DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Information Sharing with Floating Content using Wi-Fi Access Points

Ghulam Nasir

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Information Sharing with Floating Content using Wi-Fi Access Points

Einsatz von Floating Content mit Wi-Fi-Routern zur Verteilung von Inhalten

Author:	Ghulam Nasir
Supervisor:	Prof. Dr.-Ing. Jörg Ott
Advisor:	Prof. Dr.-Ing. Jörg Ott
Submission Date:	Submission date

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Submission date Ghulam Nasir

Acknowledgments

Abstract

Contents

1 Introduction

1.1 Motivation

1.2 Research Questions/Goals

Several questions/goals were explored in this work to weigh different extensions to the ONE Simulator such as adding access point functionality as well as snapping to access points etc. We have formulated these goals/research questions below :

Q1: Modeling a city and exploring the impact of access points on the performance of geo-based information sharing.
The ONE Simulator currenly models Helsinki City and doesn't have any access points feature. The main idea here is to Model Munich City (especially the central part), Implement Access Points and then explore the impact of the access points on the geo-based information sharing performance using ONE Simulator. It also involves mapping publicly available Access Points on the Munich Model City Map.

Q2: How does snapping to access point affect the availabilty of a floating message in a Delay Tolerant Network (DTN)?
Snapping to Access Point is the concept of increasing the availability of the message beyond its original availability zone. The idea is to change the center of the message when it is encounters an access point. However, we do limit the number of access points a message can snap to. After implementation of snapping to access point, we will test how does the snapping to different number of access points affect the availability of the floating message in a DTN.

2 Literature Review

2.1 Geo-based Content Sharing

In typical networks (such as social networking etc.), the content is shared among users irrespective of their geographic location. There are a number of advantages to this such as:

- High availablity
- Reliability
- large amount of content.

However, there are some disadvantages with such networks such as:

- A connected network is needed for the services to work, so in case the connectivity is lost, normally the content is also gone.
- The system/network is prone to censorship

Geo-based content sharing has been introduced to make the information/content more customized based on the user location such as chatting with people nearby, posting in a local message board etc. There are two types of networks used for Geobased content sharing: *Connected Systems/Networks* and *Peer to Peer Systems/Networks*.

2.1.1 Connected Systems/Networks

TODO: Write about a few connected networks apps

2.1.2 Peer to Peer Systems/Networks

Peer to Peer Networks don't relay on the network infrastructure to relay information, so they can work even when there is No server/central controller or no pre-determined path between the sender and reciever. A key property of such systems is that they do not rely on infrastructure nodes or cloud services to ensure data availability but

rather replicate content items within the anchor zone among mobile nodes in a device-to-device (peer-to-peer) fashion. While this operation does not require infrastructure network access—and thus limits dependencies as well as vulnerability to third party actions such as traceability or censorship—it comes at the cost of unpredictability: there is no guarantee that content "posted" to an anchor zone will remain available. We refer to this property as best-effort (probabilistic) content sharing [Jör17].

Below are the four different systems utilizing this methodology:

- Hovering Information
- Locus
- Ad Loc
- Floating Content

Hovering Information [CSK09]

Hovering information is a concept characterising self-organising information responsible to find its own storage on top of a highly dynamic set of mobile devices. The main requirement of a single piece of hovering information is to keep itself stored at some specified location, which we call the anchor location, despite the unreliability of the device on which it is stored. Whenever the mobile device, on which the hovering information is currently stored, leaves the area around the specified storage location, the information has to hop - "hover" - to another device.

Locus [TCK10]

Locus, a location-based data overlay for DTNs. Locus keeps objects at specific physical locations in the network using whatever devices currently are nearby. Nodes copy objects between themselves to maintain the locality of data. Location utility functions prioritize objects for replication and enable location-based forwarding of data look-ups. As a first-of-its-kind application, Locus is compared against other possible replication policies and shown to achieve query success rates nearly 4 times higher than other approaches.

Ad Loc

Ad Loc is a platform that enable users to tie persistent virtual notes to physical locations without the need for embedded infrastructure in the environment or access to the internet [CC06].

Floating Content

Floating content is an ephemeral content sharing service, solely dependent on the mobile devices in the vicinity using principles of opportunistic networking. The net result is a best effort service for floating content in which: 1) information dissemination is geographically limited; 2) the lifetime and spreading of information depend on interested nodes being available; 3) traffic can only be created and caused locally; and 4) content can only be added, but not deleted [Hyy+11]. Floating Content is discussed in more details *later in the chapter*.

2.2 The ONE Simulator

2.2.1 Introduction

The ONE is an Opportunistic Network Environment simulator [Jöra] which provides a powerful tool for generating mobility traces, running DTN messaging simulations with different routing protocols, and visualizing both simulations interactively in real-time and results after their completion [Ari].

Delay-tolerant Networking (DTN) enables communication in sparse mobile ad-hoc networks and other challenged environments where traditional networking fails and new routing and application protocols are required. Past experience with DTN routing and application protocols has shown that their performance is highly dependent on the underlying mobility and node characteristics [KOK09]. The basic idea behind DTN is to communicate data/messages from the source to the destination without having complete network connectivity. The path from the source to the destination is unknown due to the unconnected nature of the network. In such networks, each node store the messages it receive and forward it to the next node it comes in contact with. The message may or may not reach its final destination.

The ONE simulator was originally developed in Aalto University in the SINDTN [Jörb] and CATDTN projects supported by Nokia Research Center (Finland). It is now maintained in by Aalto Univerity in cooperation with Technical University of Munich.

The ONE Simulator has the following capabilities:

- Message Generation at different intervals.
- Node Movement using different movement models such as Random Walk, Map based movement etc.
- Routing messages between nodes using different types of routers such as Epidemic Router, First Contact Router etc.
- Graphical User Interface (GUI) for visualizing nodes movement, message creation and message passing in real time.
- Report Generation for different purposes such as Message Delivery Report etc.
- Batch mode where the app can be run without a GUI.

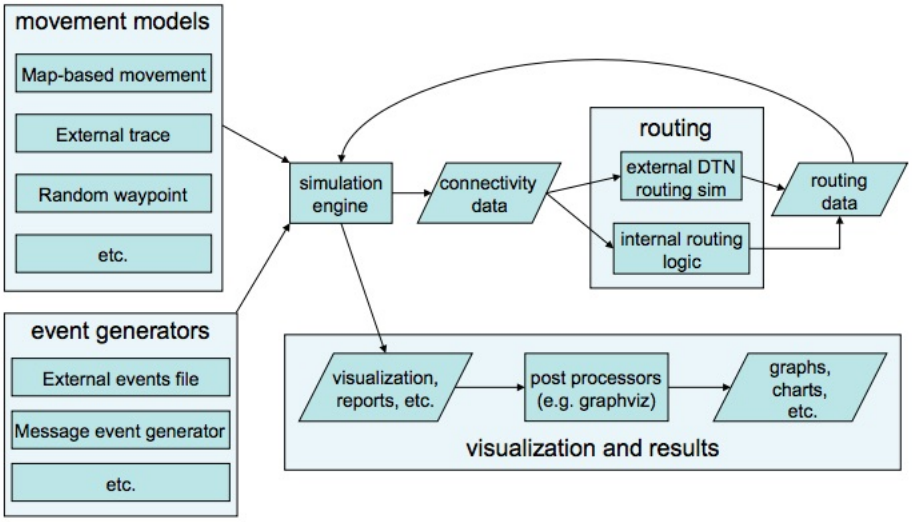

Figure 2.1: Overview of the ONE simulation Environment [KOK09]

2.2.2 Components

Host Group

Host Groups represents a collection of hosts with the same configuration such as Movement Model etc. In other words, similar nodes are grouped in a Host Group. Theoretically, each host group must have different configuration from the other host groups. All the host groups can also share configuration (such as number of hosts etc.) which are defined under "Group" namespace. However, configuration inside the specific host group takes precedence over the global "Group" namespace.

Movement Models

Movement Models dictate the node movements in the simulation. A number of Movement Models come packaged with the ONE simulator such as *Map based Movement*, *Shortest Path Map based Movement*, *Random Waypoint*, *External Movement* and *Stationary Movement*. *MapBased Movement* have further types of movements such as *Car Movement* and *Map Route Movement* etc.

Routing

Routing Module is responsible for handling the transfer (routing) of messages between hosts. ONE simulator one passive and a number of active routers. Passive router is used for interacting with other DTNs or dummy nodes. The active routers implements the well known routing algorithms for DTN such as *Epidemic*, *Spray and Wait*, *First Contact*, *Direct Delivery*, *Prophet*, *Prophet V2*, *Life*, *Wave*, *MaxProp* and *Floating Content*.

2.2.3 Compilation

ONE can be compiled from the source code using both Command Line and Eclipse.

Command Line

The ONE source contains *compile.sh* and *compile.bat* for compilation on the Linux/Unix and Windows Platform respectively.

Eclipse

The project has three main dependencies (*DTNConsoleConnection.jar*, *ECLA.jar* and *JUnit*) that need to be added.

- Import the Project into Eclipse
- Right Click on Project -> Properties -> Java Build Path -> Libraries.
- Click on Add Jar -> Navigate to lib -> Select *DTNConsoleConnection.jar* -> OK.
- Repeat for *ECLA.jar*
- Click on Add Library -> JUnit -> (Optionally) Select JUnit version -> Finish.

2.2.4 Running

Just like Compilation, ONE can be run from both Command Line and Eclipse.

Command Line

```
./one.sh [-b repetitionCount] [configuration-files]
./one.bat [-b repetitionCount] [configuration-files]
```

-b is an optional parameter. Start the ONE simulator in batch (non-UI) mode. *repetitionCount* specifies the number of times the simulations needs to repeat. If not defined, the ONE simulator starts in UI mode.
configuration-files is an optional parameters, where you can specify a number of configuration files separated by space. The simulation parameters are read from these files. Files specified later override values in the earlier config files.

Eclipse

By default, Eclipse launches the ONE simulator in UI mode. However, we can launch the simulator in batch mode and provide configuration files as well.

- Click on Run Menu -> Run Configurations...
- Click on DTNSim in the left pan -> Click on Arguments tab in the right pane
- Specify the arguments under Program arguments the same way as you would do for Command Line.
- Click on Apply -> Run.

2.2.5 Configuration

One of the amazing features of ONE is that there is no need for changing the code and recompilation if need to change any configuration (such as changing the number of hosts, changing the movement models etc.). All of these can be done in the configuration files which are ordinary text files. By default, the ONE simulator uses **default_settings.txt** configuration file. However, additional configuration files can also be passed.

Configuration files contain key-value pairs. Syntax for most of these pairs is:

```
Namespace.key = value
```

Both *Namespace* and *key* follow the camelCase and are case sensitive. *value* can be numeric, boolean, strings, comma-separated values, value filling and reference to other files. Numeric values can also use the suffixes *k* for Kilo, *M* for Mega and *G* for Giga.

Value Filling

Value Filling is used to dynamically fill the value during run-time. The main purpose is to use the value of a variable during run-time. To use value-filling, put the key name as value surrounded by

```
MovementModel.warmup = %%Reports.warmup%%
```

Run Indexing

What happens if we want to run the simulations for different configurations such as a different routing protocol for each run. One solution would be to create different configurations and write a script passing the desired configuration for each run. However, it is not a maintainable solution if we have a different number of permutations (changing 3 values each for two different parameters). ONE simulator solves this problem by using Run Indexing. Run indexing allows us to change configuration during run-time by providing the same configuration file; containing a set of semi-colon seprated values surrounded by braces ([]) for the deisred key(s).

```
Reports.warmup = [0; 100; 200; 300]
```

In the above example, the first run will use the value 0, next one will use the value 100 and so on. In case the values are less than the total number of simulations, the indexes wrap around (loops back to the start of the array when there is no further item to read).

2.3 Floating Content

Floating content is an ephemeral content sharing service, solely dependent on the mobile devices in the vicinity using principles of opportunistic networking. The net result is a best effort service for floating content in which: 1) information dissemination is geographically limited; 2) the lifetime and spreading of information depend on interested nodes being available; 3) traffic can only be created and caused locally; and 4) content can only be added, but not deleted [Hyy+11].

In Floating Content, a message (a text message, an image, etc.) deemed to be of interest to other people at a certain area is tagged with geographical coordinates of that area. This area is referred to as the anchor-zone of the message [Jörc]. Each message has two main properties namely *Center* and *Availability/anchor zone*. For the sake of simplicity, the anchor zone is defined as a circle with a radius. In short, floating messages can be relayed to other nodes inside the anchor zone. Every time a copy of the message is made, the anchor zone properties are copied to make sure that the copied message has the same anchor zone as the original message.

Whenever a message is generated, it is assigned an anchor zone (identified by a radius and the node's current location). The message is transferred to other nodes in the anchor zone. Whenever a node moves out of the anchor zone, it may delete its copy of the message or keep it until first interaction with another node depending on the configuration.

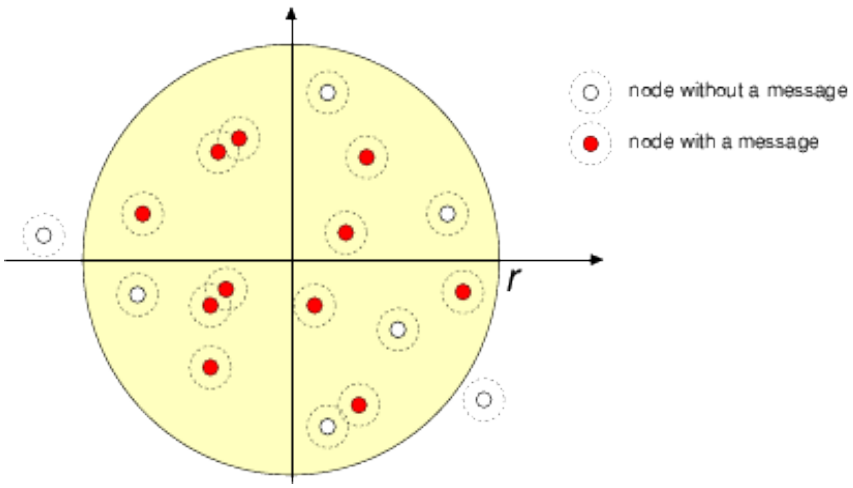

Figure 2.2: Floating-Content and its anchor zone [Hyy+11]

2.3.1 Floating Content and ONE Simulator

Before continuing the discussion on Floating Content and its implementation in the ONE simulator, let us introduce a few concepts.

Table 2.1: Floating Content: Imporant Concepts

Concept	**Detail**
interval	minimum time between generation of two messages by the same node.
radius (r)	the replication range for floating message.
anchor (a)	the availability range of the floating message.
time to live (ttl)	The message life (seconds) for the floating message.
size	The size of the floating message.
replicationPolicy	dictates how floating messages are replicated when the node comes in contact with another node.
deletionPolicy	dictates when floating messages are deleted.

Table 2.2: Floating Content: Replication Policies

Replication Policy	**Detail**
first in first out (fifo)	messages are replicated in the order they are found in the buffer. This is the default replication policy.
random shuffle (rnd)	Shuffle and select messages randomly for replication.
smallest anchor zone first (saf)	messages with the smallest anchor zone are replicated first.
smallest cylinder (radius) first (svf)	messages with the smallest cylinder *(anchor * message size)* are replicated first.
smallest cylinder (area) first (svf2)	messages with the smallest cylinder *(anchor * anchor * message size)* are replicated first.
smallest total (radius) first (stf)	messages with the smallest total *(anchor * ttl * message size)* are replicated first.
smallest total (area) first (stf2)	messages with the smallest total *(anchor * anchor * ttl * message size)* are replicated first.

Table 2.3: Floating Content: Deletion Policies

Deletion Policy	Detail
Encounter	the message is deleted outside the anchor zone on first encounter with another node. The advantage/use case is that the node still cariies the message if it leaves the anchor zone for a short period of time (and does not enouncter another node during that period). It is the default policy in ONE Simulator.
Immediate	the message is deleted as soon as the node leaves the anchor zone.

Components

The ONE Simulator has two main components for dealing with Floating Content which are *FloatingApplication* and *FloatingContentRouter*.

- *FloatingApplication* is responsible for generation of the messages with the floating content parameters (such as anchor, ttl etc.). FloatingApplication can be associated with any Host Group (or all the Host Groups) using the configuration file(s).
- *FloatingContentRouter* is responsible for all the routing related functionalities for the floating messages. It takes care of replication, deletion (and dropping) and prioritization of message. Replication and Deletion is performed as per the Replication and Deletion policies. Different parameters of the FloatingContentRouter can be set in configuration file(s).

Configuration

As mentioned in *Configuration of ONE Simulator*, all the configurations need to be provided as configuration files containing key-value pairs. Below are some examples for setting configuration for the floating content.

```
# Setting the Group (global) router to FloatingContentRouter
Group.router = FloatingContentRouter


# FloatingContent Router with first in first out replication and
    immediate deletion
FloatingContentRouter.seed = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9 ] # run-
    indexed seed for random number generator
FloatingContentRouter.replicationPolicy = fifo
```

```
FloatingContentRouter.deletionPolicy = immediate


#Floating App configuration
floatingApp.seed = [0; 1; 2; 3; 4; 5; 6; 7; 8; 9 ]

floatingApp.type = FloatingApplication
floatingApp.startTime = 0
floatingApp.interval = 1800
floatingApp.ttl = 300
floatingApp.destination = 0
floatingApp.messageSize = 5k
floatingApp.anchor = 200


# Assigning Floating app to all Groups
Group.nrofApplications = 1
Group.application1 = floatingApp


# Assigning Floating app to only 1 Group
Group1.nrofApplications = 1
Group1.application1 = floatingApp
```

3 Design and Implementation

3.1 Maps

3.1.1 Concept

Maps play a very important role in simulating real-world depiction of any place in the ONE simulator. This allows us to simulate the physical location as accurately as possible. The *MapBased Movement* provides us with the different types of map related movements.

The ONE Simulator requires the map in the modified Well-Known Text (WKT) Format. I call it the modified version as the WKT files utilizes points based system instead of coordinates based system. We use the Osm2Wkt tool[May10] for converting Open Street Maps (OSM) files to WKT for the ONE Simulator.

Filtering Maps

The ONE Simulator can accept one or more WKT map files. In case of more than 1 files, all of the files are basically overlayed on the top of each other. One thing to note is that the files need to be fully connected otherwise the ONE simulator won't start and will throw an error.

OpenStreetMaps provides with a tool called OSMFilter [Opec] which enables us to filter the OSM maps. This allows us to filter specific elements such as roads, bus routes etc. To use the filtered maps with ONE simulator, we need to convert each of them to WKT.

The current implementation of the ONE simulator uses the following 4 wkt files:

Table 3.1: ONE Simulator: WKT Map files used by default

Map File	**Details**
main_roads.wkt	represents the main roads (primary, secondary, tertiary).
roads.wkt	represents the normal roads and streets.
shops.wkt	represents the shops.
pedestrian_paths.wkt	represents the pedestrian paths.

OSM Maps

The first step in getting the maps is to download the OSM map from the Open Street Map export tool [Opea]. Below is a depiction of the OSM Map for Central Munich.

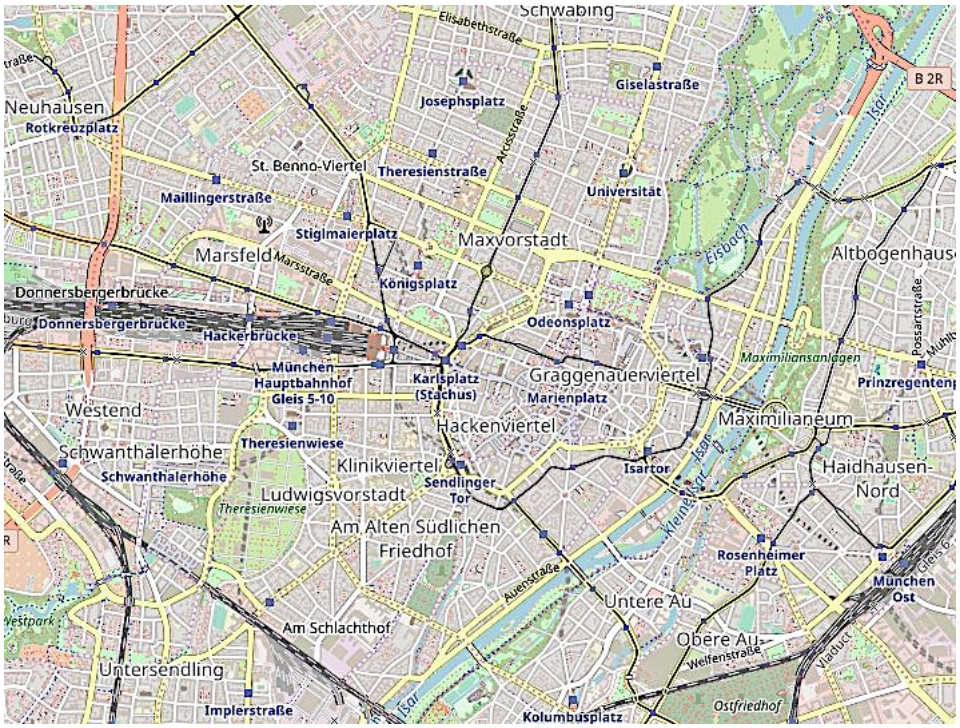

Figure 3.1: Central Munich OSM Map - from OpenStreetMaps [Opeb]

Conversion to WKT

There are a number of tools that canconvert an OSM to WKT, however, there are a few reasons we do not use any of those tools:

- These tools do not make provide us with a fully connected maps. They just perform the conversion.
- These tools do not convert the coordinates to the points-based system. Even though the resulting WKT file would work with the ONE simulator, however, the distances would be very small (as compared to the original). Let me give you an example, The distance between *1.0001,1.2* and *1.0002,1.2* in points is *11m* while if

we provide this directly to the ONE simulator the distance would be *0.001 m (1 mm)*.

```
POINT (11.5859318 48.1383583)
POINT (11.5763905 48.1367741)
LINESTRING (11.5669265 48.1385126, 11.5670542 48.1384848, 11.5671648 48.1384666,
11.5671242 48.1383751, 11.5670732 48.1382434, 11.5668682 48.1382749, 11.5669134
48.1384592, 11.5669265 48.1385126)
LINESTRING (11.5692615 48.1417543, 11.569205 48.1418468, 11.5691685 48.1419658,
11.5692268 48.142013, 11.5692535 48.1420517, 11.5693459 48.1422348, 11.5693431
48.142284, 11.5693435 48.1423297)
LINESTRING (11.568828 48.1419005, 11.5691311 48.1419929, 11.5692268 48.142013,
11.5692977 48.1420006, 11.5696013 48.1419195, 11.5696696 48.1418847, 11.5697647
48.1418473)
LINESTRING (11.5721983 48.1367036, 11.5723376 48.1366081, 11.5724347 48.136661,
11.5725711 48.1367384, 11.5726917 48.1368081, 11.5725562 48.1369124, 11.5725 48.1368797,
11.5721983 48.1367036)
LINESTRING (11.5637581 48.1384165, 11.5639442 48.1384023, 11.5647922 48.1383376,
11.5649032 48.1383314)
LINESTRING(11.5643483 48.1366282, 11.5641708 48.1366269, 11.5641363 48.1366285,
11.5641466 48.1366981, 11.5641555 48.1367525, 11.5640278 48.1367587, 11.5639643
48.1367618, 11.5637824 48.1367706, 11.5637833 48.136853, 11.5639831 48.1368482,
11.5641354 48.1368384, 11.5643864 48.1368167, 11.5643483 48.1366282)
POLYGON ((11.5639106 48.133894, 11.5637535 48.1338771, 11.5637492 48.1339479, 11.5637481
48.1340577, 11.5637652 48.13413, 11.5638975 48.1341169, 11.5640656 48.1340961,
11.5640161 48.133999, 11.5639036 48.1340153, 11.5639106 48.133894))
```

Figure 3.2: Snapshot of WKT File Contents - Converted using OpenJump [Oped]

We have the OSM2WKT tool [May10] as it has solution for both of the above issues and can generate a full-connected point-based WKT file that can be used with the ONE Simulator.

```
LINESTRING (1248.752 800.047, 1271.99 845.571)
LINESTRING (1103.019 812.246, 1097.124 790.941, 1087.514 793.921, 1084.103 782.868,
1091.984 780.433, 1088.632 769.58, 1099.912 766.089, 1103.456 777.542, 1106.432 776.619,
1107.203 779.121, 1113.118 777.297, 1121.215 806.263, 1103.019 812.246)
LINESTRING (1275.053 852.376, 1281.869 868.343)
LINESTRING (1022.878 526.619, 1014.281 548.336, 1015.027 565.193)
LINESTRING (1021.207 492.76, 1021.517 501.156)
LINESTRING (1477.012 257.739, 1467.313 258.006, 1453.699 239.948, 1459.437 233.565)
LINESTRING (1459.437 233.565, 1465.346 229.34)
LINESTRING (1271.99 845.571, 1275.053 852.376)
LINESTRING (1010.935 458.913, 1013.933 457.967, 1017.567 469.487, 1018.005 470.888,
1015.007 471.833, 1010.935 458.913)
LINESTRING (1026.5 478.227, 1029.343 477.36, 1026.947 469.543, 1025.108 463.538,
1022.265 464.406, 1024.498 471.689, 1026.5 478.227)
LINESTRING (1521.922 462.526, 1531.1 468.965, 1540.879 475.836, 1541.725 476.448,
1542.252 476.815, 1541.822 473.957, 1540.724 474.113, 1531.805 467.886, 1523.243
461.893, 1523.073 460.97, 1520.52 461.503, 1520.98 461.837, 1521.922 462.526)
LINESTRING (431.842 301.683, 431.085 303.573, 432.918 304.296, 433.667 302.406, 431.842
301.683)
LINESTRING (363.027 280.456, 362.426 282.391, 364.303 282.98, 364.912 281.045, 363.027
280.456)
LINESTRING (426.662 293.021, 433.616 295.79, 434.477 293.555, 435.308 291.531, 428.347
288.762, 426.662 293.021)
```

Figure 3.3: Snapshot of WKT File Contents - Converted using OSM2WKT tool [May10]

Below is graphics depiction of the Central Munich WKT file.

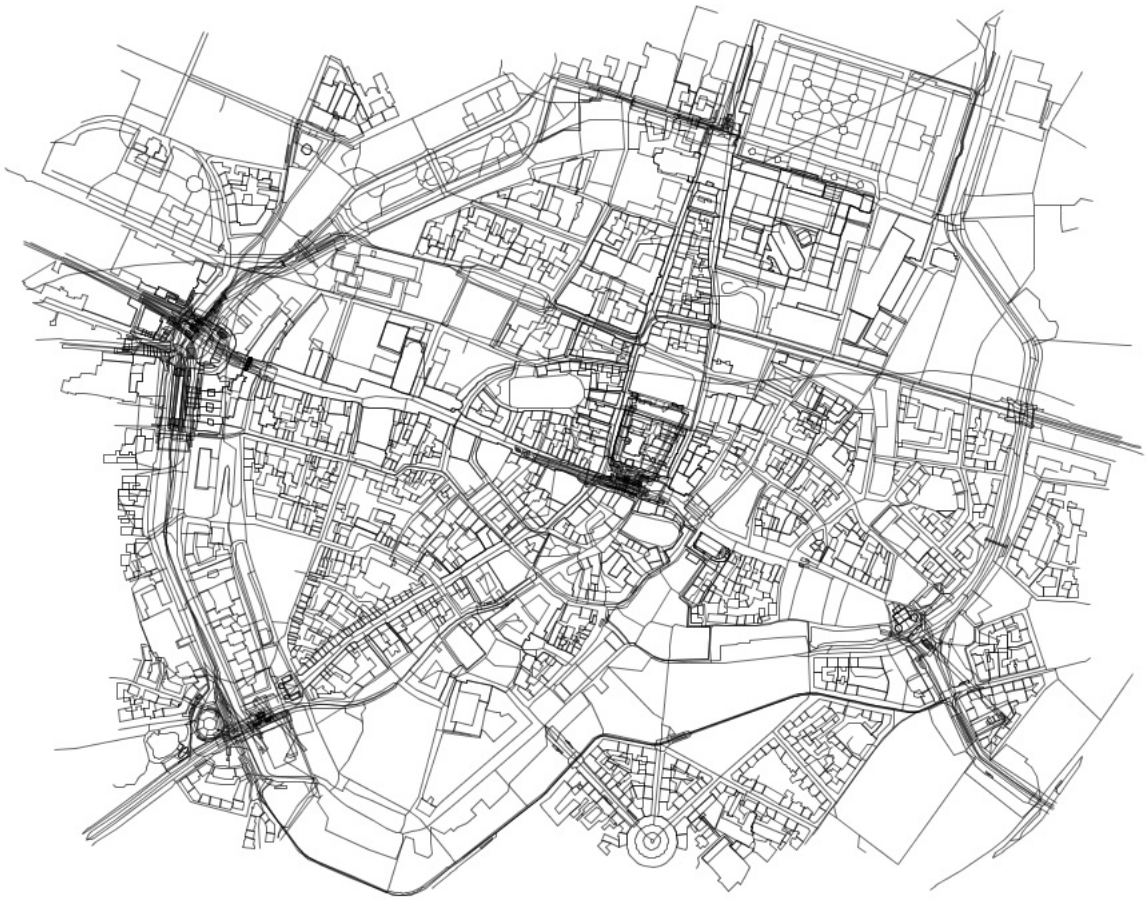

Figure 3.4: Central Munich WKT Map - Created using OSM2WKT tool [May10] and Screenshot using OpenJump [Oped]

3.1.2 Implementation

OSM to WKT

Using OSM2WKT tool [May10], any OSM file can be converted into a fully connected WKT file for use in the ONE simulator. Below are the details of how to use the OSM2WKT tool [May10] to generate a WKT file:

```
java -jar ./osm2wkt.jar mapfile.osm

options:
       -o outputfile - specifiying the output file. In case no outputfile
            is mentioned, .wkt is appended to the name of input file.
       -a - open the output file in append mode (adding the wkt output at
            the end of the output file)
       -t deltaX deltaY - translate map by deltaX and deltaY meters
```

Listing 3.1: Using OSM2WKT tool [May10] to generate WKT

The OSM2WKT tool [May10] performs the following operations on the OSM file:

1. Reads the OSM Map.
2. The coordinates are converted to the points based system:
   a) The bounding box of the map is converted into point based system (width and height in meters).
   b) All the coordinates are converted to the point based system.
3. The completeness of the graph is then checked and all the weird and unconnected edges are deleted so that we have a complete graph.
4. The file is then saved as a WKT file.

Filtering OSM

In order to filter OSM files, OSM Filter tool [Opec] is needed. Here is the basic syntax to use the OSM Filter tool [Opec].

```
osmfilter inputfile.osm --keep="{condition}" > outputfile.osm
```

Listing 3.2: How to use the OSM Filter tool [Opec]

The listings below show how to create the four different types of OSM files which can then be converted to WKT files.

```
osmfilter mapfile.osm --keep="highway=*ary trunk" > main_roads.osm
```

Listing 3.3: Generating main_roads.osm file

```
osmfilter mapfile.osm --keep="highway=residential service *link
    living_street"
> roads.osm
```

Listing 3.4: Generating roads.osm file

```
osmfilter map.osm --keep="highway=pedestrian footway living_street"
> pedestrian_paths.osm
```

Listing 3.5: Generating pedestrian_paths.osm file

```
osmfilter map.osm --keep="shop=*" > shops.osm
```

Listing 3.6: Generating shops.wkt file

The main issue with using separate osm files is that we cannot use Osm2WKT tool [May10] and it is a difficult task to make sure the resultant map is fully connected. I have modified the Osm2WKT tool [May10] to accept more than 1 OSM files and generate the corresponding WKT files that are fully connected, however, that is beyond the scope of this document. In our simulations, we are using a single map file.

3.1.3 Configuration

In order to utlize the MapBasedMovement and the map files we have prepared (using method in the above sections), we would need to add the following to one of the configuration files:

```
Group.movementModel = MapBasedMovement

# Configuration for 1 map file
MapBasedMovement.nrofMapFiles = 1
MapBasedMovement.mapFile1 = data/maps/map.wkt


# Configuration for multiple maps files (these files need to be connected
    (when overlayed on each other))
MapBasedMovement.nrofMapFiles = 4

MapBasedMovement.mapFile1 = data/maps/roads.wkt
MapBasedMovement.mapFile2 = data/maps/main_roads.wkt
MapBasedMovement.mapFile3 = data/maps/pedestrian_paths.wkt
MapBasedMovement.mapFile4 = data/maps/shops.wkt
```

Listing 3.7: Map-based Movement Configuration

3.2 Access Points

3.2.1 Concept

Access Points are devices that connect devices to one another and facilities the flow of information among devices. An Access Point can operate in any of the following three modes:

Table 3.2: ONE Simulator: Modes of Operation for Access Points/Wifi Interface

Mode	**Description**
Access Point (AP)	Access Point (AP) provides a bridge for two Station Adapters (SAs) to connect and communicate with each other. Two Access Points cannot talk to each other directly.
Station Adapter (SA)	Station Adapter (SA) is a passive device that can connect to an Access Point and communicate to other Station Adapters using the Access Point (AP). Two Station Adapters cannot talk to each other directly.
Adhoc	Adhoc Mode is used when we want two devices to connect and communicate direclty to each other. Hosts in Adhoc mode cannot connect to either an Access Point (AP) or Station Adapter(SA) and vice versa. This is the default mode.

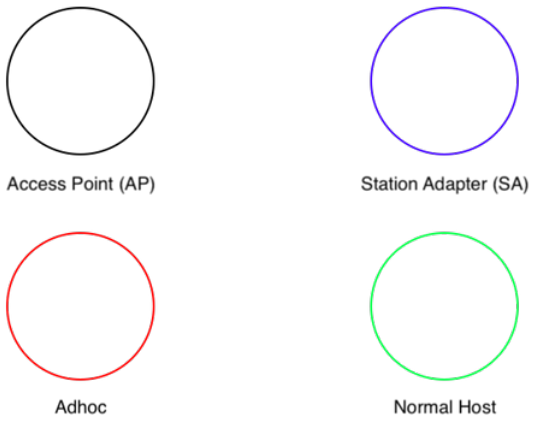

Figure 3.5: Hosts with Access Points (SA, AP and Adhoc) and Normal Host

The above coloring combination is used in the ONE Simulator to give user the ability to visually different between the hosts with different modes (or no mode at all). Below is a screenshot from the ONE Simulator with all of the above Hosts in action.

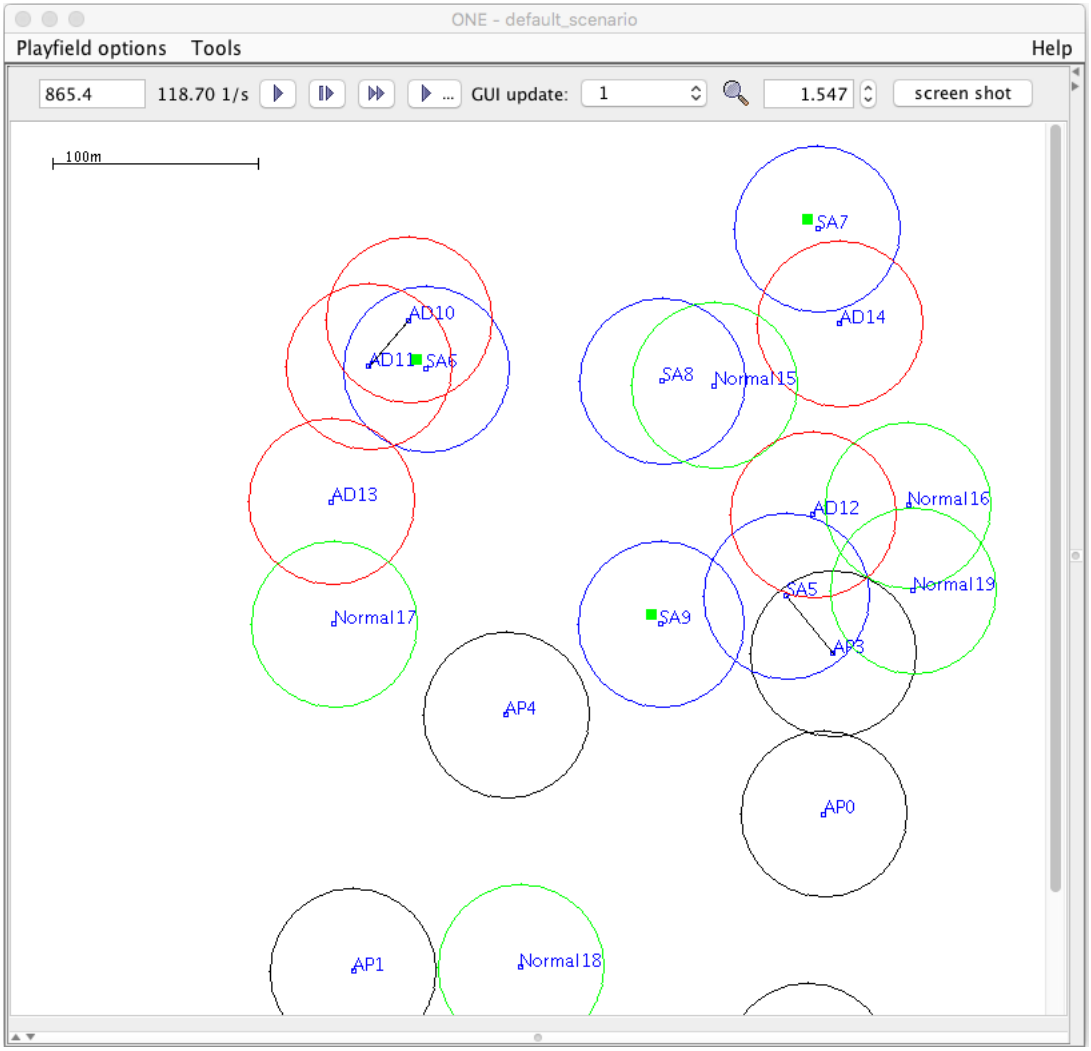

Figure 3.6: ONE Simulator showing all the Hosts (AP enabled and normal) in action

3.2.2 Implementation

Before discussing implementation details, I would like to discuss the pre-requisite for the implementation. The following table lists two classes and what they are used for:

Class	Description
DTNHost	This class represents a DTN Capable host. It contains all the properties and functionalities fo the host such as groupID, MessageListeners, MovementListeners etc.
DTNHostWithWifi	This class is a subclass of the *DTNHost* class and represents a host with Access Point/Wifi capability. It inherits everything from the parent class and adds an extra mode (which can either be ADHOC, AP or SA).

Table 3.3: ONE Simulator: Classes related to Hosts/Nodes

DNTHostWithWifi is a very lean host that takes everything from its parent *DTNHost* and adds the mode. The following listing shows the implementation of *DTNHostWithWifi*.

```
public class DTNHostWithWifi extends DTNHost {
        public static int AP = 1;
        public static int SA = 2;
        public static int ADHOC = 3;
        private int mode;

        public DTNHostWithWifi(List<MessageListener> msgLs, List<
            MovementListener> movLs, String groupId,List<NetworkInterface>
            interf, ModuleCommunicationBus comBus, MovementModel mmProto,
            MessageRouter mRouterProto, String mode) {
                super(msgLs, movLs, groupId, interf, comBus, mmProto,
                    mRouterProto);
                this.mode = mode.equalsIgnoreCase("ap") ? AP : mode.
                    equalsIgnoreCase("sa") ? SA : ADHOC;
        }
        public int getMode() {
                return mode;
        }
        public String getModeName() {
                return mode == ADHOC ? "ADHOC" : (mode == AP ? "AP" : "AP")
                    ;
        }
```

```
}
```

Listing 3.8: ONE Simulator: DTNHostWithWifi Implementation

There is still another component that needs to be implemented i.e. an interface to the new *DTNHostWithWifi*. We call it *WifiInterface*. The following table discusses the two main classes.

Table 3.4: ONE Simulator: WifiInterface & its parent interface

Class	**Description**
SimpleBroadcastInterface	*SimpleBroadcastInterface* is a Network Interface that is responsible for connectivity among hosts. It allows one-to-one transmissions with a constant bit-rate.
WifiInterface	*WifiInterface* is responsible for connectivity between two different *DTNHostWithWifi* objects. It is also responsible that the connection rules (SA can connect only to AP, Adhoc hosts can connect to each other only) are upheld.

WifiInterface takes almost everything from its parent but adds the extra ability to check if two *DTNHostWithWifi* can connect to each other. Another method that it overrides is the ability to return *DTNHostWithWifi* with the *getHost()* method.

```
public class WifiInterface extends SimpleBroadcastInterface {
        ...
        public void connect(NetworkInterface anotherInterface) {
                if (canConnectTo(anotherInterface)) {
                        ...
                }
        }

        /*
         * Along with the usual connectivity rules, we also check if the
         * DTNHosts can connect to each other by having compatible modes.
         */
        boolean canConnectTo(NetworkInterface anotherInterface) {
                boolean canConnect = isScanning()
                && anotherInterface.getHost().isRadioActive()
                && isWithinRange(anotherInterface)
```

```
                && !isConnected(anotherInterface)
                && (this != anotherInterface);

                if(canConnect) {
                        WifiInterface thisInterface = (WifiInterface) this;
                        WifiInterface secondInterface = (WifiInterface)
                            anotherInterface;

                        int thisInterfaceMode = thisInterface.getHost().
                            getMode();
                        int secondInterfaceMode = secondInterface.getHost().
                            getMode();

                        if(thisInterfaceMode == DTNHostWithWifi.ADHOC) {
                                canConnect = thisInterfaceMode ==
                                    secondInterfaceMode;
                        } else {
                                canConnect = (thisInterfaceMode ==
                                    DTNHostWithWifi.AP && secondInterfaceMode
                                     == DTNHostWithWifi.SA)
                                || (thisInterfaceMode == DTNHostWithWifi.SA
                                    && secondInterfaceMode == DTNHostWithWifi.
                                    AP);
                        }
                }

                return canConnect;
        }

        public DTNHostWithWifi getHost() {
                return (DTNHostWithWifi)host;
        }
}
```

Listing 3.9: ONE Simulator: WifiInterface Implementation

Now, we need to combine both of these to make sure that any host having a *WifiInterface* is instantiated as *DTNHostWithWifi* objects and all othe other hosts are instantiated as *DTNHost* objects. This needs to be done inside *createHosts()* function of *SimScenario* class. Below are the changes that are made to achieve this purpose: vspace3mm

```
boolean hasWifiInterface = false;

// setup interfaces
for (int j=1;j<=nrofInterfaces;j++) {
        ...
        if(iface instanceof WifiInterface) {
                hasWifiInterface = true;
        }
        ...
}
...
//Get Wifi modes for hosts (only exist for DTNHostWithWifi)
List<String> modes = new ArrayList<String>();
String wifiMode = s.getSetting("mode",null);

if(wifiMode == null) {
        String modesFile = s.getSetting("modesFile",null);
        if(modesFile != null) {
                try {
                        Scanner in = new Scanner(new FileReader(modesFile));
                        while(in.hasNextLine()) {
                                modes.add(in.nextLine());
                        }
                        in.close();
                } catch (FileNotFoundException e) {}
        }
}

// creates hosts of ith group
for (int j=0; j<nrofHosts; j++) {
        ...
        if(!hasWifiInterface) {
                DTNHost host = new DTNHost(this.messageListeners,
                        this.movementListeners, gid, interfaces, comBus,
```

```
                    mmProto, mRouterProto);
            hosts.add(host);
        } else {
            String mode = wifiMode != null ? wifiMode : (j < modes.size
                () ? modes.get(j) : "");

            DTNHostWithWifi host = new DTNHostWithWifi(this.
                messageListeners, this.movementListeners, gid,
                interfaces, comBus, mmProto, mRouterProto, mode: mode);
            hosts.add(host);
        }
}
```

Listing 3.10: ONE Simulator: Changes to SimScenario for Implementation of the AP-capable hosts

3.2.3 Configuration

Here we will show how to configure *DTNHost* and *DTNHostWithWifi*. We will also show how to configure *mode* for *DTNHostWithWifi*.

```
# We need to create two different types of interfaces (based on the types
    of interface, either DTNHost or DTNHostWithWifi object is created)

# Declaring WifiInterface (any host/group of hosts having this interface
   will be the DTNHostWithWifi hosts)
wifiInterface.type = WifiInterface
wifiInterface.transmitSpeed = 50k
wifiInterface.transmitRange = 40

# Declaring SimpleBroadcastInterface (any host/group of hosts having this
    interface will be the DTNHost hosts)
btInterface.type = SimpleBroadcastInterface
btInterface.transmitSpeed = 250k
btInterface.transmitRange = 10

# Making sure all the groups are counted and created
Scenario.nrofHostGroups = 5
```

```
# Defining general behavior for Groups
Group.nrofInterfaces = 1
Group.interface1 = btInterface

# Defining normal DTNHosts (We do not need to attach btInterface as all
    the groups would have the btInterface by default)
Group1.groupID = Normal

# Defining Station Adapters (groupID can be anything as only the mode can
     identify between different types of hosts)
Group2.groupID = SA
Group2.interface1 = wifiInterface
Group2.mode = sa

# Defining Access Points
Group3.groupID = AP
Group3.interface1 = wifiInterface
Group3.mode = ap

# Defining Adhoc nodes (We can eliminate mode here as well as adhoc is
    the default mode)
Group4.groupID = ADHOC
Group4.interface1 = wifiInterface
Group4.mode = adhoc

# Declearing Wifi nodes with different modes (modes are contained in the
    modesFile)
Group5.groupID = D
Group5.interface1 = wifiInterface
Group5.modesFile = data/modes.txt #modes.txt is containing one mode per
    line. The mode is applied to the hosts in the same order
```

3.3 Snapping to Access Points

3.3.1 Concept

In the *Floating Content Section*, it was discussed that a message is tagged with geographical coordinates and an availability/radius. *Snapping to Access Point* is the concept of increasing the availability of the message beyond its original availability zone. In the configuration, we can define how many Access Points should the message snap to at maximum. Theoretically, the number of snapped Access Points should directly influence the TTL (Time to live) of the message. Below is a step-wise explanation of the process:

1. The host Station Adapter (*SA1*) generates a message *M* with a center *C* (which is the location of *SA1* at that time), availability *a* and number of maximum access points to snap to (*k*). Assuming $k = 3$.
2. *SA1* comes in contact with an Access Point (*AP1*), connection is made and a copy of the message *M* is transferred (*M'*).
3. If *k*-value for the original message *M* is greater than 0:
   a) *k*-value for the original message (*M*) is decreased by 1.
   b) *k*-value for the copied message (*M'*) is set to zero.
   c) The center of the copied message (*M'*) is changed to the center of the Access Point (*AP1*).
4. The above process continues for the original message *M* until k=0.

Since the ID of the message *M* and all of the copies is the same, they are treated as the same message; which allows us to extend the availability of the message.
There are a few important points to note:

1. Copies of the message are not snapped to any access point, Only the original message is snapped.
2. If a message is snapped to 1 access point which drops it due to any reason (apart from TTL expiry), whenever that message (the original or even the copy which knows about this access point) is copied again, it is snapped to that access point again.
3. The availability zones of the copies might or might not intersect each other, however, the availability zone of the original message and each copy would have an overlap.

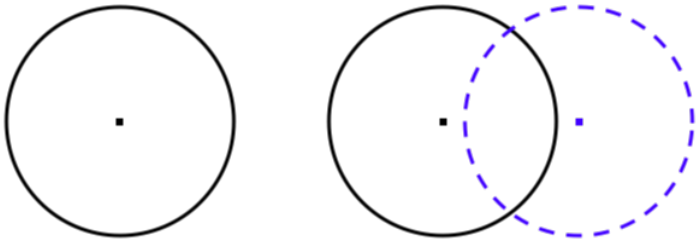

Figure 3.7: Availability zones of Message with k=0 (not snapped yet) to the left and k=1 (snapped to 1 Access Point) to the right

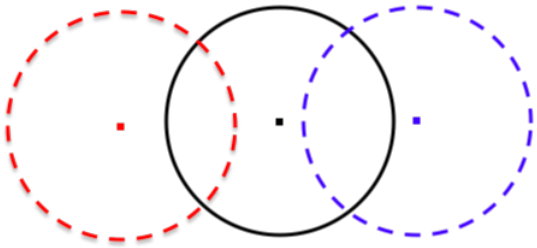

Figure 3.8: Availability zones of Message with k=2 (snapped to 2 Access Points)

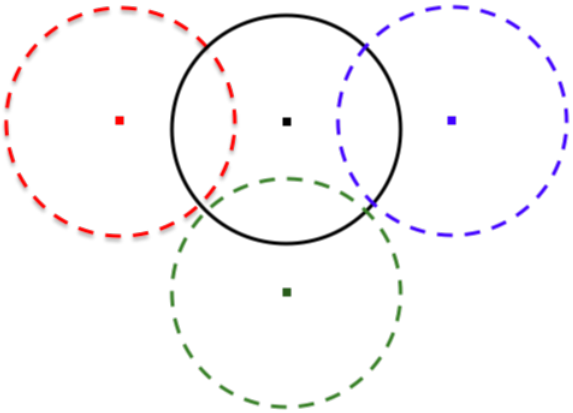

Figure 3.9: Availability zones of Message with k=3 (snapped to 3 Access Points)

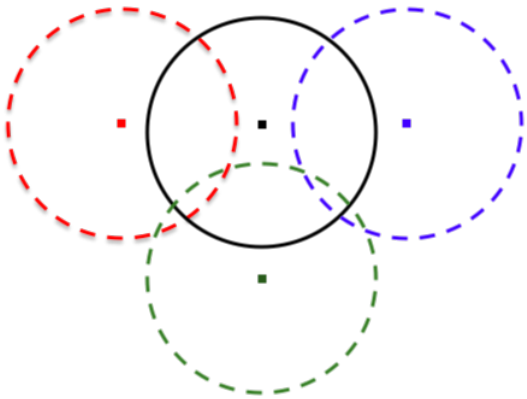

Figure 3.10: Special Case: Message with k=3 where there is no overlap between availability zones of the copies

3.3.2 Implementation

Snapping to Access Point

1. The very first point is to attach an *originalAnchor* to the message (which is the point where the message was originally generated).

   ```
   m.addProperty("originalAnchor", m.getProperty("anchor"));
   ```

2. The settings for the number of Access Points to snap to (*kNextAPs*) is fetched inside *FloatingContentRouter*.

   ```
   if (fcSettings.contains(FC_NUM_OF_APS))
           kNextAPs = fcSettings.getInt(FC_NUM_OF_APS);
   ```

3. When message transfer is complete (*messageTransferred* method in *FloatingContentRouter*), we need to check f the message need to be snapped to any Access Point (by checking *kNextAPs* property of the message).:

   a) If the host does not have a wifi interface, the control is returned back to the caller.

   ```
   if(!(this.getHost() instanceof DTNHostWithWifi)
           return m;
   ```

   b) If the receiving node is not an access point, the control is returned back to the caller.

```
DTNHostWithWifi receivingNode = (DTNHostWithWifi) this.getHost();

if(receivingNode.getMode() != DTNHostWithWifi.AP)
        return m;
```

c) *kNextAP* for the copied message is set to 0.

```
m.updateProperty(FC_NUM_OF_APS, 0);
```

d) The *anchor* of the copy is set to location of the access point if the message had been already anchored to this access point and the message still can be anchored to access points (*kNextAPs>0*). Decrease *kNextAPs* only if the message was not previously anchored to this access point.

```
if ((int) msg.getProperty(FC_NUM_OF_APS) > 0 || (
    anchorAccessPoints != null && anchorAccessPoints.contains(
    receivingNode.toString()))) {
        m.updateProperty(FC_ANCHOR, receivingNode.getLocation().
            clone());

        if((int) msg.getProperty(FC_NUM_OF_APS) > 0)
                msg.updateProperty(FC_NUM_OF_APS, (int) msg.
                    getProperty(FC_NUM_OF_APS) - 1);
}
```

e) The access point ID is added to the list of APS (*anchorAPs*) property of the message. This helps us in re-snapping the message to the access point if the access points get a new copy of the same message.

```
anchorAccessPoints.add(receivingNode.toString());
msg.updateProperty(FC_ANCHOR_APS, anchorAccessPoints);
m.updateProperty(FC_ANCHOR_APS, anchorAccessPoints);
```

f) The access point location is added to the list of APS (*anchorAPsLocation*) property of the message. This is used to make sure that the new availability zones are also accounted for before trying to delete a message.

```
anchorAccessPointsLocations.add(receivingNode.getLocation().
    clone());
msg.updateProperty(FC_ANCHOR_APS_LOC,
    anchorAccessPointsLocations);
m.updateProperty(FC_ANCHOR_APS_LOC, anchorAccessPointsLocations);
```

Changes to Message Deletion Algorithm

Here we are making sure that the message is deleted only after the it is outside of the new availability zones (in case a message is snapped to access point, we need to check that the message is outside all the availability zones for that message).

```
for (Message m : m_set) {
        ArrayList<Coord> locations;
        try {
                locations = new ArrayList<Coord>((HashSet<Coord>)m.
                    getProperty(FloatingContentRouter.FC_ANCHOR_APS_LOC));
        } catch(NullPointerException ignored) {
                locations = new ArrayList<>();
        }

        locations.add(0, (Coord)m.getProperty(FC_ANCHOR));
        locations.add(1,(Coord)m.getProperty(FC_ANCHOR_ORIGINAL));

        int counter = 0;
        while(counter<locations.size()) {
                distance_curr = loc.distance(locations.get(counter));

                if ((deletion_check(distance_curr, (Double) m.getProperty(
                    FC_R), (Double) m.getProperty(FC_A)) != 1))
                            break;
                counter ++;
        }

        if(counter == locations.size() && !d_list.contains(m.getId()))
                d_list.add(m.getId());
}
```

3.3.3 Configuration

```
FloatingContentRouter.kNextAPs = 1
```

3.4 Access Points Data

3.4.1 Concept

Data Collection

We are using the wifi access points database from RadioCells [Rad]. The database has a number of tables, however, we are only concerned with *wifi_zone*. We are only using *latitude* and *longitude* data.

Data Processing

We follow the following steps to process the data for the ONE Simulator:

1. The data from sqlite database is exported to a text file.
2. The data is processed by removing the double quotes and separating latitude and longitude by a comma (,).
3. To make sure that the coordinates are converted to the correct scaling, we add the boundary coordinates to the file. Below are the boundary coordinates:
   - minimumLatitude,0
   - maximumLatitude,0
   - 0,minimumLongitude
   - 0,maximumLongitude
4. The processed file is then passed to a Coordinate Converter which returns a WKT file with the transformed coordinates.
5. The WKT file can then be used for setting up the access points on the map using *StationaryMapBasedMovement*.

```
POINT (1062.64139413 199.02659474)
POINT (376.98851001 206.11914023)
POINT (992.97309405 214.68331232)
POINT (520.67853701 221.98222492)
POINT (520.9567434 228.63351036)
POINT (557.31248005 238.69873078)
```

Figure 3.11: Transformed Access Points Data

3.4.2 Implementation

A majority of the implementation is taken from Osm2Wkt [May10]. The implementation is pretty straight-forward and can be explained in the following steps:

1. We open the file and read the content into an array of Landmarks (*Landmark* has *latitude*, *longitude* for the original coordinates as well as x, y for storing the transformed coordinates)

```
Scanner in = new Scanner(new FileReader(filePath));

while(in.hasNextLine()) {
        String text = in.nextLine().trim();
        if(text.length() > 0) {
                String[] coordinates = text.split(",");
                Landmark landmark = new Landmark(Double.parseDouble(
                    coordinates[0]), Double.parseDouble(coordinates[1])
                    );
                landmarks.add(landmark);
        }
}
in.close();
```

2. The coordinates are then transformed using *transformCoordinates* [May10].

3. Finally the coordinates are written in a WKT file which can be used by the ONE Simulator.

To Read the WKT file in ONE simulator, we have implemented *StationaryMapBasedMovement* and *MapNodeExtended* classes. The following table explains the functionality of each of these classes.

Class	**Details**
MapNodeExtended	Inherits from the *MapNode* class. It reads the WKT file and apply transformations (mirror and translation) to the location points.
StationaryMapBasedMovement	Inherits from *MapBasedMovement* class. Get list of transformed locations (from the WKT file) by passing it to *MapNodeExtended*.

Table 3.5: ONE Simulator: Classes for WKT based Node positioning

Below are two main functions from *StationaryMapBasedMovement*.

```
public StationaryMapBasedMovement(Settings settings) {
        super(settings);

        String fileName = settings.getSetting(ROUTE_FILE_S);
        allNodes = MapNodeExtended.readNodes(fileName, getMap());
}

@Override
public Coord getInitialLocation() {
        counter = (counter + 1) \% allNodes.size();
        return allNodes.get(counter).getLocation();
}
```

Listing 3.11: Main features of *StationaryMapBasedMovement* class

Below is the main code snippet from *MapNodeExtended* class.

```
WKTReader reader = new WKTReader();
List<Coord> coords;
File routeFile = null;

double xOffset = map.getOffset().getX();
double yOffset = map.getOffset().getY();

List<MapNode> nodes = new ArrayList<MapNode>();

try {
        routeFile = new File(fileName);
        coords = reader.readPoints(routeFile);
}
catch (IOException ioe) {
        throw new SettingsError("Couldn't read MapRoute-data file " +
                        fileName +  " (cause: " + ioe.getMessage() + ")");
}
```

```
for (Coord c : coords) {
        if(map.isMirrored())
                c.mirror();

        c.translate(xOffset, yOffset);

        MapNode node = map.getNodeByCoord(c);
        if (node == null) {
                node = new MapNode(c);
        }
        nodes.add(new MapNodeExtended(node));
}

return nodes;
```

Listing 3.12: Getting List of all MapNodes (taken from *MapNodeExtended*)

3.4.3 Configuration

```
Group1.routeFile = data/path_to_wkt_file/file_name.wkt
```

4 Simulations

4.1 Scenarios

4.2 Results

4.2.1 Inference from Results

5 Conclusions and Discussions

6 Appendix

List of Figures

List of Tables

Bibliography

[Ari] Ari Keränen. *The Opportunistic Network Environment simulator README*. URL: https://github.com/akeranen/the-one/wiki/README.

[CC06] D. J. Corbett and D. Cutting. "AD LOC : Location-based Infrastructure-free Annotation." In: 2006.

[CSK09] A. A. V. Castro, G. D. M. Serugendo, and D. Konstantas. "Hovering Information – Self-Organizing Information that Finds its Own Storage." In: *Autonomic Communication*. Ed. by A. V. Vasilakos, M. Parashar, S. Karnouskos, and W. Pedrycz. Boston, MA: Springer US, 2009, pp. 111–145. ISBN: 978-0-387-09753-4. DOI: 10.1007/978-0-387-09753-4_5.

[Hyy+11] E. Hyytia, J. Virtamo, P. Lassila, J. Kangasharju, and J. Ott. "When does content float? Characterizing availability of anchored information in opportunistic content sharing." In: (May 2011), pp. 3137–3145.

[Jöra] Jörg Ott, Ari Keränen, Teemu Kärkkäinen, Mikko Pitkänen, Frans Ekman and Jouni Karvo. *The Opportunistic Network Environment simulator*. URL: https://github.com/akeranen/the-one/.

[Jörb] Jörg Ott, Cheng Luo, Ari Keränen. *Security Infrastructure for DTN*. URL: https://www.netlab.tkk.fi/tutkimus/sindtn/.

[Jörc] Jörg Ott, J. Kangasharju, J. Virtamo, Aalto University, P. Lassila, E. Hyytiä, M. S. Desta. *Floating Content*. URL: http://www.floating-content.net/.

[Jör17] Jörg Ott, Ljubica Kärkkäinen, Ermias Andargie Walelgne, Ari Keränen, Esa Hyytiä, Jussi Kangasharju. "On the sensitivity of geo-based content sharing to location errors." In: (Feb. 2017).

[KOK09] A. Keränen, J. Ott, and T. Kärkkäinen. "The ONE Simulator for DTN Protocol Evaluation." In: *SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. Rome, Italy: ICST, 2009. ISBN: 978-963-9799-45-5.

[May10] C. P. Mayer. *osm2wkt - OpenStreetMap to WKT Conversion*. http://www.chrismc.de/osm2wkt. 2010.

[Opea] Open Street Maps. *Open Street Maps*. URL: https://www.openstreetmap.org/export.

[Opeb] Open Street Maps. *Open Street Maps*. URL: https://www.openstreetmap.org/.

[Opec] Open Street Maps. *OSM Filter*. URL: https://wiki.openstreetmap.org/wiki/Osmfilter.

[Oped] OpenJump. *OpenJump*. URL: https://jsom.com.

[Rad] RadioCells.Org. *Wifi and Cell Data - Germany*. URL: https://cdn.radiocells.org/de.sqlite.

[TCK10] N. Thompson, R. Crepaldi, and R. Kravets. "Locus: A Location-based Data Overlay for Disruption-tolerant Networks." In: *Proceedings of the 5th ACM Workshop on Challenged Networks*. CHANTS '10. Chicago, Illinois, USA: ACM, 2010, pp. 47–54. ISBN: 978-1-4503-0139-8. DOI: 10.1145/1859934.1859945.