection tree responsible for forwarding the data to the BS. However, if we turn on the heterogeneity layer, then the data now only has to travel two hops and for the rest five hops the processed data is sent via CTP. This approach saves energy of SNs participating in the collection tree, reduces the traffic flow in the sensor network and also allows heavy data traffic without burdening the entire WSN except the participating nodes.

A

APPENDIX

A.1 INSTALLATION OF TINYOS ON MACOSX

A.2 INSTALLATION OF TINYOS ON UBUNTU

[draft created on April 12, 2017 at 23:59 (delete drafting option in thesis-config.tex to remove this line)]