ABSTRACT

GREWAL, ASHUTOSH KUMAR. Design of a Flexible, Extensible Interface for the CentMesh Architecture for Auxiliary Sensing Devices. (Under the direction of Dr. Rudra Dutta).

The phenomenal advancement in computing has empowered researchers with the capability to embed sensors in the physical environment directly. The CentMesh project provides an open, programmable and extensible testbed for research on Wireless Mesh Networks. The existing CentMesh architecture can only retrieve measurements from the nodes that can communicate using the CentMesh software suite. The contribution of this work is to design a flexible and extensible interface for the CentMesh architecture to provide researchers the capability to interact with heterogeneous sensors loosely connected to the wireless mesh network. This thesis proposes the changes in the CentMesh architecture and the design of a communication protocol that would allow mesh nodes to communicate with the auxiliary sensors and retrieve the measurements. We demonstrate the implementation of this protocol using Motorola Droid smartphone as the auxiliary sensor that connects with the CentMesh testbed over Wi-Fi.

Design of a Flexible, Extensible Interface for the CentMesh Architecture for Auxiliary Sensing Devices

by
Ashutosh Kumar Grewal

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Master of Science

Computer Science

Raleigh, North Carolina

2012

APPROVED BY:

_____	_____
Dr. Mihail Sichitiu	Dr. Sarah Heckman

_____

Dr. Rudra Dutta

Committee Chair

DEDICATION

To my parents, younger sister Mamta, professors and to all my friends.

BIOGRAPHY

Ashutosh Grewal was born on 25th January, 1988 in Hisar, Haryana, India. He did his primary and secondary schooling from Campus School, Hisar, India He received his Bachelor of Technology degree in Electronics & Communication Engineering from National Institute of Technology, Durgapur, India in 2009. He has been a Master's student in Department of Computer Science at North Carolina State University, since Spring 2010.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude towards Dr. Rudra Dutta for guiding me during the course of this work. I am deeply grateful to him for helping me throughout the course of my graduate program at North Carolina State University.

I would like to thank Dr. Mihail Sichitiu and Dr. Sarah Heckman for serving on my thesis advisory committee.

I thank my parents and my sister for their unwavering support, continuous encouragement and love.

I also thank Parth, Can and Anand for their help over the course of this work.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

In this thesis, we extend the reach of CentMesh [3], a Wireless Mesh Network testbed, as a distributed sensing instrument. To develop such a system we propose the design of a protocol that communicates between the CentMesh nodes and the sensor nodes and plan to develop a custom API (Application programming interface) for the CentMesh software suite. We use auxiliary sensor devices (Motorola Droid smartphone) that have multiple sensors, are Wireless Fidelity (Wi-Fi) capable and have some degree of programmability to implement such a capability [15]. We extend the architecture of CentMesh to have its nodes expose one of its interfaces in the access point *(*AP*)* mode. We plan to offload some part of our measurement framework on to the sensing device. On the auxiliary sensor device side, we develop an application for the auxiliary sensors that monitors its built-in sensors, connects to its nearest mesh node - which is running one of the cards in AP mode and stores these readings at the auxiliary store. These readings at the auxiliary store are sent to the Monitoring Manager running at the controller node with the help of Monitoring agent at the mesh node. The Monitoring Manager running at the controller organizes the data into the database, which can be visualized. We also outline the system design issues for connecting such a loosely coupled auxiliary sensor device with the CentMesh testbed.

1.1 Wireless Sensor Networks

A wireless sensor network (WSN) [8] consists of sensors (referred to as sensor nodes) distributed over diverse locations to obtain readings such as temperature, pressure, light intensity, speed etc. Sensor networks unlike other classes of distributed systems are unique in the sense that they generate data from the physical world itself and not by humans [14]. Distributed sensor networks allow researchers to efficiently collect real-time data [13]. The sensor nodes have limited resources but due to their scale they can be very effective. The entire mass of data could be aggregated in a centralized database or an alternative approach could be to store the data locally at the nodes. The sensor nodes generally suffer from storage, processing and power constraints [14]. These nodes may contain more than one type of sensors, can process the information obtained from the sensors and are capable of using networking protocols [15].

The applications of such a network are diverse and could range from military applications, environment management etc. The nodes in such a network could be the same or constitute of heterogeneous elements.

The topology of such networks could be anything from a star network (where all nodes are connected to a single hub) to a multi-hop wireless network.

1.2 Wireless Mesh Networks

Wireless mesh networks is a communication network where the wireless nodes are connected to each other in a mesh topology. Mesh network nodes generally communicate by transmitting to their nearest neighbor [10]. There are two kinds of nodes in such a network, viz., clients and routers. On top of being able to act as a client, the mesh routers have additional routing capability to route traffic to and from the gateway. The Mesh routers have comparatively less mobility and help provide network access to other mesh nodes as well as clients [10]. Wireless mesh network can be connected to other networks using the gateway. Mesh networks are scalable, self-configuring and self-healing networks. These qualities increase their reliability, reduce the cost of deployment and maintenance [10]. This allows for a more effective and low cost way to build community networks, monitoring systems etc. [10].

Mesh nodes at times use multiple interfaces to reduce the interference among adjacent nodes [11]. This approach also allows the nodes to decouple the flow of control and data messages to different interfaces. The wireless mesh network may support multiple radio technologies, viz. Wi-Fi, cellular and sensor networks. The services provided by one network may be used by the other [10].

CHAPTER 2

CentMesh

CentMesh is an open testbed to support research on wireless mesh networks. It was developed at NCSU by faculty and students over the last two years and it re-uses open source code. It seeks to enable researchers who seek to focus on reliability, availability etc.

Following is a diagram that depicts the poletop installation of a mesh node at one of the locations on Centennial Campus [23].

Figure 1. Poletop installation of a mesh node

2.1 Hardware Components

The nodes of the CentMesh are standard desktop computers. Each node of the CentMesh contains four wireless cards. Nodes use MadWiFi driver for wireless cards [3].

2.2 Architecture

One of the CentMesh nodes is assigned as the controller. The controller runs a superset of services as compared to the mesh nodes. There is no wired backhaul, instead one of the wireless card supports the flow of controller messages**.** The CentMesh provides three types of modules with its software suite [3]: -

System modules: - These are the modules [3] that CentMesh nodes must have.

- Communicator - The communicator provides communication between the core modules residing in the same or a remote machine using TCP connections.
- Reporter and Disseminator – Reporter is a one way reporting module that is build upon the communicator and can be used to report to the communicator. Disseminator is a module that can be used by the controller to broadcast information.
- Neighbor Discovery – This process helps a node discover its neighbors. All this information is sent to the controller for further processing.

- Bootstrap Routing – The controller sends the routing table information for every mesh node based on topology of the network. If the experimenter does not have a module for routing, this is very useful.
- Data Repository – The system modules update the database with their reported information which can be used by core modules for their computations.

Core modules: - These modules [3] communicate with each other using the Publish-Subscribe mechanism. The publisher sends the topics they wish to publish to the communicator which maintains a list of subscribers that wish to subscribe to that topic. There are two parts to this module, one is the manager that resides at the controller and contains the algorithm while the agents running at all mesh nodes report the relevant metrics.

- Routing – The routing is based on ETT (Expected Transmission Time) metric. The agents at the nodes collect this information for their links and report this to the controller which runs the algorithm to calculate the routes. This information is then distributed to the mesh nodes.

OAM Modules: - Operations, Administrations and Maintenance modules.

- Network Monitoring – The purpose of these modules [3] is to collect and report network status information. This provides the researcher with network status samples

at fixed intervals at the controller. The researchers can specify a set of commands which are then run at a specified frequency on the mesh nodes with the help of Monitoring Agents.

- Visualization – The visualization program [3] extracts information provided by the network monitoring modules from the database and converts it into Keyhole Markup Language (KML) format, which is then used to visualize the information with the help of Google Earth.

CHAPTER 3

Sensors

As already stated, we'll be using the Motorola Droid phone as our auxiliary sensor. The Motorola Droid has the following sensors built into it, viz., accelerometer, magnetometer, proximity sensor, ambient light sensor and can also provide location services using Global Positioning System (GPS) [24]. The Android API exposes these sensors using the SensorMananger class and provides location using the LocationManager class. On registering a listener for a given sensor, a SensorEvent object specifies the sensor's data, type, time-stamp and accuracy.

Accelerometer -

The accelerometer measures acceleration that the device is subjected to. When the device is stationary it would read gravity as 9.81 m/s^2. On the other hand, if the device is in a free-fall, it would read it as 0 m/s^2.

The return values of the values method of the SensorEvent class for the accelerometer sensor are in m/s^2 units.

[0]: Acceleration minus Gx on the x-axis

[1]: Acceleration minus Gy on the y-axis

[2]: Acceleration minus Gz on the z-axis

Orientation –

The orientation is calculated with the help of both the accelerometer and magnetometer. The device coordinate system is converted to the world co-ordinate system. Following diagram depicts the coordinate system used by the SensorEvent API [5].

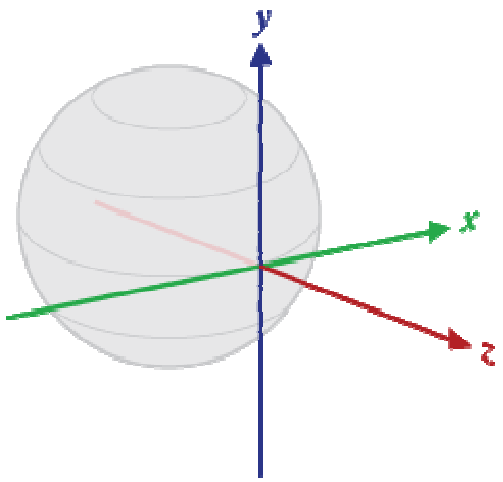

Figure 2. Reference coordinate-system

The following are the values representing the device orientation [6]

- [0]: *azimuth*, rotation around the Z axis.
- [1]: *pitch*, rotation around the X axis.

- [2]: *roll*, rotation around the Y axis.

In the reference co-ordinate system

- X is defined as the vector product **Y.Z** (It is tangential to the ground at the device's current location and roughly points West).
- Y is tangential to the ground at the device's current location and points towards the magnetic North Pole.
- Z points towards the center of the Earth and is perpendicular to the ground.

Proximity -

The proximity sensor reports the distance in centimeters of the device from another object.

Light -

The Ambient light reports the luminance level in lux units

Location services using GPS -

This provides the longitude and latitude in decimal degrees.

CHAPTER 4

Related Work

Network researchers require frameworks to gather data from sensors deployed in the environment in a distributed manner. Also, mobile networks are limited by their single interfaces, sporadic connectivity and no dedicated control network constraints to report readings using a measurement framework. There are several existing measurement frameworks to collect real time data from a network testbed. In the following section we introduce two of them.

4.1 OML (ORBIT measurement library)

OML is a measurement framework for researchers that can be used to collect data from a distributed environment. It was developed as part of the OMF (cOntrol and Management Framework) but has since graduated to a standalone system. The method of logging the results locally was deemed ineffective owing to the various types of formats and the sheer number of such log files. Therefore, the need for such a framework that can collect experimental data efficiently was felt [16]. OML provides the researchers with the tools to describe and deploy the experiment and retrieve measurements from the testbed [12]. The experimenter describes his experiment in a high level language and the framework runs that

experiment on the network testbed. Once the experimenter describes the experiment, OMF manages the configuration, deployment and execution of that experiment on the testbed [12]. The readings are pulled from the testbed by the library and stored in a repository. The readings are pulled while the experiment is running. The real-time retrieval provides the experimenters the ability to actuate things based on the readings received [16].

The framework has two types of nodes, viz., control and experiment nodes. The control nodes would connect with the experiment nodes in the network testbed and configure them for the experiment.

OML is based on client/server architecture. The OML measurement library consists of three main components: -

- OML client library: Client library creates streams from the measurement points (MP) by filtering the inputs they receive from the MPs. The Measurement Streams (MS) can be discarded or accepted depending on the requirements of the current experiment. The experimenter can specify the frequency or event at which the MPs should send their readings.
- OML Server: Server collects the data it receives from these streams and stores it in a repository [18].

- OML Proxy Server: Proxy Server buffers the data until the node reconnects to the network.

When researchers wish to conduct experiments using mobile nodes, they face the challenge that the nodes might go out of the network. The OML framework addresses this by creating an OML Proxy Server. The Proxy Server can run on the same or a different node depending on the requirements and provides a similar API as the OML server to the client library [16].

The experiment data is stored in a SQL database.

OML also provides a mechanism to include the timestamp during the time of each recording.

4.2 perfSONAR

perfSONAR ("**Perf**ormance focused **S**ervice **O**riented **N**etwork monitoring **AR**chitecture") is a network monitoring system based on Service Oriented Architecture [2]. It provides researchers with the ability to test and measure network performance. Service Oriented Architecture is a way of developing software in the form of entities each of which can perform certain function and which can interoperate. The perfSONAR framework contains services that assist in network measurement data in a federated way. The perfSONAR framework uses Web Services Notification which defines the APIs that publishers and

subscribers expose to the outside world.

Each measurement tool is wrapped by perfSONAR web service. perfSONAR uses Apache axis to provide its Web Services framework. A client sends a formatted Simple Object Access Protocol (SOAP) message which is interpreted by the Axis as a method exposing the service. The client would only need to use the SOAP to be able to avail the services making this communication platform independent. The formatting of the SOAP would be dictated by the Web Service Description Language [2].

Architecture

At the lowest end of the pyramid are the Measuring Points. They collect various types of network measurements by probing the network and store this data in the archives. The Measurement Points report their readings to Measurement Point Service. Measurement Points Service collects and publishes the data that the network domain measurement tools have collected. The frequency of the collection of data can be specified [2].

In the middle of the User Interface layer and the Measurement point layer there is the Service layer which controls access and manages various resources across the network. The User Interface layer is primarily a visualization layer. The layer visualizes the data it receives from measurement points and lets researchers see this data in an easy to understand format [2].

The web services that perfSONAR provides have been categorized in to two families, the enabling services and the performance data services. These web services provide a communication framework to facilitate the exchange of data from Measurement tools to the archives. The messages exchanged could be performance data, instructions, requests, responses etc. There is another categorization of web-services based on the specific type of functionalities that it provides [2]. These families are described below: -

The data can be stored in an archive or published to another web service.

- Measurement Archive Service – It reads measurement data held in filesystem and databases and publishes it to the archives. This can also be used to write measurement data to a repository.
- Transformation Service – This service sits between the data producers and consumers and modifies data depending on the requirements. It could do things as aggregation, and filtering of data.

The above two services belong to the performance data services family. The performance data services interact with entities that collect and store network monitoring data [2].

- Lookup Service – Services register their existence to the Lookup Service. Clients query the Lookup Service to find the services they want to avail.

- Authorization Service – This service helps restrict access to users based on what group they are part of and depending on the policies of that domain.
- Resource Provider Service – With the help of this service access to limited resources can be controlled. It can also schedule the usage of such constrained resources.

The above three services belong to enabling services family [2].

CHAPTER 5

Our Approach

It is a desired feature of wireless mesh network like CentMesh to allow access to heterogeneous clients. This significantly increases the usability of such a network [10].

One such class of clients could be auxiliary sensors loosely connected to the CentMesh network reporting measurement data. These sensors could be connected by a wire or wirelessly. We view the auxiliary sensors as devices hanging off the CentMesh Network like a GPS device hangs off an automobile. The capabilities and ubiquitous presence of smartphones make them an appealing choice.

In the following sections we use the example of a Motorola Droid phone as an auxiliary sensor for the development of this custom CentMesh architecture. As noted before, the Motorola Droid includes sensors like GPS with internal antenna, Accelerometer, Magnetometer, Proximity Sensor, Ambient Light Sensor etc. to collect information from their environment.

perfSONAR and OML are less than appealing choices for designing a framework for such a network scenario. The perfSONAR measurement system is limited by the fact that it needs to

know the topology beforehand to know exactly where the data is being held. Also, the framework is focused on monitoring network performance across multiple domains.

OML on the other hand is an efficient way to run experiments limited by time while we seek to provide access to CentMesh for longer periods of time.

Our goal is to develop a flexible, less complex framework for retrieving measurements from auxiliary sensors to extend the reach of CentMesh as a distributed sensing instrument.

5.1 Design

To be able to initiate a connection between CentMesh network and auxiliary sensor, we create a custom API for the CentMesh architecture. We assume that the auxiliary sensor has some sort of computer network technology to be able to communicate with the CentMesh node. As noted before, we thus propose to implement such a system using Android smartphones. Smartphones are an ideal platform for such sensing devices because they can provide such a connection over the Wi-Fi. We plan to offload some part of our framework on to the sensing device. This means that the sensing device would have to have some programmability to be able to instantiate our program. Again, smarphones' capability to host programs makes them an appealing choice.

5.2 Implementation

The Motorola Droid smartphone connects to the CentMesh nodes using Wi-Fi. The Android phone starts an application that collects the sensor readings from the Android device. For the purpose of this implementation, the mesh nodes are using one of their cards in AP mode advertising a “centmesh-access” network. Also, each mesh node is running a Dynamic Host Configuration Protocol (DHCP) server that assigns Internet Protocol (IP) addresses to the Motorola Droid phones that are trying to connect to the network.

The mesh nodes are also running a UDP server where these Motorola Droid phones connect as a client. Once the Motorola Droid phone obtains an IP address in the same subnet as that of the DHCP server, it will be able to connect as a client. The server assigns the Motorola Droid phone an ID (identification). The server specifies the time span after which the client should report the measurement readings it collects to the server. Everytime the timer goes off, the client sends its readings to the server – the mesh node to which the Motorola Droid phone is connected. These readings are stored in the auxiliary storage (text file) at the mesh node. The Monitoring Agent reads the auxiliary file while it collects other network status information and sends it to the controller. At the controller the readings from the sensors are stored in a database table.

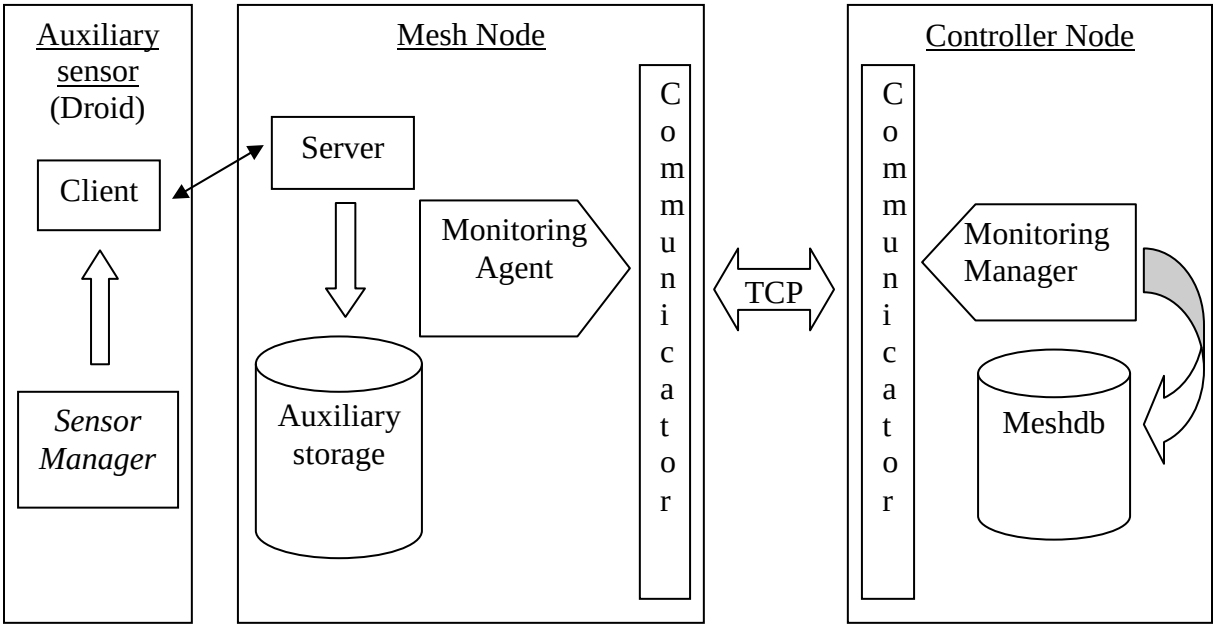

Figure 3. Architecture to connect the auxiliary sensor to the CentMesh

5.3 Design Decisions

Rendezvous

This protocol seeks to enable resources, viz., mesh nodes and sensors to discover each other when they would try to enable connection for the first time. The first time when a sensor node tries to connect to the CentMesh network, it sends out broadcast messages at fixed time interval and waits for reply from any CentMesh node in its range. The CentMesh nodes act as

UDP server and listen on port 8888, well-known listening port for this service. This enables the sensor nodes to not remember IP address of the various CentMesh nodes and makes the protocol more extensible since adding a new node would not require any changes to the sensor node's API.

Rediscovery

Here we address the issue of how would a client reconnect after it has gone out of range. We first address the issue of how a client would recognize that it is no longer connected to the network. The client recognizes that it is no longer connected to the network by realizing that it is no longer receiving acknowledgements for the measurement readings that it is sending. On the loss of three consecutive acknowledgements, it stops sending the measurement readings and instead seeks to reestablish its connection with the network retaining its identity.

Identity

Every sensor node that connects to the network must be uniquely identified. The sensor node that connects to a mesh node may move and get out of the range of the current mesh node and instead connect to a different mesh node. In our protocol we preserve the identity of the

node in such situations. Every mesh node is allocated a set of unique identities that it can only allocate. On moving out of range of the CentMesh node that allocated the ID to the sensor node, the sensor node retains and doesn't seek a new ID while trying to reestablish the connection. This enables a sensor node's data to be uniquely associated with the sensor node at the CentMesh controller node even though the sensor node may have connected to several mesh nodes to report its data since it came up.

Virtual AP vs Real AP

The current CentMesh nodes have four wireless cards. There is no wired backhaul, instead one of the wireless cards supports the flow of controller messages while the other three are to help with transfer of data messages.

There are several ways to create a wireless network that will allow auxiliary sensors to connect. Here we outline the rationale behind our choice of custom API for CentMesh.

The first way to approach the problem was to have one of the cards running two virtual interfaces. This approach would not require changing the existing CentMesh architecture. The new virtual interface would run in the AP mode and run a DHCP server. However, it was noticed that even though the Motorola Droid phone was being assigned an

IP address, the mesh node was unable to ping/send traffic to it while the Motorola Droid phone was able to ping/send traffic to the mesh node. We verified this observation by creating User Datagram Protocol (UDP) / Transmission Control Protocol (TCP) data streams using other network testing tools, viz., Iperf. Moreover, we observed that the Motorola Droid

phone would seek an IP address from the DHCP server every few minutes, much before the lease time ended. We explored several possibilities in the following sequence to ascertain the reason behind this phenomenon –

The first possibility that we explored was whether the Motorola Droid was not accepting packets. We ran the Android tether application, which acts as a DHCP server to assign IP addresses to other Motorola Droids, on one of the Motorola Droid and observed that all the nodes were able to ping/send traffic to each other, so we concluded that this was not the reason. No CentMesh node was part of this setup.

We also investigated whether the mesh node from where pings/traffic was being sent was suffering from something called Address Resolution Protocol (ARP) flux (where a node with multiple interfaces arbitrary chooses one of the NICs to ARP. To ascertain if this was indeed the problem, we tried changing the /etc/sysctl.conf file to force the Linux to use a particular interface for a particular subnet. The ARP requests continued to fail.

We noticed that the mesh node to Motorola Droid ping worked when there was only one mesh node up. Therefore, we believe that the reason for this is that when only one CentMesh node is up, the CentMesh software does not communicate with any other node.

We tried to ascertain which module of the CentMesh software suite is causing the problem, when more than one CentMesh node (a controller and mesh nodes) are communicating, by disabling the managers and agents one by one. We observed that even after every module was disabled and if just both these machines were booted up, the ping won't work.

However, when the other virtual interface, that was running in ad-hoc mode (on the same interface on which the virtual interface in AP mode is running), was deleted, Motorola Droid and mesh node were able to ping/send traffic to each other as expected.

While configuring the virtual interfaces, we had realized that to create a virtual interface in AP mode, we needed to create it first before creating the virtual interface in adhoc mode on the same card. If we were to interchange the sequence of creation of these virtual interfaces on the same card, the second virtual interface can't be created in AP mode. Ordering of creation of interfaces in the AP to adhoc mode creates a dependency and is a non-intuitive design choice.

We conclude that that even though the AP mode virtual interface gets created, if it is created

before the virtual interface in adhoc mode on the same card, it doesn't eventually work. We note that this phenomenon is recurring characteristic [22] as noted below –

"If you are new to MadWifi-NG, it is designed in such a way as to allow multiple virtual cards to work in parallel on the same physical device. This means one could (in theory) create one virtual card to run as an AP, another to run as a client on some other network, and a third to run in ad hoc mode, all at once.

…

Tim has found that release version 0.9.2.1 of the MadWifi drivers work well when using only a single virtual interface, but when multiple are used (such as one running an AP and another connecting to a secondary network), the secondary won't work."

Hence, we conclude that we cannot run two virtual interfaces in different modes on the same card. In our opinion, such a design would be brittle. Therefore, instead we destroy one of the interfaces running in adhoc mode on the CentMesh nodes to free it up for creation in AP mode. This would require changes in the CentMesh software as the current software architecture is based on four wireless cards per CentMesh node. We discuss the proposed custom API for CentMesh in greater detail in Section 5.4.

Adhoc vs access point

In the AP mode the device acts as the Access Point for other wireless clients in its wireless local area network (WLAN) while in the adhoc mode, the device is in a peer-to-peer WLAN

and no Access Point is required [19]. Adoption of wireless ad-hoc networks for the sensors would significantly enhance the reachability of the sensor network. Such a sensor network would then be able to implement its own routing protocol, viz., the Dynamic Source Routing or Ad Hoc On-Demand Distance Vector Routing etc. [20]. However, currently Motorola doesn't support wireless ad-hoc networks for its Droid phone. There are third party applications that enable access to Wireless ad-hoc networks. These methods involve rooting the Motorola Droid phone, flashing the official Android based firmware and then switching to a custom firmware developed by the Android community.

Since the realization of ad-hoc WLAN requires customization at the auxiliary sensor end, in our opinion the effort and hardware resources required to enable such access for Motorola Droid phones would limit the possibility of upgrading the phones' firmware to the latest available build and would introduce a third party dependency in the design effectively reducing the extensibility of the system. Hence, we instead chose to connect the auxiliary sensors via Wi-Fi to the AP.

5.4 Planned custom API for the existing CentMesh software suite

As noted in the previous section, we decided to pursue another approach which would not require any change in the current CentMesh hardware configuration. One of the three data

cards that are in the data plane can be used in AP mode instead of adhoc mode. This will not disturb the CentMesh routing if certain changes are made in the source code as highlighted below.

We create an interface in AP mode at the wifi3 interface (with a different Extended Service Set Identification (ESSID) called centmesh-access), install DHCP server on it and connect the Motorola Droid to this network (centmesh-access) by requesting the IP address from this DHCP server.

To make the CentMesh software work with just three interfaces and use the fourth wireless card only in AP mode, we changed the CentMesh's source code to stop discovering ath3 (now an ap mode interface) as a neighbor on one's local machine and have metric calculations done, since it'll not be participating in the routing of the CentMesh. We would need to stop the neighbor discovery process in startmeshnode.sh, a configuration script in the CentMesh source code common to all mesh nodes, to not let the ath3 interface which is now

in AP mode, try and find its neighbors. We also need to stop the routing manager on the ath3 interface in startManagers.sh, a configuration script for the controller node that runs the manager processes.

As noted before, the mesh node will store the readings at the mesh node's local machine by saving the measurement readings in an auxiliary storage. The use of an auxiliary structure decouples client/server architecture that is responsible for communication between the mesh node and the Motorola Droid from publish-subscribe mechanism that CentMesh uses for the flow of its control messages.

CHAPTER 6

Protocol Design

The protocol maintains a Finite State Machine (FSM) for each client or server. Each mesh node server must listen on UDP port 8888 (server's well-known listening port for our service). The client and server exchange messages.

At the client side the following actions prompt it to change its state as shown in Figure 7. below.

1. The first step of the protocol is to make the rendezvous happen for the client and the server. When the sensor node connects to the centmesh-access network, the sensor node will try to connect as a client to the UDP server at the CentMesh node that is listening on its 8888 port. The sensor node sends out a rendezvous request, as show in Figure 4, as a subnet broadcast and sets the Connect timer to ten seconds. If the sensor node has not been assigned an ID yet (hasn’t established a session with a mesh node ever), it requests an ID (the client will highlight that by having 00000000 as the bit values in the ID field) in the rendezvous request message.

Preamble	Action code	ID

←1 byte→ ←1 byte→ ←1 byte→

Figure 4. Packet representing a rendezvous request by client

2. The Connect timer expires before any reply is received from the server.
3. Sensor node receives a reply for its rendezvous request from the mesh node. It disables the Connect timer, sets Report timer to the value specified by the mesh node and sets its own ID to that received from the server.
4. Receive capability request from the mesh node.
5. Send the capabilities i.e. supported frequency for every sensor on the sensor node, as show in Figure 5.

Preamble	Action code	ID	Sensed quantity code	Representation	Frequency

… last three tuples repeat

←1 byte→ ←1 byte→ ←1 byte→ ←1 byte→ ←1 byte→ ←10 byte→

Figure 5. Packet representing capability reply by client

6. Receive the negotiated capabilities from the server.
7. Set the Collect timer for each sensor to the values specified by the server.
8. Collect timer expires for a particular sensor, signaling that the sensor node should collect readings for this particular sensor.
9. Collect the readings for the sensor whose corresponding Collect timer expired and reset the collect timer for another set of potential readings.
10. Report timer expires, signaling that measurements are due.
11. Report all the readings to the mesh node since the last transmission, as show in Figure 6 and reset the report timer for another set of transmission.

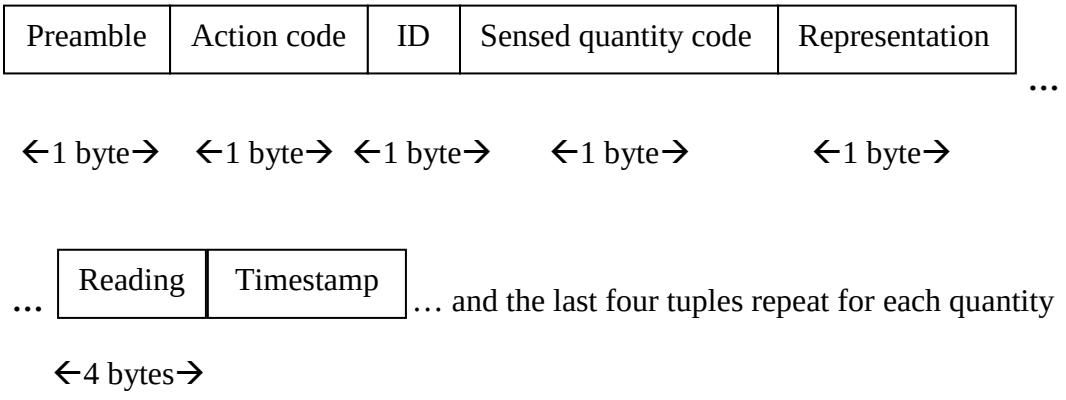

Figure 6. Packet representing data sent by the client

12. Connect timer expires for the third time without any acknowledgement for the last three readings sent.

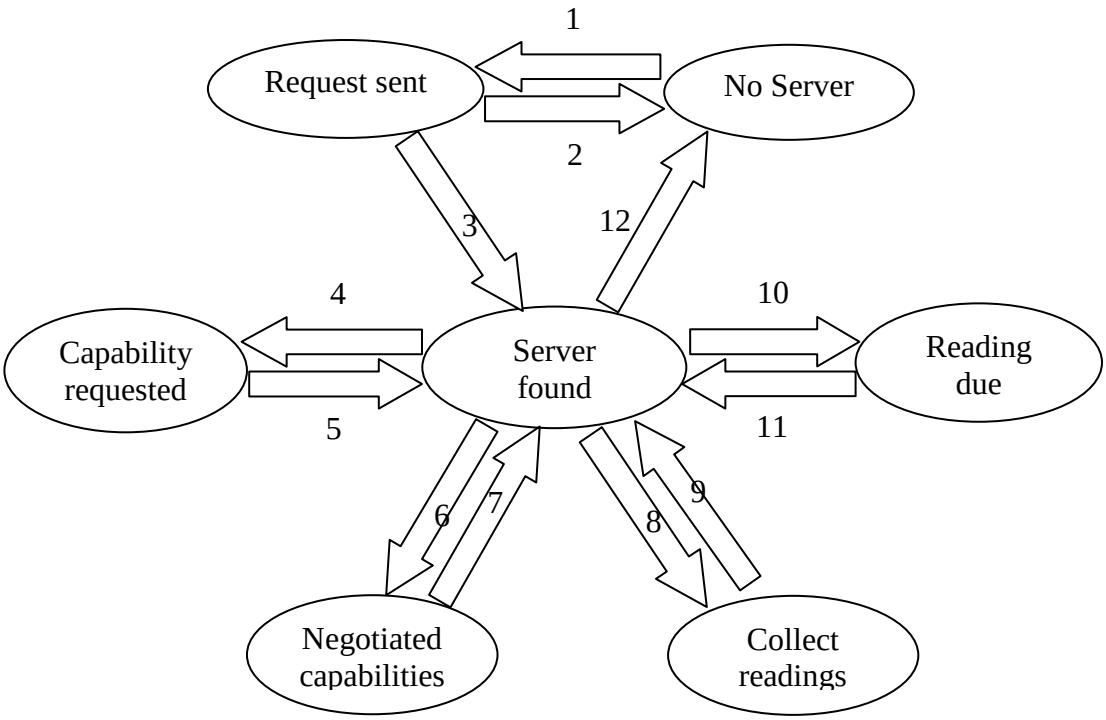

Figure 7. State diagram of the client

6.1 Client pseudo code

The pseudo code will be as follows:

constants:

CONNECT_TIMEOUT	*= 10000 milliseconds*
SERVER_FOUND	*= 1*
NO_SERVER	*= 0*

state:

my_id	//ID of the sensor node
meshNode	//Mesh node that sensor node is connected to
connectionStatus	//Connection status with the mesh node
noAckReceived	//Counter to track missed acknowledgements
s	//A particular sensor code

reading$_s$ //Reading of sensor with sensor code *s*

reading$_{s,t}$ //A 3-tuple *(s, reading*$_s$*, t)* where *reading*$_s$ of sensor with //code *s* was taken at time *t*
readings //Set of *reading*$_{s,t}$
sensors //The set of sensor codes representing all the sensors on //the sensor node
frequency$_s$ //Ordered pair (*s, frequency*) of sensor code *s* and //frequency related to sensor *s*
frequencies //Set of *frequency*$_s$
timer *connectTimer* //Connect timer that monitors the connection status with //the mesh node
timer *collectTimer*$_s$ //Collection timer for the sensor with sensor code *s*, set //by the mesh node
timer *reportTimer* //Report timer for the sensor node, set by the mesh node

initially:
{
id ← *0*
sensors ←*getAllSensors* ()
noAckReceived ← *0*
readings ← *Ø*
connectionStatus ← *NO_SERVER*
set *connectTimer* **to** *CONNECT_TIMEOUT*
}

events:
receive RENDEZVOUS_REPLY (*meshNode, id, reportFrequency*)
receive CAPABILITY_REQUEST ()
receive NEGOTIATED_CAPABILITY (*frequencies*)
receive READING_ACKNOWLEDGEMENT ()
timeout (*connectTimer*) //Triggered when *connectTimer* times out
timeout (*collectTimer*$_s$) //Triggered when *collectTimer*$_s$ times out
timeout (*reportTimer*) //Triggered when *reportTimer* times out

utility functions:
broadcast (*msg*)
{
Broadcast the message *msg* in sensor node's subnet.
}

send (*msg, meshNode*)
{

Send the message *msg* to mesh node *meshNode*.

}

getAllSensors ()
{

Get the set of all the sensor codes corresponding to all the sensors on this node.

The protocol requires that each implementation of this protocol agrees upon the sensor codes for each sensor that it envisages exchanging information about. The protocol doesn't specify a single way to accomplish this. The rationale behind not providing a rigid list of sensor codes is that we realize that such a list would not be comprehensive given the proliferation of different types of sensors and unnecessarily rigid.

As general guidelines - a particular implementation could choose to statically save all the sensor codes for sensors present on the sensor node listed in the vendor's manual.

}

restart (**timer**)
{

Restart the **timer** with the same timeout as specified before.

}

findSuportedFrequency (*sensor*)
{

Return the ordered pair of (*sensor, frequency*) i.e. $frequency_s$ where *frequency* is the frequency at which this sensor node can collect the readings from the inbuilt sensor with code *s*.

}

getReading (*sensor*)
{

Return the data of inbuilt sensor with code *sensor*.

The protocol doesn't specify a single way to accomplish this as different sensor nodes will have different ways to collect such data. In our implementation of this protocol using the Motorola Droid phones, we used the Android API as it exposes these sensors using classes as described in Chapter 3.

}

event handlers:

receive RENDEZVOUS_REPLY (*node, id, frequency*)
{
    meshNode ← *node*
    if (*id* ≠*0*)
    {
        my_id← *id*
    }
    set *reportTimer* **to** *frequency*
    deactivate *connectTimer*
    connectionStatus ← *SERVER_FOUND*
}

receive CAPABILITY_REQUEST ()
{
    for each *s* ∈ *sensors* **do**
    {
        $frequency_s$ ← (*s*, *findSuportedFrequency* (*s*))
        frequencies ←*frequencies* + {$frequency_s$}
    }
    send ([CAPABILITIES(*frequencies*)], *meshNode*)*;*
}

receive NEGOTIATED_CAPABILITY (*frequencies*)
{
    for each $frequency_s$ ∈ *frequencies* **do**
    {
        set $collectTimer_s$ **to** *frequency* | *frequency* ∈ $frequency_s$
    }
}

receive READING_ACKNOWLEDGEMENT ()
{
    noAckReceived ← *0*
}

timeout (*connectTimer*)
{
    if (*connectionStatus* = *SERVER_FOUND*)
    {
        if (*noAckReceived* < *3*)

```
                {
                        noAckReceived ← noAckReceived + 1
                }
                else
                {
                        noAckReceived ← 0
                        connectionStatus = NO_SERVER
                }
        }
        else
        {
                broadcast ([RENDEZVOUS_REQUEST (id)])
        }
        set connectTimer to CONNECT_TIMEOUT
}

timeout (collectTimer_s)
{
        reading_s ← getReading (s)
        time ←getSystemTime ()
        reading_s,t ←(s, reading_s, time)
        readings ← readings + {reading_s,t}
        restart (collectTimer_s)
}

timeout (reportTimer)
{
        send ([READING (readings)], meshNode)
        readings ← Ø
        restart (reportTimer)
}
```

At the server side the following actions prompt it to change its state as shown in Figure 10 below.

1. Server receives connection/rendezvous request.

2. On accepting the client connection, the server sends the reply to the rendezvous request, as show in Figure 8. It may also send back a unique ID for the client, if the client had requested one during the request. The server proceeds to request capabilities.

Preamble	Action code	ID	Frequency

←1 byte→ ←1 byte→ ←1 byte→ ←10 bytes→

Figure 8. Packet representing rendezvous reply by server

3. Server receives capability information regarding the various sensors present on the sensor and their corresponding supported frequency from the client.
4. Server specifies to the client what sensors it wants to read from client and at what frequency by sending the negotiated capabilities, as show in Figure 9.

Preamble	Action code	ID	Sensed quantity code	Frequency

←1 byte→ ←1 byte→ ←1 byte→ ←1 byte→ ←10 bytes→

…last two tuples repeat

Figure 9. Packet representing negotiated capabilities sent by the server

5. Server receives reading and archives them at the auxiliary storage.

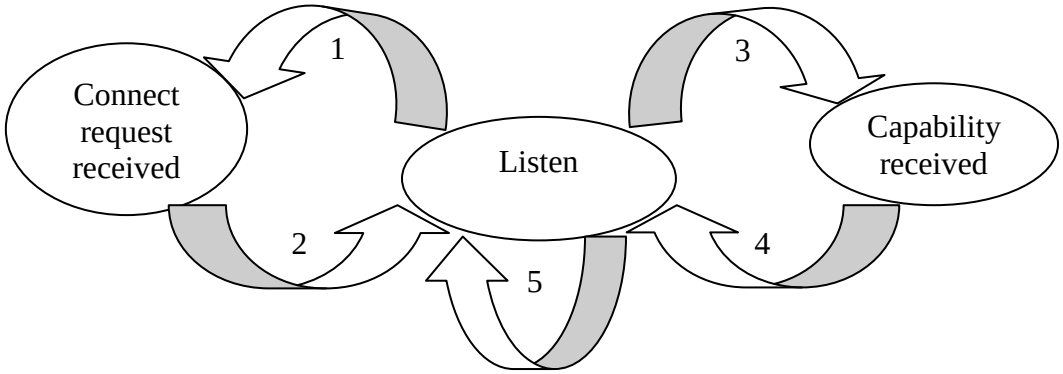

Figure 10. State diagram of the server

6.2 Server pseudo code

The pseudo code will be as follows:

constants:

MIN_ID_VALUE	//The minimum value of an ID that can be allocated by this //mesh node

state:

auxiliaryStorage	//Auxiliary storage to store the sensor readings
idCounter	//Counter to track the last allocated *id* number
allIds	//Set of all IDs that can be allocated by this mesh node. The //protocol requires this set to be unique for each mesh node i.e. //no two mesh nodes can have any overlapping IDs in their //respective sets.

initially:

{

idCounter ← *MIN_ID_VALUE*

allIds ← *getListOfIds*()
}

events:
receive RENDEZVOUS_REQUEST (*id, sensorNode*)
receive CAPABILITIES (*frequencies, sensorNode*)
receive READING (id, readings, sensorNode)

utility functions:
send (*msg, sensorNode*)
{
Send the message *msg* to sensor node *sensorNode*.
}

setId (*sensorNode*$_{id}$*, id*)
{
Set *id* as the id of the sensor node *sensorNode*$_{id}$
}

getNewId (*sensorNode*)
{
allIds ← *allIds* – {*idCounter*}
setId (*sensorNode, idCounter*)
idCounter ← *idCounter + 1;*
Return the newly assigned id (which is *idCounter)* of the *sensorNode*$_{id}$

}

getReportingFrequency (*id*)
{
Return the *frequency* at which the sensor node should report readings to the mesh node.
}

evaluate (*frequency*$_s$)
{
Find out if the readings of this sensor with code $s \mid s \in freqency_s$ are of interest. Return the *frequency*$_s$ at which the sensor node should collect readings for this sensor considering the *frequency*$_s$ the sensor has specified it can support. If this sensor is of no interest return an empty set, Ø.

As highlighted in the client pseudo code before, the protocol requires that each implementation of this protocol agrees upon the sensor code for each sensor that it envisages exchanging information about.

}

saveToAuxiliaryStorage (id, readings)
{
Save *readings* of *sensorNode* with ID *id* to the *auxiliaryStorage.*
}

getListOfIds ()
{
Return the list of all IDs that can be allocated by this mesh node. As noted before, the protocol requires no two mesh nodes can have any overlapping IDs in their respective sets. Such a set must be either statically configured or the mesh nodes should programmatically negotiate this amongst themselves. We leave it up to the discretion of the researcher to decide which approach is best for his/her setup.
}

event handlers:
receive RENDEZVOUS_REQUEST (id, sensorNode)
{
if *(id = 0)*
{
id ← *getNewId(sensorNode)*
}
frequency = getReportingFrequency (id)
send ([*RENDEZVOUS_REPLY* (*id, frequency*)], *sensorNode*)
send ([*CAPABILITY_REQUEST* ()], *sensorNode*)
}

receive CAPABILITIES (*frequencies, sensorNode*)
{
for each $frequency_s \in frequencies$ **do**
{
$frequencies \leftarrow frequencies - frequency_s$
$frequencies \leftarrow frequencies + evaluate\ (frequency_s)$
}
Send ([NEGOTIATED_CAPABILITY (*frequencies*)], *sensorNode*)

}

receive READING (*id, readings, sensorNode*)
{
	saveToAuxiliaryStorage (*id, readings*)
	send ([*READING_ACKNOWLEDGEMENT* ()], *sensorNode*)
}

We believe that capabilities of a sensor node aren't likely to change for an established connection. In its present form, our protocol doesn't support capability renegotiation for an established session. We do recognize that there is a potential for sensor capabilities, viz. power etc. to change and adding that to the protocol in the future may be beneficial.

Figure 11. Sensor readings displayed at the client

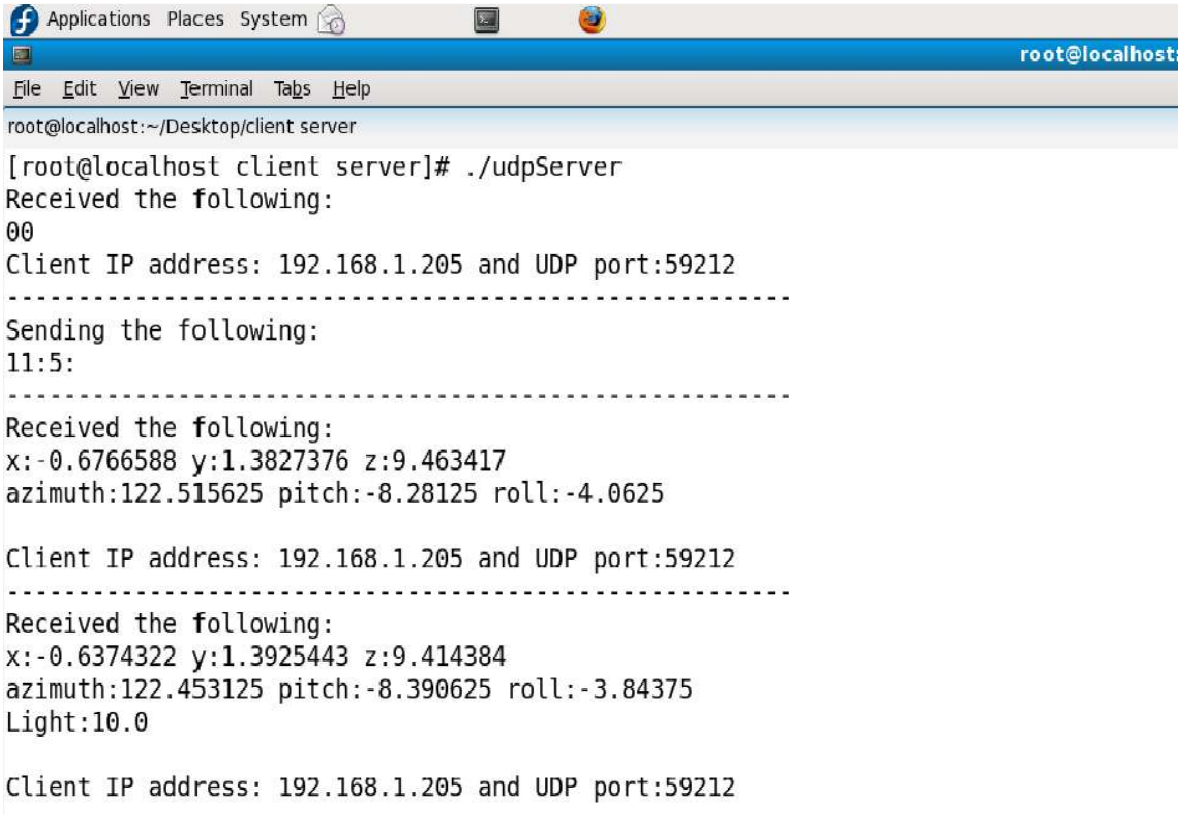

Figure 12. Sensor readings displayed at the meshnode

```
x:-1.0885382 y:1.1571847 z:9.433997
azimuth:116.96875 pitch:-6.953125 roll:-6.53125
Light:33.0

x:-1.0885382 y:1.1375713 z:9.433997
azimuth:116.5625 pitch:-6.84375 roll:-6.53125

x:-1.0787314 y:1.176798 z:9.4241905
azimuth:116.4375 pitch:-7.0625 roll:-6.484375
Light:10.0

x:-1.0689248 y:1.1669914 z:9.414384
azimuth:116.65625 pitch:-7.015625 roll:-6.421875
Light:33.0

x:-1.0689248 y:1.1964113 z:9.45361
azimuth:116.21875 pitch:-7.1875 roll:-6.390625
Light:10.0

x:-1.0689248 y:1.1866046 z:9.433997
azimuth:116.8125 pitch:-7.125 roll:-6.421875
```

Figure 13. Sensor readings archived at the auxiliary storage

Preamble:

It consists of a distinctive pattern of bits that lets the receiver know that a data frame will be transmitted [25]. It is a value unique and consistent for all packet headers of the protocol.

Action Code:

This one-octet unsigned integer represents specific patterns that help trigger corresponding actions. The following action codes have been defined:

Action Code	Event
0	Rendezvous request
1	Rendezvous reply
2	Server requests capabilities of the client and specifies reporting frequency
3	Client replies with a list of capabilities
4	Server request specifying which sensors does it want to read and at what frequency.
5	Client sends data
6	Server acknowledges receipt of data

The protocol must recognize all these action codes.

ID:

This one octet unsigned integer uniquely identifies the sensor node.

Sensed quantity code:

This one octet code helps both mesh nodes and sensor nodes identify what sensor they are referring to when they exchange relevant details on them.

Frequency:

This ten octet float specifies the frequency at which a sensor can collect the readings from its inbuilt sensors. The 80 bit floating point format provides greater precision and more exponent range which allows the protocol to specify sensing frequency as low as 1E-37 [26].

Representation:

This field is reserved for the future. This field should specify the units of the sensed data. One way would be to represent as much number of units for every sensed quantity as possible by an unsigned one-octet integer and have one particular integer value represent unrecognized unit for all sensed quantities.

Reading:

This consists of the data sensed by the sensing node and is represented as 4 byte float.

Timestamp:

The sensing node also transmits the timestamp to the packet at the time of sensing that particular sensor.

CHAPTER 7

Summary and Future Work

This work proposes a flexible and extensible interface for the CentMesh architecture to provide researchers the capability to interact with heterogeneous sensors loosely connected to the wireless mesh network. This thesis summarizes the design of a communication protocol that would allow mesh nodes to communicate with the auxiliary sensors and retrieve the measurements. We highlight the various design decisions we made during this work and reasons for our choices. We envisage that in the future this framework would include the security policies by having authentication and authorization mechanisms. We hope that other types of sensors can also be integrated to CentMesh using this framework.

REFERENCES

[1] J. White, G. Jourjon, T. Rakotoarivelo, and M. Ott. Measurement architectures for network experiments with disconnected mobile nodes. In *TRIDENTCOM'10*, pages 315–330, 2010.

[2] A. Hanemann, J. W. Boote, E. Boyd, J. Dur, L. Kudarimoti, R. Lapacz, D. M. Swany, S. Trocha, and J. Zurawski. Perfsonar: A service oriented architecture for multi-domain network monitoring. In *In Proceedings of the Third International Conference on Service Oriented Computing (ICSOC 2005). ACM Sigsoft and Sigweb*, 2005.

[3] J. Lim, P. H. Pathak, M. Pandian, U. Patel, G. Deuskar, A. Danivasa, M. L. Sichitiu, and R. Dutta. Centmesh: Modular and extensible wireless mesh network testbed. In *TRIDENTCOM'10*, pages 619–621, 2010.

[4] S. Graham and P. Niblett. Publish-Subscribe Notification for Web services. In IBM *Germany Scientific Symposium Series*, 2004.

[5] SensorEvent. http://developer.android.com/reference/android/hardware/SensorEvent.html#values. Accessed July 31, 2012.

[6] SensorManager http://developer.android.com/reference/android/hardware/SensorManager.html. Accessed July 31, 2012.

[7] Position sensors. http://developer.android.com/guide/topics/sensors//sensors_position.html. Accessed July 31, 2012.

[8] D. J. Cook and S. K. Das. How smart are our environments? an updated look at the state of the art. *Pervasive Mob. Comput.*, 3(2):53–73, Mar. 2007. ISSN 1574-1192. doi: 10.1016/j.pmcj.2006.12.001. URL http://dx.doi.org/10.1016/j.pmcj.2006.12.001.

[9] G. Held. Wireless Mesh Networks. Auerbach Publications, Boston, MA, USA, 2005. ISBN 0849329604.

[10] I. F. Akyildiz, X. Wang, and W. Wang. Wireless mesh networks: a survey. *Comput. Netw. ISDN Syst.*, 47(4):445{487, Mar. 2005. ISSN 0169-7552. doi: 10.1016/j.comnet.2004.12.001. URL http://dx.doi.org/10.1016/j.comnet.2004.12.001.

[11] C. M. Oh, H. J. Kim, G. Y. Lee, and C. K. Jeong. International journal of future generation communication and networking 59 a study on the optimal number of interfaces in wireless mesh network.

[12] G. Jourjon, T. Rakotoarivelo, and M. Ott. A portal to support rigorous experimental methodology in networking research. In *7th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom)*, page 16, Shanghai/China, April 2011.

[13] Goldman, Jeff; Ramanathan, Nithya; Ambrose, Richard F; Caron, David; Estrin, D; Fisher, Jason; et al.(2007). Distrbuted Sensing Systems for Water Quality Assesment and Management. UC Los Angeles: Center for Embedded Network Sensing. Retrieved from: http://escholarship.org/uc/item/5v7619xw

[14] M. Farhadiroushan (2001) *Distributed sensing system*. US Patent: US6285446

[15] G. Pottie. Wireless sensor networks. In Information Theory Workshop, 1998, pages 139 –140, jun 1998. doi: 10.1109/ITW.1998.706478.

[16] M. Singh, M. Ott, I. Seskar, and P. Kamat. Orbit measurements framework and library (oml): Motivations, design, implementation, and features. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities*, TRIDENTCOM '05, pages 146–152, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2219-X. doi: 10.1109/TRIDNT.2005.25. URL http://dx.doi.org/10.1109/TRIDNT.2005.25.

[17] C. Dwertmann, E. Mesut, G. Jourjon, M. Ott, T. Rakotoarivelo, and I. Seskar. Mobile experiments made easy with omf/orbit. SIGCOMM, Aug 2009.

[18] Omf. http://mytestbed.net/. Accessed July 31, 2012.

[19] MadWifi. http://madwifi-project.org/wiki/About/MadWifi. Accessed July 31, 2012.

[20] Motorola Droid care https://motorola-global-portal.custhelp.com/app/answers/detail/a_id/68743/~/android---wireless-ad-hoc-networks. Accessed July 31, 2012.

[21] E. Royer and C.-K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *Personal Communications, IEEE*, 6(2):46-55, apr 1999. ISSN 1070-9916. doi: 10.1109/98.760423.

[22] Linux on the fujitsu lifebook s2110. http://projects.sentinelchicken.org/howtos/ Fujitsu_s2110?PRINTABLE=true#Wireless. Accessed July 31, 2012.

[23] CentMesh. http://centmesh.csc.ncsu.edu/. Accessed July 31, 2012.

[24] Droid By Motorola, A855. http://developer.motorola.com/products/droid/. Accessed July 31, 2012.

[25] A. Bachir, M. Heusse, and A. Duda. Preamble mac protocols with non-persistent receivers in wireless sensor networks. In *Proceedings of the 7th international IFIP-TC6 networking conference on AdHoc and sensor networks, wireless networks, next generation internet*, NETWORKING'08, pages 36-47, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-79548-0, 978-3-540-79548-3. URL http://dl.acm.org/citation.cfm?id=1792514.1792520.

[26] float.h. http://www.acm.uiuc.edu/webmonkeys/book/c_guide/2.4.html. Accessed July 31, 2012.