Sustainable Communication Networks
University of Bremen
Prof. Dr. Anna Förster

Master Mini Project

Implementation of the Epidemic routing protocol

of

Leonardo Sarmiento

Matriculation Number: 3110474

Bremen, 9th Jan. 2019

Supervised by:
Prof. Dr. Anna Förster
PhDc Vishnupriya Kuppusamy

I assure, that this work has been done solely by me without any further help from others except for the offical attendance by the Chair of Sustainable Communication Networks. The literature used is listed completely in the bibliography.

Bremen, 9th Jan. 2019

(Leonardo Sarmiento)

ABSTRACT

There is a gap between the theory behind the epidemic routing protocol and the hardware implementation. This project closes the gap between both of them using of a Raspberry-Pi board. It also offers a framework to record the device activities during experiments. The records are used to reproduce the same experiments in Omnet++ and they are also used to extract features. These features can be later used to create a more realistic epidemic protocol model.

CONTENTS

Abstract 1

1 Introduction 5

2 Algorithm 7
- 2.1 Operation modes . . . 10
  - 2.1.1 Operation mode 1 . . . 10
  - 2.1.2 Operation mode 2 . . . 10
  - 2.1.3 Common operation mode . . . 11
- 2.2 States . . . 11
- 2.3 Signals . . . 12
- 2.4 Record events . . . 13

3 Node components 15
- 3.1 Hardware . . . 15
- 3.2 Software . . . 15

4 File system 17
- 4.1 Source files . . . 17
- 4.2 Configuration files . . . 17
- 4.3 Message storage . . . 17
- 4.4 Initial messages . . . 17
- 4.5 Log files . . . 17
- 4.6 Start up script . . . 18

5 Results and Conclusions 19
- 5.1 Data representation . . . 19
  - 5.1.1 Graphical representation . . . 19
  - 5.1.2 Numerical representation . . . 20
    - 5.1.2.1 Events . . . 20
    - 5.1.2.2 Processes . . . 20
- 5.2 Time distance relation . . . 21
- 5.3 Future Work . . . 23

List of Figures 25

List of Tables 27

Bibliography 29

CHAPTER 1

Introduction

The epidemic routing protocol is a delay tolerant protocol designed to forward messages without building a path between the origin node and the destination node first. This protocol can be used as a backup protocol in case the primary routing protocol needs a centralized infrastructure and that infrastructure is out of service for any reason. The implementation of the protocol is based on the academic paper entitled "Epidemic Routing for Partially-Connected Ad Hoc Networks" (2000) by Amin Vahdat and David Becker [1].

In this implementation when two nodes meet, one node (A) sends the headers of the messages that it already has in the memory (Summary vector), then node (B) compares node's A headers with its own headers, node B sends the messages that are unknown to A, then B sends its summary vector to A, and Finally A sends the messages that are unknown to B.

This report is structured in the following way, first it starts explaining the algorithm behavior by describing the operation modes, the states, and the signals; Then, it mentions the hardware and software that compose the node. Later on, it explains the framework used to record results it the node, it describes how the files are organized in the operating system; It explains the experiments and the scenarios, and finally, it presents the results from the experiments.

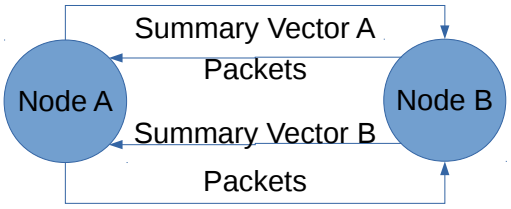


Figure 1.1: Communication process between 2 nodes

CHAPTER 2

Algorithm

The algorithm is represented with state diagrams.

The first state diagram describes the mechanism used to select other nodes in the environment for communication.

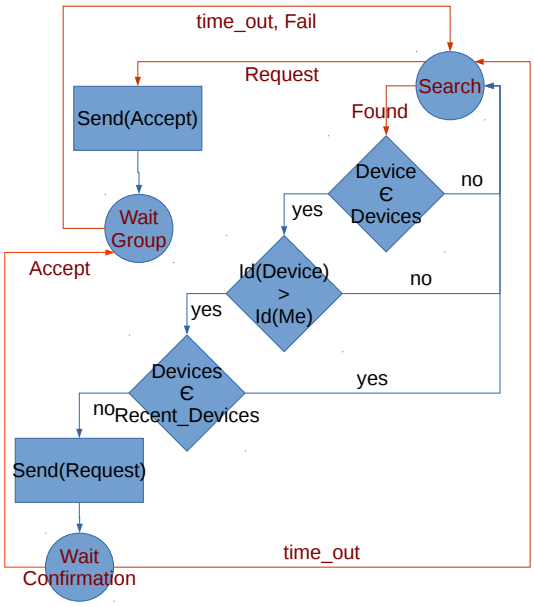


Figure 2.1: First UML State diagram

The second state diagram shows the processes involved in the epidemic protocol after the node's selection. The processes are connecting 2 nodes, exchanging messages between them, and disconnecting the 2 nodes.

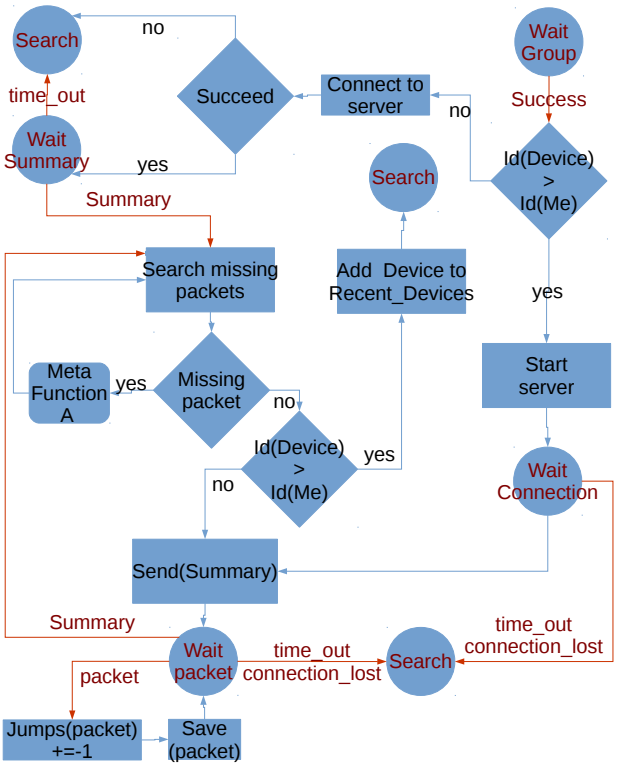


Figure 2.2: Second UML State diagram

The second state diagram contains a meta function (A). The meta function describes the logic for sending packets.

The node sends a packet if the final destination of the packet is not the node itself, and if the packet's attribute called Jumps is bigger than 1 or if the packet's final destination is the connected device.

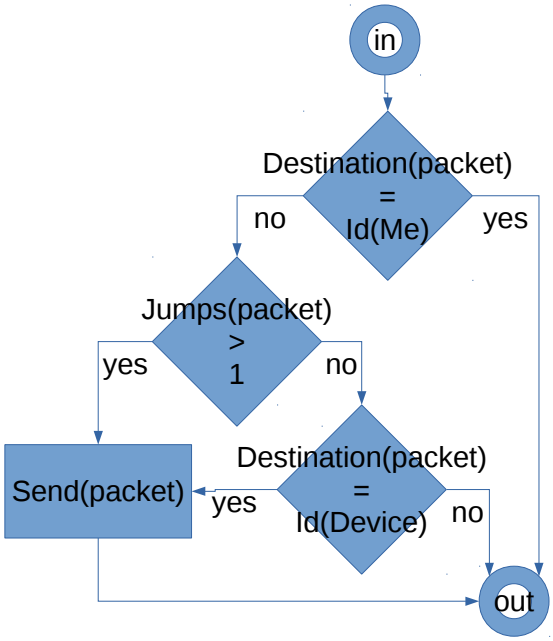


Figure 2.3: Flow chart of meta function A

2.1 Operation modes

The behavior of the algorithm can be analyzed with 2 operation modes.

2.1.1 Operation mode 1

The node starts in Search state until it receives a Found signal.

When the node receives a Found signal it analyzes the information of the discovered device that is attached to that signal, looking for 3 conditions:

1. If the name of discovered device contains the string "EPI".
2. If the device identifier is bigger than the identifier of the node that received the signal.
3. If the device is not registered in the Recent Devices list.

If those 3 conditions are met, the node sends a Request signal and it changes its state to Wait confirmation. In any other case the node will return to Search state.

In order to find nodes and to create a network between 2 nodes, The algorithm sends shell commands to the Wi-Fi direct controller (Linux WPA -Supplicant) [2].
Signals and events related with Wi-Fi direct are published in the Inter-process communication bus: (Linux D-Bus). Therefore, the algorithm listens to the Linux D-Bus [3].

In the Wait confirmation state, if the node receives an Accept signal from the same device from which the node previously sent a request signal, the node changes to Wait group state.

In the Wait group state the node creates a Wi-Fi network and then it waits for the device that sent the accept signal to join this network. If it succeeds, the node starts a TCP-IP server and it waits for the client to get connected (Wait connection state).

In the Wait connection state if the TCP connection succeeds, the server (node) sends a summary vector to the client (connected device). Then the server would save any packages sent by the client (Wait packet state).

Every time a packet is received, the value of the packet's attribute called Jumps is reduced by 1. In the Wait packet state if the server receives a Summary vector from the client, the server searches for missing packets in the received Summary vector.

The node (server) sends a packet by following the logic described in the meta function A. After sending the last packet the server registers the connected device in the Recent Devices list and it changes its state to Search closing the TCP service and the Wi-Fi network.

2.1.2 Operation mode 2

The node starts in Search state until it receives a Request signal. When the node receives a Request signal, it sends an accept signal and it changes its state to Wait group.

In the Wait group state the node will try to connect to a Wi-Fi network created by the device that sent the Request signal. If it succeeds the node waits a given time and then try to associate to a TCP service created by the other device in the Wi-Fi network.

After that, the node waits for a Summary vector, it searches for missing packets, and sends the packets following the same logic as in Operation mode 1 (Wait packet state).

After sending the last packet the node sends its Summary vector. Then the node saves any packets sent by the server following the same logic as in Operation mode 1 (Wait packet state).

Finally, when the node detects that the server got disconnected, the node changes its state to Search.

2.1.3 Common operation mode

If any of the mentioned operations in both modes fail, the node changes its state to Search closing any open service or connection.

The Recent Devices list removes entries that are older than a threshold time.

2.2 States

When a node enters in a state, it executes operations called Enter actions; and when it leaves the state, it executes operations called Exit actions.

The implementation is based in a state machine with the 6 following states.

- Search
  - Enter actions
    - It closes any active TCP service active.
    - If the device belongs to a P2P group. It disintegrates the P2P group.
    - It starts the neighbor's discovery mechanism provided by Wi-Fi direct.
  - Exit actions
    - It stops the neighbor's discovery mechanism.
    - It removes from a table (Recent visited) the devices that are older than a time threshold.
- Wait Packets
  - Enter actions
    - Search for packets older than a threshold time and remove them from the memory.
    - If the number of packets in memory overpass a threshold, the device removes the oldest packet.
    - It resets the time in a time counter.
    - It starts the time counter (if the time counter overpass a threshold, a time out event is generated).
  - Exit actions
    - It stops the time counter.
- Wait Confirmation, Wait group, Wait Connection, and Wait Summary
  - Enter actions
    - It resets the time in a time counter.
    - It starts the time counter.
  - Exit actions
    - It stops the time counter.

2.3 Signals

The mode changes its state when an event (signal) is detected.

Some signals are generated and used inside the node (internal signal), other signals are generated by a neighbor device and transmitted via Wi-Fi (external signals).

The algorithm is composed by the following 9 main signals.

- Time out
  It is an internal signal generated to avoid the node getting stuck in a state indefinitely.
- Connection lost
  It is an internal signal generated when the TCP session fails.
- Fail
  It is an internal signal generated when the node is not able to establish a connection with Wi-Fi direct.
- Success
  It is an internal signal generated after the node has established a connection with Wi-Fi direct.
- Request and Accept
  They are external signals generated as a hand shake previous to the Wi-Fi direct group formation.
- Packet
  It is an external signal that includes the content of a communication packet.
  Each packet has 2 parts, the content and the header.
  - Packet header
    The packet's header is built with the concatenation of its attributes.
    The packet's attributes are destination node (d), origin node (o), packet index (i), number of left Jumps (j), and packet size (s).
    For example, the message file 1d2o8i5j10s was generated by the node 2, the destination node is 1, its index is 8, it can only be forwarded 5 times more, it has 10 bytes of information.
- Summary (Summary vector)
  It is an external signal that contains a list of packet headers.

2.4 Record events

In order to analyze the results each node records several events during an experiment. The records are saved as a table.
The first column is the time in seconds (Hour*3600+Minutes*60+Seconds), the second column is the event index, and the third column is the neighbor node index related with that communication event (-1 means that no neighbor node is related with the event).

index	Event
0	Device found
1	Request connection
2	Accept request
3	Ignore request
4	Connected as Operation mode 1
5	Connected as Operation mode 2
6	Connection Failed
7	Completed communication
8	Interrupted communication
9	Disconnected
10	Time out
11	Device erased from Recent visited list
12	Packet sent
13	Packet received
14	Delete packet
15	Start algorithm

Table 2.1: Recorded events

]

The nodes also record periodically the GPS positions in a different table. The first column is the time in seconds, the second column is the East longitude, and the third column is the North latitude [4].

CHAPTER 3

Node components

Each node is built in a single board computer with capabilities for wireless communications.

3.1 Hardware

1. Embedded board: Raspberry pi 2
2. GPS receiver module: NAVILOCK NL-602U
3. Wireless module: EDIMAX EW-7811Un
4. Power-bank: X-DRAGON XD-PB-014 24000mAh

One of the nodes had additional hardware to look into errors during the experiments.

1. HDMI screen: WAVESHARE 7inch HDMI LCD
2. Keypad: USB Numeric Keypad, Jelly Comb N001
3. Extra Power-bank for HDMI screen: Goobay 71599PocketPower 2.0

3.2 Software

1. Operating system: Arch Linux OS for ARM [5].
2. Programing language: Python v2.7
3. Inter-process communication bus: Linux D-Bus [6].
4. Wi-Fi direct controller: Linux WPA -Supplicant [7].

Figure 3.1: Left: Node with debug capabilities. Right: Node without debug capabilities.

CHAPTER 4

File system

The epidemic implementation is organized with the following file system.

4.1 Source files

The `$~alarm/epidemic/src` directory contains the source code of the implantation. The usbgps.py file records the mobility track of the node and synchronize the time of the operating system.
The event_record.py file records all the communication events of the node. The file p2p_events.py is in charge of matching 2 nodes and establish the connection between them with Wifi direct.
The files eclient.py and eserver.py are in charge of the exchange of summary vectors and messages between 2 nodes with TCP sockets.

4.2 Configuration files

The folder `$~alarm/epidemic/p2pfiles` contains two configuration files.
The p2p.conf file contains the parameters related with the configuration of wpa supplicant in Linux and other parameters like the name and index of the device.
The epidemic.xml file describes parameters related with the epidemic protocol such as the time (in seconds) that a node remains in the Recent visited list (peer life), the length of the Recent visited list (visit limit) and the maximum number of messages that a node can store (maxpkj). It also describes parameters related with the implementation algorithm like the threshold time in seconds for the timeout signal (refresh), the buffer size of the TCP sockets (buffer), and the delay in seconds between the client and the server for the TCP sockets (synchronization).

4.3 Message storage

The messages (packets) are stored as files in two different folders.
The `$~alarm/epidemic/efiles` directory contains packets that the node has to forward.
The `$~alarm/epidemic/myfiles` directory contains packets that do not need to be forwarded.
Each message file includes the header in its name.

4.4 Initial messages

The folder `$~alarm/epidemic/initialfiles` includes messages that are going to be loaded in the folder `alarm/epidemic/efiles` at the beginning of each experiment.

4.5 Log files

The node can record time stamps, GPS positions, and communication events automatically. All the records are saved in the directory `$~alarm/epidemic/logs`.

Inside the logs folder there are different experiments. The past experiments are saved in folders with different time stamps. For example, `$~alarm/epidemic/logs/2018-09-14_20-52-35`. The current experiment is saved in the directory `alarm/epidemic/logs/log`.
The log folder contains contains 2 files. The first one is the communication events with the time in seconds of each event, and second file is the record of GPS positions.

4.6 Start up script

The script `$~alarm/1.0` configures the log files, resets the message storage, and starts the application by loading the file `$~alarm/epidemic/src/p2pevents.py`. Therefore, it is possible to start a new experiment by rebooting the operating system without losing the records of previous experiments.
For nodes without debug capabilities, the Start up script is called automatically at the start of the operating system. On the other hand, for nodes with debug capabilities, the Start up script should be called manually using a keyboard.

CHAPTER 5

Results and Conclusions

There were 3 completed experiments. In each experiment there were 3 nodes, 2 were static (B,C) and one node (A) was moving from B to C and, then from C to B, and finally from B to C again.
Node B started with 100 messages for C, node C started with 100 messages for B. Each message had 100 bytes of data.
The moving node has debug capabilities, but the static nodes do not have it.
The distance between B and C was 149 meters for the first experiment, for the second experiment 251 meters, and for the third one 176 meters. B and C were at line of sight in all of the experiments.

5.1 Data representation

It was designed the 2 following methodologies to represent the data recorded from each experiment.

5.1.1 Graphical representation

Using the log files for position and events a framework was designed to reproduce the experiments in Omnet++.
The framework is useful to find discrepancies in the implementation of the epidemic routing protocol and to see the experiments from a different perspective.

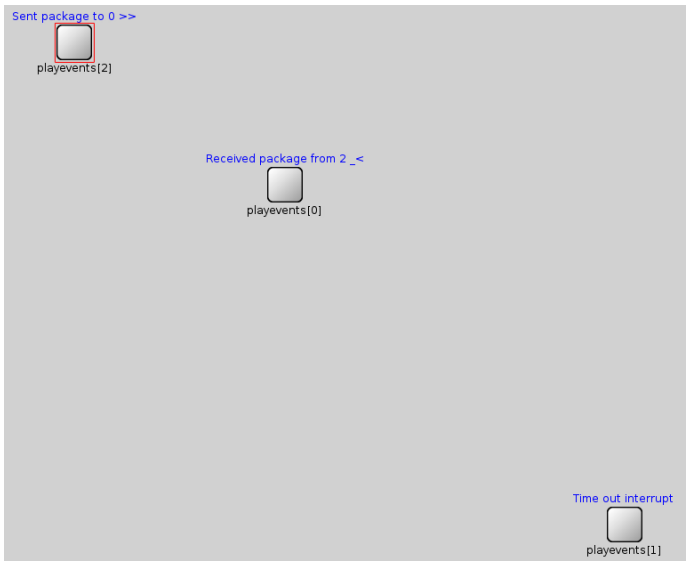


Figure 5.1: Graphical representation of an experiment

5.1.2 Numerical representation

The log files for position and events are used to extract features that can be used to analyze the behavior of the implementation of the routing epidemic protocol.
The 2 following types of features were extracted from the log files.

5.1.2.1 Events

There were 4 events analyzed.

1. Request
   It is when a node sends a request signal to another node.
2. Accept
   It is when a node accepts the connection request from another node.
3. Established
   It is when a node has sent or received the first packet after establishing a connection with another node.
4. Packet
   It is when a node sends o receives a packet.

The occurrence possibility of any of the 5 events depends on the Wi-Fi antenna coverage, and the lower layers protocols. Therefore, the events are limited by a radius or a maximum distance as is shown in the following table.

Event	Maximum distance (m)
Request	251.66
Accept	251.66
Established	71.22
Packet	87.65

Table 5.1: Coverage distance for each event

]

To find the geographical position of each event, the positions log file is interpolated. Then, to find the distance between the nodes that are involved in the event, the Haversine Formula is used [8].

5.1.2.2 Processes

Each one of the following processes is a set of operations between events.

1. Start
   It is the process between the events Request and Established.
2. Exchange
   It is the process between the events Request and Established.
3. Stop
   It is the process of closing the connection after sending or receiving the last packet.

As is described in the following table, each process needs a different amount of time to complete its tasks.

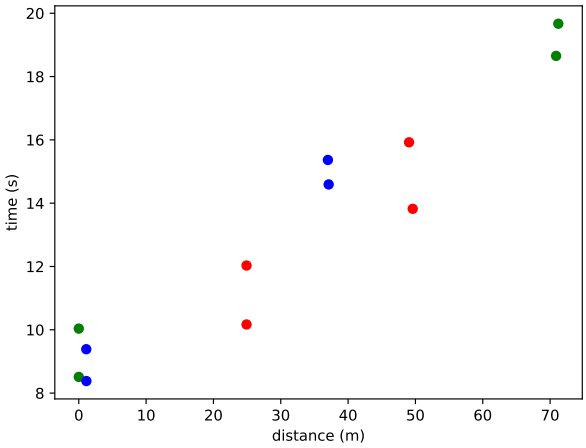
Process	Minimum time (s)	Maximum time (s)	Mean time (s)
Start	8.38	19.669	13.045
Exchange	0.071	3.794	0.301
Stop	5.694	11.342	9.741

Table 5.2: Times needed by each process

]

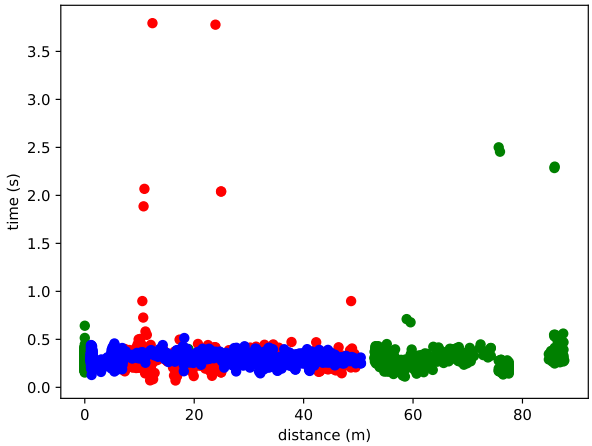
5.2 Time distance relation

In the Figure 5.2 can be seen that the time needed to complete the start process has a relation with the distance between nodes. On the other hand, according to the Figure 5.3 the time needed to complete the Exchange process seems to be independent of the distance.



Green: first experiment
Red: second experiment
Blue: third experiment

Figure 5.2: Start process



Green: first experiment
Red: second experiment
Blue: third experiment

Figure 5.3: Exchange process

Even if the time seems to be independent of the distance for the Exchange process, the Figure 5.4 shows that the time does not follow a random uniform distribution.

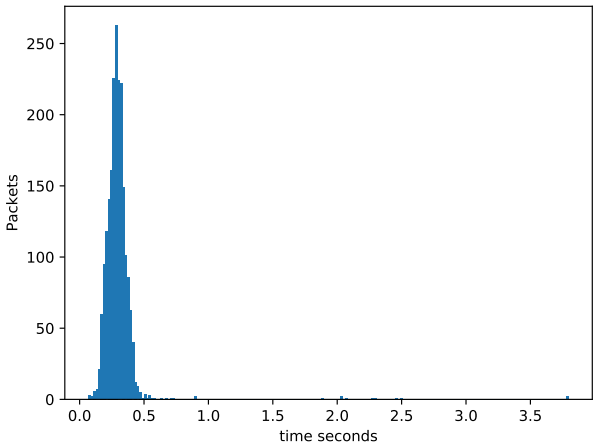


Figure 5.4: Time histogram of the Exchange process

5.3 Future Work

Based on the results and conclusions obtained by this implementation, the following tasks are proposed:

- Design a node's model that represents this routing protocol implementation.
- Implement the model in the simulator Omnet++.
- Use the implemented model with more complex scenarios.

LIST OF FIGURES

1.1 Communication process between 2 nodes 5

2.1 First UML State diagram 7
2.2 Second UML State diagram 8
2.3 Flow chart of meta function A 9

3.1 Left: Node with debug capabilities. Right: Node without debug capabilities. 15

5.1 Graphical representation of an experiment 19
5.2 Start process 21
5.3 Exchange process 22
5.4 Time histogram of the Exchange process 22

LIST OF TABLES

2.1 Recorded events 13

5.1 Coverage distance for each event 20
5.2 Times needed by each process 21

BIBLIOGRAPHY

[1] A. Vahdat, D. Becker *et al.*, "Epidemic routing for partially connected ad hoc networks," 2000.

[2] "Wi-Fi Direct p2p module," https://w1.fi/wpa_supplicant/devel/p2p.html, accessed: 2019-01-03.

[3] "Wpa Supplicant dbus api," https://w1.fi/wpa_supplicant/devel/dbus.html, accessed: 2019-01-03.

[4] "Arch Linux wpa supplicant," https://hemispheregnss.com/gnssreference/GPGGA.htm, accessed: 2019-01-03.

[5] "Arch Linux arm," https://archlinuxarm.org/platforms/armv6/raspberry-pi, accessed: 2019-01-03.

[6] "GNSS gpgga message," https://wiki.archlinux.org/index.php/D-Bus, accessed: 2019-01-03.

[7] "Arch Linux wpa supplicant," https://wiki.archlinux.org/index.php/WPA_supplicant, accessed: 2019-01-03.

[8] C. C. Robusto, "The cosine-haversine formula," *The American Mathematical Monthly*, vol. 64, no. 1, pp. 38–40, 1957.