HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Kiran Narayanaswamy

Vehicular Micro Cloud Prototype Implementation

Masterarbeit im Fach Informatik

10. März 2019

Distributed Embedded Systems (CCS Labs)
Heinz Nixdorf Institute, Paderborn University, Germany

Fürstenallee 11 · 33102 Paderborn · Germany

http://www.ccs-labs.org/

Vehicular Micro Cloud Prototype Implementation

Masterarbeit im Fach Informatik

vorgelegt von

Kiran Narayanaswamy

geb. am 17. March 1990
in Bengaluru

angefertigt in der Fachgruppe

Distributed Embedded Systems
(CCS Labs)

Heinz Nixdorf Institut
Universität Paderborn

Betreuer:	**Gurjashan Pannu**
Gutachter:	**Prof. Dr.-Ing. habil. Falko Dressler**
	Prof. Dr. Christian Scheideler
Abgabe der Arbeit:	**10. März 2019**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Kiran Narayanaswamy)
Paderborn, 10. März 2019

Abstract

One of the important paradigms of vehicular networking applications is information based services where the vehicles with communication capabilities interact with the internet cloud servers. This is facilitated by cellular networks for uploading and downloading the relevant data. In future, the number of such vehicles is expected to increase. This is going to result in an increase in the usage of cellular bandwidth (which is limitedly available). This may cause vehicles to suffer a higher end to end communication latency. The solution to this problem is addressed by a newly proposed architecture called Vehicular Micro Cloud architecture. In this architecture, the connected vehicles are bundled together forming small clusters called Vehicular Micro Clouds. The pool of computing, storage, sensing and communication resources contributed by vehicles themselves can be used to aggregate and process the upstream data and cache the downstream data. This minimizes the bandwidth consumption in vehicular networks. In this thesis, the first prototype of the Vehicular Micro Cloud architecture is implemented along with two Micro Cloud services - Data Collection and Aggregation, and Task Distribution. The prototype implemented offers the scope for real-world experimentation and serves as proof of concept. This has been evaluated using the Manhattan grid scenario at different traffic densities and data size transfers.

Contents

Chapter 1

Introduction

In the near future, the number of vehicles on road with communication capabilities such as the WLAN based IEEE 802.11p [1] and LTE/LTE-D2D [2] will increase. Vehicles equipped with such capabilities can interact with other vehicles and with the data centres in the backend using LTE. This vehicular networking aspect has created an opportunity for the automotive developers to develop a wide range of vehicular applications [3] related to road safety, traffic information systems, infotainment, platooning, etc.

One such paradigm of applications is information-based services [4], where the connected vehicles [5], [6] interact with internet cloud servers facilitated by cellular networks, for uploading and downloading relevant data, e.g., in up-to-date HD live maps, the data related to live traffic in an area is transferred to cloud servers, where it is aggregated and processed to create a live map. This live map is sent back to the requesting vehicles. In such a scenario, more vehicles in the future would require more cellular bandwidth to upload and download the data, which is available in limited capacity. This may cause vehicles to suffer a higher end-to-end communication latency.

A similar problem in cellular mobile networks has been handled by introducing a network architecture concept called Mobile Edge Computing (MEC) [7]. The architecture suggests deploying the mobile edge servers at the edge of the networks. These servers have computation and storage capabilities that help devices to not rely on internet cloud servers for these services. This reduces communication latency.

In order to overcome the latency problem in vehicular networks, Hagenauer et al. [8] introduced the Vehicular Micro Cloud architecture inspired by MEC. This architecture proposes that the connected vehicles could be bundled together forming small clusters, as shown in Figure 1.1 called Vehicular Micro Clouds acting as Virtual Edge Servers at different geographical locations. As the number of connected vehicles on road increases, the pool of computing, storage, sensing and communication resources from these vehicles also grows bigger. A pool of resources, called the Virtual Edge Servers, can be used to aggregate and process the upstream data and cache the downstream data. This minimizes the bandwidth consumption in vehicular networks and enhances the scalability of the networks [8], [9].

The simulations conducted by Higuchi et al. [10] on feasibility of micro clouds and by Hagenauer et al. [8] on efficient data collection, test and verifies the micro cloud architecture and its services. A prototype, as emphasized by Kordon and Luqi [11], however, plays a crucial role in the early evaluation of the final product.

In this thesis, we have focused on the implementation of the prototype of vehicular micro cloud architecture [8] and its services. To visualize the clustering status at runtime, a visualisation tool has been implemented. At the end, we have evaluated the architecture and its services at different traffic densities and data sizes in Manhattan grid scenario.

In the prototype implemented, a vehicle or Access Point (AP) (which can be a base station or Roadside Unit (RSU)) is a multi-process entity implemented in C++

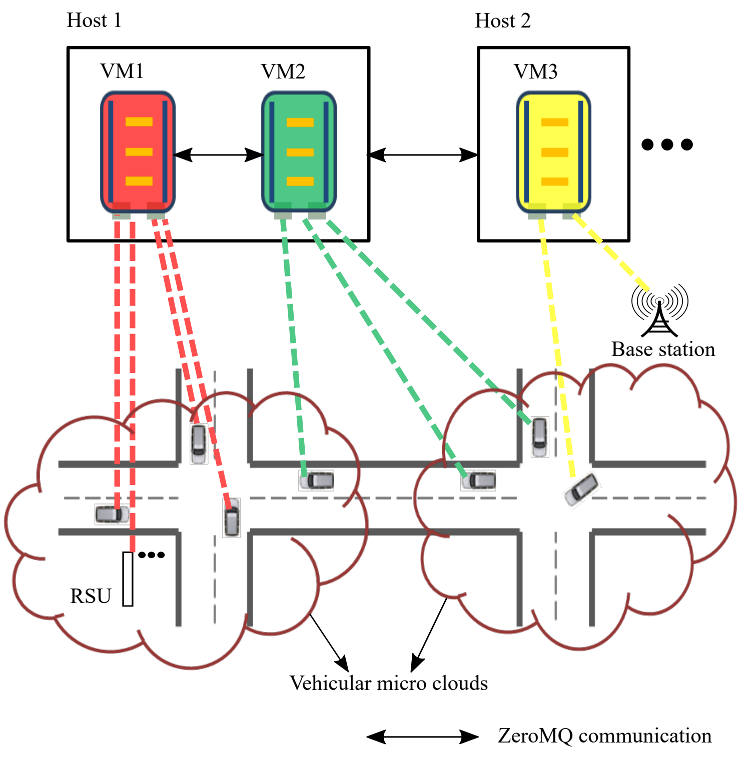


Figure 1.1 – Cars bundled together forming small clusters called vehicular micro clouds acting as virtual edge servers where upstream data is aggregated and downstream data is cached. The cars and access points are multi-process entities implemented in C++ programming language which can be run on virtual machines.

programming language. Each has multiple components such as sensor module, a communication module, clustering module and other application service modules. A Helper module has been implemented in order to provide vehicles with the latest sensor information and help them communicate with each other and with AP. As shown in Figure 1.1, vehicles and APs can be run on physical or virtual machines. The primary objectives of the prototype are:

- Validation and evaluation of vehicular micro clouds formation.
- Validation and evaluation of vehicular micro cloud services.
- Evaluation of usage of system resources such as CPU and memory.

The rest of the thesis is structured into different chapters as follows. Chapter 2 explains the fundamentals - the tools used in the implementation and the related work. Chapter 3 describes the design and the implementation of the prototype and Chapter 4 covers the validations and evaluations. Finally, the thesis is concluded in Chapter 5.

Chapter 2

Fundamentals

In this chapter, we explain the tools and libraries used in the implementation of the prototype - ZeroMQ, Protocol buffers and Chilkat library methods. In addition, the insights to the related work is given.

2.1 ZeroMQ - Messaging Library

The implemented prototype includes multiple processes and the communication between them. This communication is enabled by ZeroMQ's asynchronous messaging library. The library provides the following features[1]:

- supports multiple languages across multiple platforms,
- multiple messaging patterns like Push-Pull, Request-Reply, Router-Dealer and Publisher-Subscriber,
- easily scalable for distributed or concurrent applications from intra-application to inter-application communication,
- zero or low latency, and
- open source library backed by an active open source community.

In order to fulfil the different messaging requirements in the prototype, the following four message patterns[2] have been used in four different types of use cases.

Push-Pull The messages from the push socket end are distributed to load balance on all pull socket workers in Round-robin fashion.

[1]https://www.gridprotectionalliance.org/docs/WhyZeroMQ.pdf
[2]http://zguide.zeromq.org/php:chapter2

Req-Rep	Multiple clients are allowed to send requests/talk to one server asynchronously supporting two-way communication.
Pub-Sub	The publisher socket distributes the same message to all subscriber sockets.
Simple Mad Box	In Pub-Sub pattern, the subscriber is broken into a multi-threaded design, wherein the one set of threads is busy in reading messages and the other set is processing the incoming messages.

The ZeroMQ library has been used in various industrial and experimental research applications. Meng et al. [12] have used ZeroMQ to build a messaging mechanism for industrial IoT applications that requires machine interaction. The cross-platform nature, the flexibility to interact between the different software platforms and its efficiency have made ZeroMQ's messaging mechanism a promising tool for machine communication and data sharing. Gougeaud et al. [13] have used ZeroMQ as a solution to the synchronization problems occurring in a simulation tool called "OGSSim" which is used to study the behaviour of large-scale data storage systems. According to the results, ZeroMQ along with data communications have solved the synchronization problems caused due to the parallelism in the tool. Laux et al. [14] also use ZeroMQ for messaging in their Open Source experimental and prototyping platform that provides vehicular networking solutions.

The prototype we have implemented is also an application based software where several applications communicate with each other to transfer messages and data. Mainly, the different messaging patterns and the flexibility of support for multiple languages has been exploited in the implementation to communicate easily between the C++ and Python methods.

2.2 Protocol Buffers

In the prototype implemented, ZeroMQ messaging uses data serialization. It has mainly two uses - first when a message is transferred from source to destination, it helps to recover the original structure of the message at the destination. Second, it minimizes the data size, thereby reducing the bandwidth requirement. Protocol buffers[3] is one such method used to serialize and deserialize the structured data using a simple "interface description language". Using this language, a structure of the data is defined in a .proto file, and a program will generate source code of this structured data which is later used to generate and parse the stream of bytes representing the structured data.

[3]https://developers.google.com/protocol-buffers/docs/overview

The following features of protocol buffers make it easier for programmers to decide using protocol buffers over other serialization tools.

- Provides a code generator for many different languages such as C++, Java, Python, Ruby, Javascript.
- Simpler - once data is structured, programmatically it is very easy to populate, serialize and parse the data.
- Smaller - the serialized data consumes very less space compared to other serialization tools like JSON and XML.
- Faster - many online resources[4] [5] claims that performance is far better than JSON and other serialization tools.
- Available as an open source tool.
- Backward compatible.

However, Google uses protocol buffers as their common language for structured data. They use it in almost all their projects[6]. It is not only used as a serialization tool in their RPC systems and networking but also as persistent storage in various storage systems like data analysis pipelines, mobile clients etc.

2.3 Chilkat API Library

Chilkat software[7] provides API libraries for many different protocols and algorithms. In the implementation, data is transferred in fragments which requires the huge files to be split into multiple files or fragments at the sender side. These fragments have to be re-assembled into the original file at the receiver side. One of the libraries provided by Chilkat - CkFileAccess[8] which is a no licence required tool, is used in the implementation to split and re-assemble the huge files. The Chilkat API Library provides the following features:

- supports multiple environments such as Windows, Linux, MAC OS X, iOS, ARM Linux, Raspberry Pi,
- supports different programming languages such as C/C++, Python, C#, PHP, VB.NET, and
- provides many no licence required API libraries like CkFileAccess.

[4]https://auth0.com/blog/beating-json-performance-with-protobuf/
[5] https://dzone.com/articles/is-protobuf-5x-faster-than-json
[6]https://opensource.google.com/projects/protobuf
[7]https://www.chilkatsoft.com/
[8]https://www.chilkatsoft.com/refdoc/vcCkFileAccessRef.html

2.4 Related Work

2.4.1 Mobile Edge Computing

The mobile devices such as Tablets, Smart Phones and Laptops have become a platform for complex applications such as gaming, healthcare [15] and education [16]. Although, nowadays, mobile devices are much powerful in terms of computing, still not enough to compute huge load for complex applications. In order to address this issue, many computation offloading techniques such as [17] have been proposed where computation is leveraged to internet clouds. However, these techniques implicate higher communication latency since data has to move to and fro between the mobile devices and the internet clouds.

As a solution to such challenges, initially, Fog computing was introduced. It is a concept of edge computing introduced by Cisco. It bridges the end devices and the cloud servers by providing services such as computing and storage to the end devices at the edge of the network [18]. However, as mentioned in [7], it lacks in providing QoS as the computing is not integrated into the mobile network architecture.

To overcome this drawback, a concept of integrating the edge computing into the mobile network architecture developed by industry specification group (ISG) within European Telecommunications Standards Institute (ETSI) called Mobile Edge Computing [7], [19] was introduced. MEC is an emerging technology that brings high computation power and storage capacity in the vicinity of the mobile devices. In MEC, the Mobile Edge Servers having higher computation and storage capabilities are deployed at the edge of the networks. These edge servers enable mobile devices to access the cloud capabilities in their proximity thereby reducing the communication latency. Also, MEC is used to optimise the data before uploading to the internet clouds [19].

2.4.2 Vehicular Clouds

In the initial days, the concept of vehicular clouds was introduced by Gerla [9], Olariu, Hristov, and Yan [20], and Abuelela and Olariu [21]. According to Gerla [9], nowadays, mobile devices in terms of both mobile phones and vehicles have increased, which are used as both data servers and data consumers. Consider an example when the data of interest for consumers is of local relevance e.g. restaurant recommendations. This type of data should be processed and stored locally for example in vehicular clouds, rather than internet data servers. Otherwise, for a user, it would not be cost-effective in terms of both time and bandwidth to upload and download this data from internet data servers. Thus, in [9], the necessity of the vehicular clouds is very well outlined.

Further, Gerla et al. [22] explains the importance of vehicular clouds in terms of its requirements in autonomous vehicular applications. The authors emphasize how vehicular clouds can be an asset for efficient communication between the vehicles. Vehicular clouds also provide a computing environment that can enhance the scope of vehicular applications. Eltoweissy, Olariu, and Younis [23] coined the term "Autonomous Vehicular Clouds" that emphasizes the importance of vehicular clouds in various sectors in traffic management and asset management scenarios.

Later, Gu, Zeng, and Guo [24] introduced an architecture for vehicular cloudification. Here, authors quote smart vehicles as "Computer-on-wheels", as the automotive developers, nowadays, have altered the purpose of vehicles from only transportation to sensing the surroundings, and communicating, processing, and storing the data. In supporting these technologies, a typical smart car contains a processor, a storage device, GPS, and other various in-car sensors. The architecture which authors have introduced underpins the possibilities of two types of clouds based on the mobility of the vehicles, i.e. mobile and static. Mobile clouds offer data carrying service whereas static clouds, that are the parked vehicles, offer temporary storage service.

Lee et al. [25] discuss a vehicular cloud architecture and its design principles - how vehicular cloud model combined with information-centric networking could be an efficient way of vehicular networking. The vehicular cloud model addresses the problem of how content can be produced, maintained and consumed. Whereas, information-centric networking focuses on how to direct the content to the consumers. However, Ahmad et al. [26] go one step further and discusses the security aspects of the vehicular clouds, the possible threats, and their impacts on the vehicular cloud components.

Later, Hagenauer et al. [8] emphasized the importance of micro clouds (cluster of cars) through their simulations using Veins LTE simulation framework [27] in a small-scale. However, Higuchi et al. [10] use realistic vehicle probe dataset from the city of Luxembourg [28] and a major intercity highway in Japan to study the feasibility of micro clouds. Their results show the possibilities of forming both static and on-demand mobile micro clouds in different locations in the city, and the availability of micro cloud services to other vehicles which are not part of any cloud.

2.4.3 Vehicular Clustering

One of the challenges in vehicular cloudification would be the formation of clusters. Cooper et al. [29] outlined the general steps for cluster formation and have discussed the strategies for cluster head election, cluster membership and cluster maintenance. The steps can be generalized as follows:

- *Vehicular local info collection:* In order to determine clusters, the local information of vehicles needs to be exchanged, either between the cars if the clustering strategy is distributed (decision taken by cars) or, between the cars and AP if the clustering strategy is centralized (decision taken by AP). This control information could be the position, direction, speed, or any other parameters describing the status of the cars.

- *Cluster computation:* AP or car, according to the clustering strategy used, identifies Cluster Head (CH) and Cluster Members (CMs) either in a centralized or a distributed fashion. The car elected as CH is responsible for the interaction with other clusters and AP.
- *Cluster info distribution:* Once the cluster is computed, the information should be sent to all other cars in the cluster through broadcast (and if required through an appropriate routing strategy).
- *Data collection and processing:* Once the above steps are finished, CH will start collecting the data from the cluster members, process the data, and if required, the data is uploaded to AP or data server.

In general, any clustering algorithm's responsibility is to determine the clusters as per the set of rules defined by the algorithm. The applications built on top of the clusters designate roles and responsibilities to CH and other cluster members. However, different clustering algorithms or strategies [30]–[32] have been proposed by different authors, where each of them is tailored to suit the specific application and there exists no general solution. In order to make readers understand the different vehicular clustering strategies available, Cooper et al. [29] have conducted a survey and presented the three main aspects of the clustering strategies. They are:

- the applications of the clustering algorithms,
- creating and maintaining the clusters for example how the CH is elected, how other vehicles are affiliated to this CH and how the interaction happens between the CH and other clusters, and
- comparison of these algorithms based on performance evaluation.

Meanwhile, location of micro clouds also plays a crucial role in efficient vehicular cloudification. In that perspective, Higuchi, Dressler, and Altintas [33] have presented a design mechanism to position vehicular micro clouds in an intelligent way. The authors propose to choose such locations where there is a consistency in the availability of the vehicular resources for the provision of micro cloud services.

However, in this thesis, a clustering algorithm called *map-based* proposed by Hagenauer et al. [8] has been implemented. This approach offers to consider the geographical features and suggests a suitable position to locate CH. Accordingly, if the cluster head is positioned at an intersection, it will have a good line-of-sight in multiple directions and can communicate better with the cluster members, as shown in Figure 2.1.

Till date, all research experiments happened only through simulations whereas, in this thesis, Vehicular Micro Cloud prototype will be implemented that could be deployed in real-world experiments with minimal efforts.

Prototyping has been common in vehicular networks/clouds. A prototype implemented by Kwak et al. [34] as a proof of concept exploits vehicular cloud concept for route planning. Vehicles in cloud, capture traffic images using onboard cameras and share them with each other. These images later get processed to produce a user-friendly route summary. Another prototype implemented by Häberle et al. [35] provides a platform for automotive application developers to prototype connected-cars (virtual vehicles). The platform allows developed telematics services to be tested, and thus favours a reduction in time-to-market for deployment.

Similarly, the prototype that we have implemented also allows developers to further implement vehicular applications or services, and test them on the vehicular micro cloud architecture.

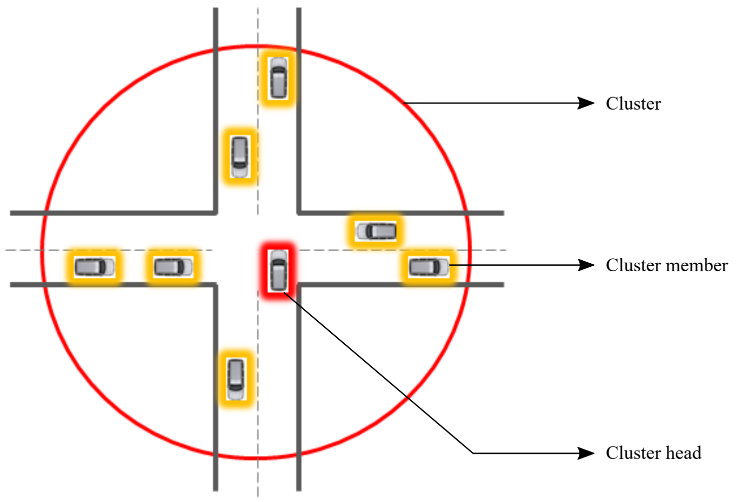


Figure 2.1 – Map-based clustering approach. In figure, red circled cluster is centered at the intersection. The red colored car which is closest to the intersection is the cluster head and other yellow cars within the cluster are the cluster members.

Chapter 3

System Design and Implementation

The previous chapter covered the fundamentals and methods on the basis of which the prototype has been implemented. In this chapter, we explain the overview of the architecture, the design and the implementation of the prototype, and the visual component.

3.1 Architecture Overview

The architecture shown in Figure 3.1 has mainly two types of nodes commonly called VMC nodes. The two types are vehicle nodes and AP nodes. An AP node can be a base station or a RSU. These VMC nodes run on multiple virtual machines (or physical machines). In other words, each virtual machine can have many vehicles and AP nodes running. The communication between the nodes is through the ZeroMQ messaging library. It can be categorized into three different types. They are, the communication

- between the concurrent applications within the vehicle node,
- between the vehicle nodes and between the vehicle and AP nodes, and
- between the vehicle node and the helper (details about helper given in Section 3.4) and between the AP node and the helper.

3.2 Vehicular Micro Cloud Node

A VMC node as shown in Figure 3.2 represents a vehicle or a base station or a RSU. A node has several modules such as communication, GPS, speed, etc. that help to carry out different responsibilities. Each of these modules contributes to the overall functionality of a node. The modules are independent and they are implemented

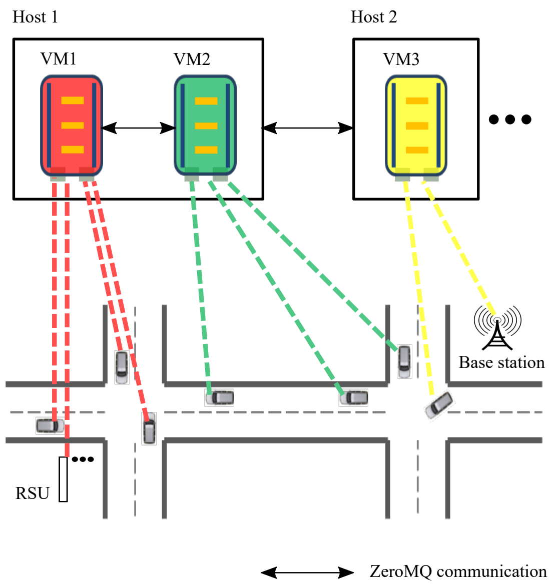


Figure 3.1 – Architecture overview - the vehicular micro cloud nodes (vehicles and AP) can be run on one or multiple machines.

as separate processes. All the modules corresponding to one VMC node are bound to run on one virtual machine. Each module is explained in detail from both the Vehicle and AP perspective.

3.2.1 Communication Module

The communication module in both Vehicle and AP nodes is used for sending and receiving messages. It is an abstraction of the Network Interface Card. The nodes

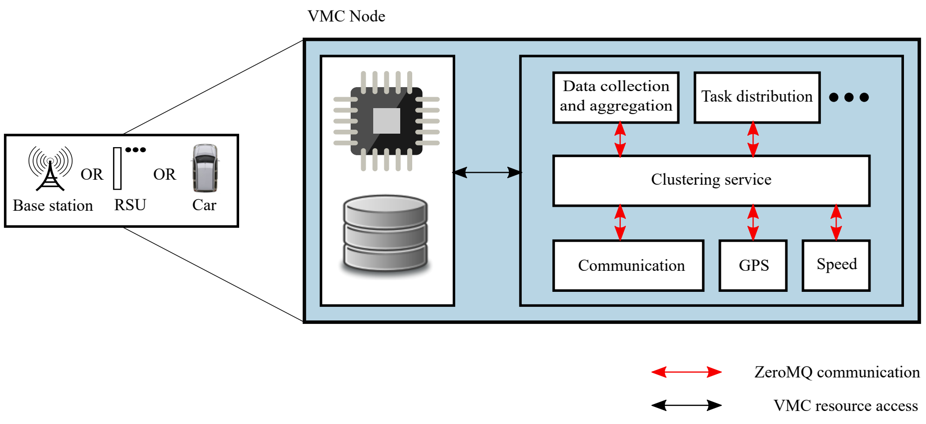


Figure 3.2 – VMC Node representing a vehicle or an AP. The communication mechanism between the concurrent applications within a node uses ZeroMQ library.

communicate with each other using two types of messages, i.e. unicast and broadcast messages. Whenever a node wants to send a message, the communication module helps by adding the additional fields to the data, required to forward the message. The additional fields include the source ID, the destination ID and the message type. The unicast or broadcast message contains the following fields.

MType	Message type indicating the message is a unicast or a broadcast message,
SRC_ID	The ID of the sender node,
SRC_TYPE	The source type indicates the sender node is a vehicle or base station or RSU,
DEST_ID	The destination ID is the ID of the node the message to be sent to,
SUB_DEST_ID	The ID of the application in the node to which the message is intended to be delivered,
DATATYPE	Datatype indicates the type of message. The type, for example, could be the local info (GPS and Speed), the RSU beacon, cluster info,
DATA	The actual data to be transferred,

Whenever a new message arrives at a node, the communication module receives this message and delivers it to the right application with the help of `DEST_ID` and the `SUB_DEST_ID`. Thus, the communication module helps VMC node to interact with the other VMC nodes in the network.

3.2.2 GPS And Speed Modules

Each vehicle has a GPS module and a speed module from which the applications fetch the latest location and the speed information. These modules periodically fetch the information from the in-car sensors (the in-car sensors details are given in Section 3.4.2) and store the updated values.

These modules are two independent multi-threaded processes that enable them to communicate with the in-car sensors and the applications individually and in parallel. In the case of AP nodes, the GPS value has been set to constant and the speed value set to zero.

3.2.3 Clustering Service Module

The fundamentals of the clustering service are covered in chapter 2. Here we explain the implementation details. Firstly, the details are explained from the vehicle's perspective.

3.2.3.1 Clustering Service In A Vehicle

Each vehicle has two types of cluster managers:

1. `default_cluster_manager` - an AP to which vehicle periodically updates its GPS and speed information by default. This is normally a base station.
2. `current_cluster_manager` - an AP to which, currently, vehicle periodically updates its GPS and speed information.

The clustering service in a vehicle node has mainly two responsibilities. They are:

- Update local information to AP - Each vehicle updates its local information such as location and speed to its `current_cluster_manager` periodically. This local information is periodically fetched from the node's GPS and speed modules.
- Cluster management - In a vehicle node, the cluster management's role is to store and maintain the `current_cluster_manager` to which the vehicle periodically sends its local information. Initially, it is set to `default_cluster_manager` of the vehicle which is by default a base station. If the vehicle enters a region where there exists a RSU it starts receiving the RSU beacons which indicate the RSU's existence. Now the vehicle changes its `current_cluster_manager` to the RSU's ID retrieved from the beacons. Then the vehicle will continue sending its local information to its new cluster manager until it misses out two consecutive RSU `beacons` in which case the vehicle will set its `current_cluster_manager` back to `default_cluster_manager`.

  Also, based on the cluster information received from the `current_cluster_manager`, the vehicles identify themselves in 3 states - INIT, CH and CM. If a vehicle does not belong to any cluster it will be in INIT state, or if a vehicle is a Cluster Head of some cluster, it will be in CH state or if it is a Cluster Member of some cluster it will be in CM state until the next cluster information is received.

3.2.3.2 Clustering Service In An AP

From an AP's perspective, the clustering service module has two roles. They are:

- Vehicle local info table maintenance - On receiving the local information from cars, AP stores the information in the vehicle local info table. The table contains details such as ID of the vehicle, geographical coordinates of the vehicle, speed of the vehicle and system time at which the message was received.

  Each entry in the table corresponds to each vehicle whose information is sent to AP. This entry will be removed later from the table for a vehicle if AP does not receive 2 consecutive such local information messages assuming either vehicle has gone out of the communication range or exited from the network. Normally a base station will be set as the `default_cluster_manager` for all the vehicles. It builds the vehicle local info table on receiving the local information from vehicles. In the case of RSU, if it is managing a micro cloud, it keeps sending out beacons to advertise its existence. Whenever vehicles come in the communication range of the RSU they send their local information to RSU and RSU builds its vehicle local info table. A RSU beacon contains RSU's ID and its geographical coordinates.

- Cluster computation - As AP starts populating its vehicle local info table, periodically it computes clusters. The cluster computation is identifying the vehicles that belong to different micro clouds and sending the cluster information message to the vehicles which are part of the cluster. The message contains details such as micro cloud's ID, CH, CM list, total number of cluster computation intervals elapsed.

  In order to compute the clusters, a *map-based* algorithm [8] has been implemented. The algorithm uses two tables - the vehicle local info table and the micro cloud table. A micro cloud table holds the information about the micro clouds that the AP is managing. Each micro cloud table entry contains details such as micro cloud's ID, geographical coordinates and radius.

  The algorithm as shown in Algorithm 3.1 iterates through vehicle local info table and micro cloud table to calculate the euclidian distance between the geographical coordinates of the vehicles and the micro clouds. If the distance is less than or equal to the micro cloud radius the vehicle is included in the cluster members list and among them, the vehicle closest to the intersection is elected as the CH.

```
Input: Vehicle Local Info Table {$v_{id}$,$v_x$,$v_y$} where $v_{id}$ is the vehicle ID, $v_x$ and $v_y$ are
    position coordinates of vehicle $v_{id}$, MicroCloud Table {$m_{id}$,$m_r$,$m_x$, $m_y$} where $m_{id}$
    is micro cloud ID, $m_r$ is the micro cloud radius and $m_x$ and $m_y$ are the position
    coordinates of micro cloud $m_{id}$
Output: Computes Clusters ($C$)
 1: for all $m_{id}$ do
 2:   $d_{max} \leftarrow$ FLOAT_MAX
 3:   $c_h \leftarrow$ EMPTY_STRING          ▷ Initially assuming no member in the micro cloud
 4:   for all $v_{id}$ do
 5:     $d \leftarrow$ cal_euclid_dist($v_x, v_y, m_x, m_y$) ▷ $d$ is the euclidian distance between
        the vehicle and intersection (or micro cloud coordinates)
 6:     if $d \leq m_r$ then
 7:       $c_m \leftarrow v_{id}$                        ▷ populates cluster members
 8:       if $d < d_{max}$ then
 9:         $d_{max} \leftarrow d$
10:         $c_h \leftarrow v_{id}$                      ▷ identifies cluster head
11:       end if
12:     end if
13:   end for
14:   if $c_h \neq$ EMPTY_STRING then
15:     $C[m_{id}][c_h] \leftarrow c_m$ ▷ If there exists at least one member in the micro cloud put
        the cluster info in the container
16:   end if
17: end for
```

Algorithm 3.1 – Cluster computation

3.2.4 Data Collection and Aggregation

Data collection and aggregation is an application built on top of the clustering service. As discussed in Chapter 2, one of the main advantages of micro clouds is that it aggregates data before it uploads it to the data server. In the implementation, AP is acting as the data server for collecting and storing the data from the micro cloud.

3.2.4.1 Data Collection

Each CM periodically generates data and sends it to CH. If the data size is greater than the fragment size (1024 bytes), the data is sent in fragments. On the other side, the CH collects this data and stores it. In the implementation, the cluster members first generate data using simple C++ file-based operations and store it in a file. This file data is then split into fragments using a Chilkat library method *SplitFile()*

and are sent to CH through unicast messages. On receiving these fragments, CH will reassemble the fragments into the original file using Chilkat library method *ReassembleFile()*.

3.2.4.2 Data Aggregation

The data aggregation is reducing the size of the data by removing the unwanted and redundant data from it. In the implementation, the data aggregation is performed by reducing the total data size to a specified aggregation percentage. Data aggregation is performed by CH periodically before it uploads the collected data from the CMs to AP. And, when the state of the vehicle changes from CH to CM or INIT, if it has data collected from the CMs in the previous cluster computation interval and not uploaded, it will aggregate and upload that data to AP.

3.2.5 Task Distribution

Task Distribution is another application built on top of the clustering service. Whenever CH of a micro cloud wants to compute a task it may seek other vehicle nodes which are the CMs of that micro cloud for the task completion assistance. In the implementation, the CH of each micro cloud periodically creates a task (virtual task - requires no real computation). The task is split into as many subtasks as the number of current CMs in the micro cloud. These subtasks are sent to the CMs through unicast messages. The message consists of details of the subtask and *max_time* - time (in *ms*) before which the subtasks should be computed. Upon receiving a subtask request, the CM sets a random timer between 1 to *max_time* (in *ms*). Once the timer is elapsed, the CMs will send the results (virtual results) to CH.

At the other side, after assigning the subtasks, the CH waits for a specified interval for the subtask results. Once the interval is expired, it combines the results of subtasks to produce the result of the intended task. Meanwhile, there are chances that within the specified interval CH may not receive some subtask results, but such cases are ignored. Thus CH seeks computing assistance from its CMs.

3.3 Config Files

All the configurable parameters required in the implementation have been stored and structured in the XML file format. The config files in our implementation have been categorised into 6 different categories as given below:

1. AP - stores details of
    - AP - ID and position,

- cluster computation algorithm - cluster computation interval and vehicle local info table entry expiry interval, and
- micro clouds - ID, the ID of the APs which is managing it, radius and geographical locations.

2. car - stores details such as local info update to AP interval, default cluster manager.
3. common - stores details required by both car and AP.
4. data collection and aggregation - stored details such as CM to CH data update interval, CH to AP aggregated data update interval.
5. visuals - stores details required by the visual component.
6. logs - stores details required by the logging utility.

3.4 Parser, Vehicular Sensors and Unit Disk Model

In this section, we explain how two nodes are able to communicate through unicast and broadcast messages. Also, how vehicles get their sensor information such as GPS and speed. A helper module as shown in Figure 3.3 has been implemented in order to achieve these functionalities. It consists of 3 components.

1. Parser
2. In-car sensors
3. Unit Disk Model

When the helper module is created, first it accumulates the details of all the stationary modules such as AP nodes and micro clouds by parsing the XML config files. By this, the helper module is aware of all the details about the AP nodes such as their position, communication range, IP address, port address and the details about the micro clouds such as their position and the radius. All these details are stored in a hash like data structures and can be accessed by all the helper components.

3.4.1 Parser

Mobility Trace stores local information such as GPS and Speed of the vehicles at every Timestep. In the implementation, Floating Car Data has been used as a Mobility Trace, but in general, it could be replaced with SUMO traffic in runtime or any other mobility trace. A parser has been implemented to parse this trace at every timestep and retrieves the following information of the vehicles:

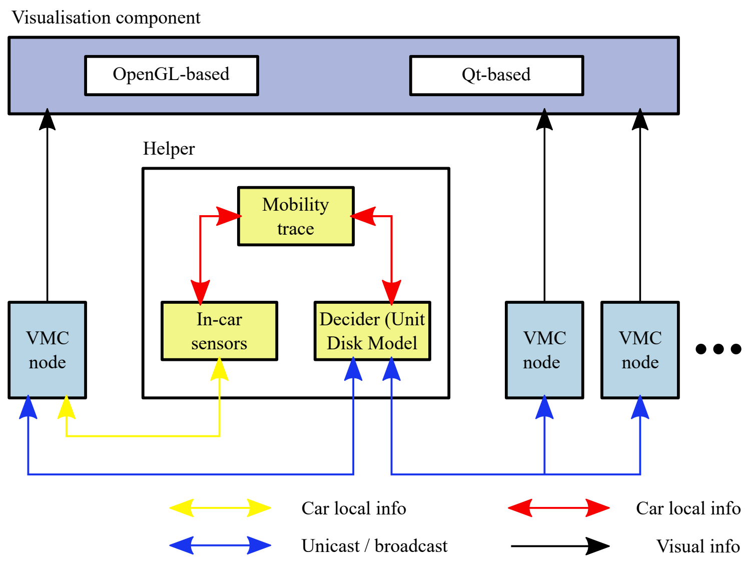


Figure 3.3 – The Communication architecture showing how the different ZeroMQ messages flow in the prototype.

- the IDs of the vehicles that are currently active in the network, and
- the local information of those vehicles.

At every timestep, these details of the vehicles keep changing. According to it, the parser performs two action sequences at every timestep. First, it checks if a new vehicle has joined the network. If yes, it initiates the creation of a new vehicle node in a virtual machine. The virtual machines are selected in a Round-robin fashion from the given pool of machines. Second, the parser checks if any vehicles have exited the network. If yes, it sends termination messages to those vehicles and they get terminated.

3.4.2 In-Car Sensors

The in-car sensors serve as the GPS and speed sensors for all the active vehicles. The ZeroMQ server sockets enable in-car sensors to receive the requests from the vehicles for their local information. On receiving the requests, the sensor module first identifies the ID of the vehicle which has sent the request and checks if the vehicle is active in the network. If yes, then checks whether the request has been made for GPS or speed. Accordingly, it accesses the data of the vehicles stored by the mobility trace and sends the appropriate updated response to the vehicles.

3.4.3 Unit Disk Model

Unit Disk Model (or the decider) assists the VMC nodes to communicate with each other. Whenever a vehicle or AP node wants to send a unicast message or a broadcast message through their communication module, first the message is sent to the decider. Then the decider will decide if the message should be forwarded to the destination node or not. The communication module in the nodes uses push sockets to send their messages to the decider and decider receive messages using the pull sockets. On receiving a message, the decider performs the action sequence as mentioned in the flowchart in Figure 3.4.

Decider uses the ZeroMQ publisher subscriber message pattern to forward messages. When the decider decides to forward a message to a node, it publishes the

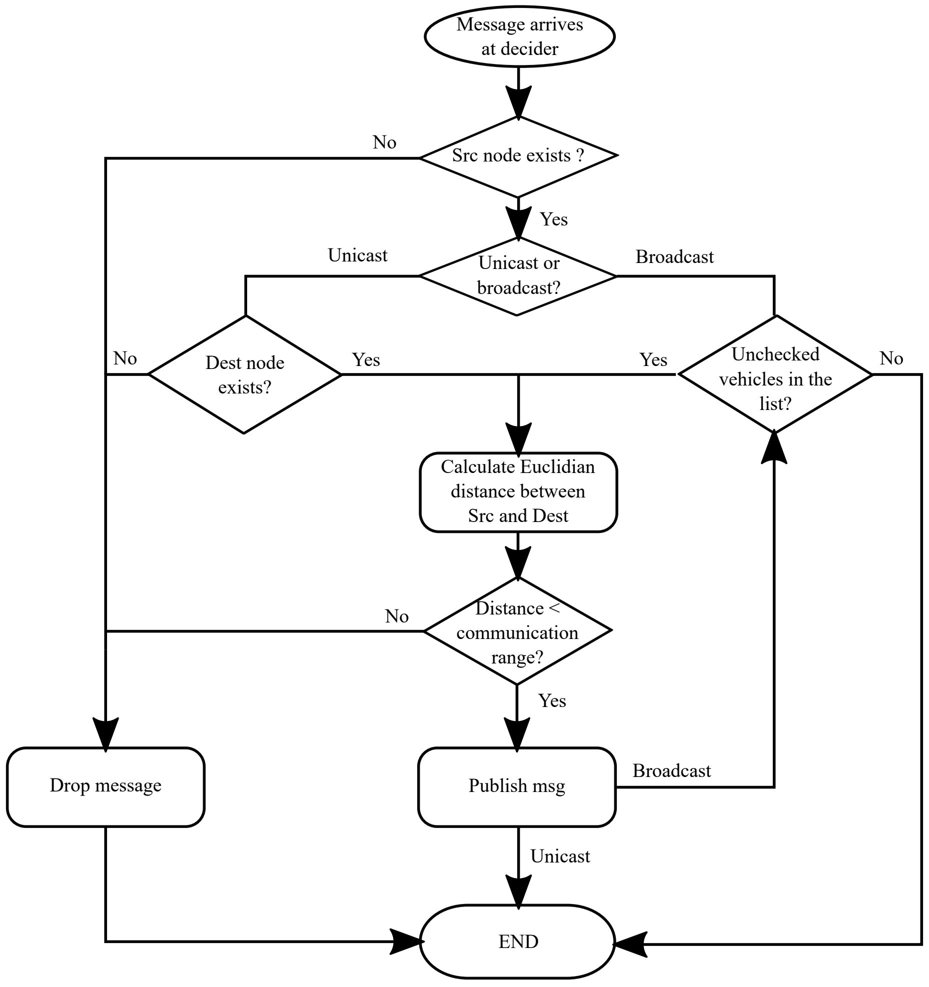

Message arrives at decider
No
Src node exists ?
Yes
Unicast
Unicast or broadcast?
Broadcast
No
Dest node exists?
Yes
Yes
Unchecked vehicles in the list?
No
Calculate Euclidian distance between Src and Dest
No
Distance < communication range?
Yes
Drop message
Publish msg
Broadcast
Unicast
END

Figure 3.4 – The sequence of actions that the unit disk model performs when a message is arrived for it to forward.

message to the node through the publisher socket with a topic set to the ID of the node or the ID of the module of the node. The node or the module of the node which is subscribed to the decider's publisher socket using its ID as the topic will receive the message and it will further process it. This is how the decider forwards a message to the correct destination.

3.5 Visualization Component

The visualization tool is mainly designed to visualize the cluster status in runtime particularly in Manhattan grid scenarios. The tool implemented has two components. One can be used to visualize graphically the movements of vehicles and the other can be used to visualize textual information. The graphical tool has been implemented using OpenGL and text form tool is implemented using Qt.

3.5.1 OpenGL based - Graphical Visualization

OpenGL based tool at the first step reads the details required to draw the scenario from the XML configuration file. In parallel, the tool starts listening to the information published by the AP through ZeroMQ sockets. AP sends information about the vehicles to OpenGL tool every second. On receiving the messages, the tool using OpenGL methods draw the vehicles at their positions. Along with the positional information of the vehicles, AP also publishes the status of each vehicle (INIT, CH or CM). Accordingly, the OpenGL tool changes the colour of the vehicles to indicate the different states of different vehicles.

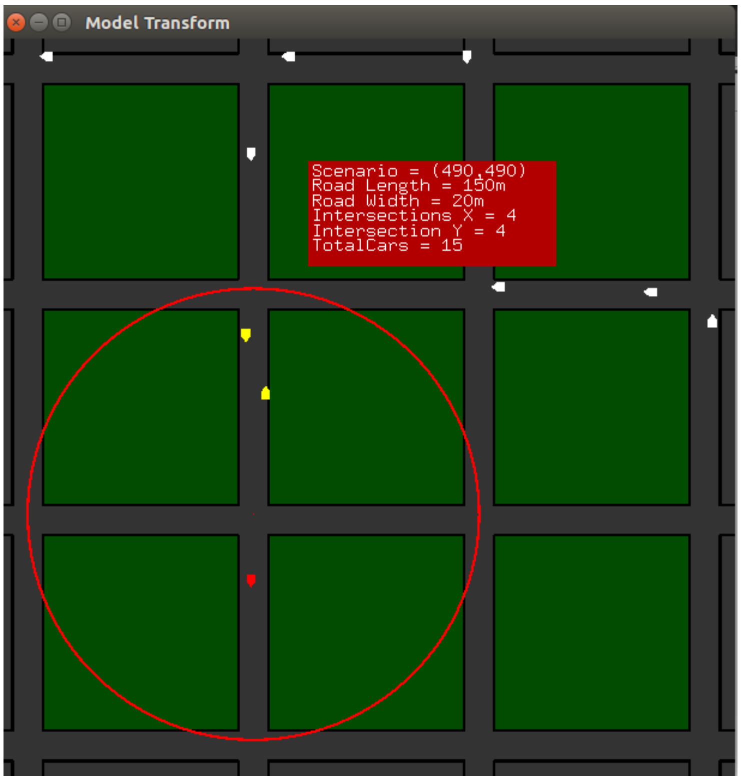
In brief, the following features are supported in the OpenGL tool.

Scenario details	The scenario details on a red banner as shown in Figure 3.5
Vehicle details	The current details of a car on a red banner are displayed. The details are ID, position, speed, direction, state and cluster ID if it belongs to some micro cloud.
Cluster status	The details of the micro cloud current status are displayed. The details are ID, position, manager (AP which is managing that micro cloud), current CH ID and CMs IDs.

3.5.2 Qt based- Textual Information

The Qt based tool along with the cluster status, in runtime, also shows the information about micro cloud services running on top of the clustering service in the vehicle. In brief, the following list shows the supported features in this tool.

List of nodes	List of IDs of currently active nodes includes both vehicle and AP nodes.
Total nodes	Total count of all the cars currently active in the network.
Cluster status	The details of the current status of the cluster.



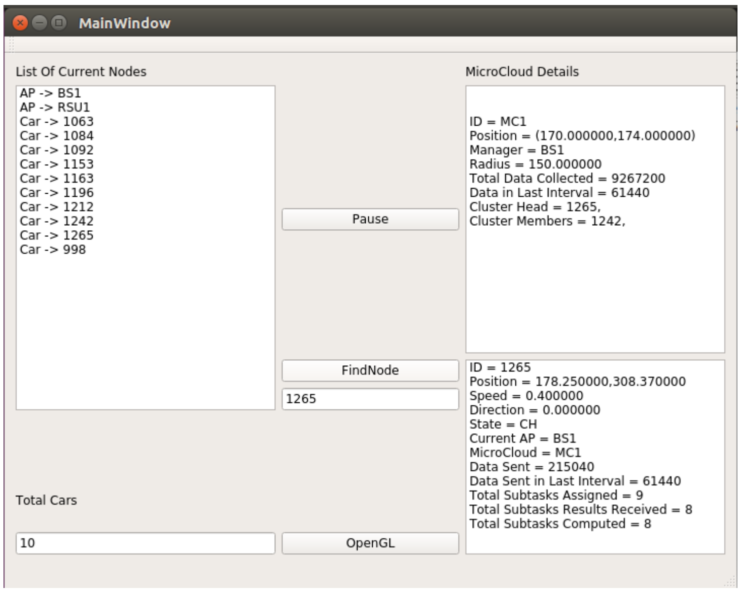
```
Scenario = (490,490)
Road Length = 150m
Road Width = 20m
Intersections X = 4
Intersection Y = 4
TotalCars = 15
```

Figure 3.5 – OpenGL based tool - the scenario details are displayed.

Find node	This is used to find the details of any active nodes in the network. Takes ID as the input and displays the cluster and application details of the node.
OpenGL	OpenGL button used to open OpenGL window
Pause	This button on the first click freezes the current status of the window, on the second click resumes the latest status of the window.

The values of the list of nodes and total nodes fields are refreshed every second and micro cloud details are refreshed every 5 seconds. In order to display this information the tool requires two ZeroMQ sockets. For the cluster status information, the tool subscribes to AP and AP publishes this information every one second. For the application information, the tool uses the pull socket, so that vehicles can push their latest application details using their push sockets.

For a quick understanding of how the different messages flow between the nodes or between the modules, one can refer to the sequence diagrams given in Appendix A.



```
List Of Current Nodes
AP -> BS1
AP -> RSU1
Car -> 1063
Car -> 1084
Car -> 1092
Car -> 1153
Car -> 1163
Car -> 1196
Car -> 1212
Car -> 1242
Car -> 1265
Car -> 998

Total Cars
10

1265

MicroCloud Details
ID = MC1
Position = (170.000000,174.000000)
Manager = BS1
Radius = 150.000000
Total Data Collected = 9267200
Data in Last Interval = 61440
Cluster Head = 1265,
Cluster Members = 1242,

ID = 1265
Position = 178.250000,308.370000
Speed = 0.400000
Direction = 0.000000
State = CH
Current AP = BS1
MicroCloud = MC1
Data Sent = 215040
Data Sent in Last Interval = 61440
Total Subtasks Assigned = 9
Total Subtasks Results Received = 8
Total Subtasks Computed = 8
```

Figure 3.6 – Qt based tool displays textual information about the cluster status in runtime and the application details.

3.6 Software Reusability

In the implementation the following modules are used as abstraction modules:

- In-car sensors - used as an abstraction for vehicular sensors that senses the vehicles local information such as GPS and speed.
- The Communication module - used as an abstraction for Network Interface Card.
- Unit Disk Model (or the decider) - used as an abstraction for vehicular physical layer.

Apart from these, the other modules in the implementation, can be used further to integrate with real sensors and real physical layer functionalities to conduct field tests.

The modules that can be re-used are:

- Clustering service module in vehicle.
- Cluster service module in AP.
- The micro cloud services - Data collection and aggregation, and Task distribution.

Wherever there was a requirement for abstraction, the abstraction classes has been implemented which can be easily overridden. For example, as shown in Figure 3.7, Communication class used for sending unicast and broadcast messages is an abstraction class of Base_communication which can be overridden to implement new methods for sending unicast and broadcast messages.

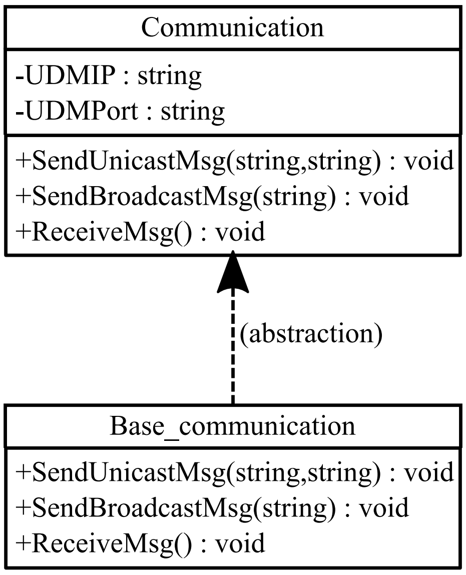

Communication
-UDMIP : string
-UDMPort : string
+SendUnicastMsg(string,string) : void
+SendBroadcastMsg(string) : void
+ReceiveMsg() : void
(abstraction)
Base_communication
+SendUnicastMsg(string,string) : void
+SendBroadcastMsg(string) : void
+ReceiveMsg() : void

Figure 3.7 – Communication class is an abstraction class of Base_communication class which can be overridden.

Chapter 4

Evaluation

After the prototype implementation, we have validated and evaluated the functionality and the performance of the prototype considering the important metrics that are closely related to the clustering and micro cloud services. This Chapter covers the details of metrics used in the validations and evaluations, the experimental setup, validations and evaluations results.

4.1 Metrics

The metrics have been classified into two categories - the validation metrics and evaluation metrics.

The following metrics have been used for the validation:

- *Cluster Size* - The total number of vehicles inside a cluster could be a validation metric to validate cluster computation.
- *Data Collection* - Comparison between the total amount of data collected at AP and the expected amount of data to be collected at AP helps in validating the data collection service in the implementation.

The following metrics have been used for the evaluation:

- *Dynamicity Of The Traffic* - The distribution of the number of consecutive times the cars elected as cluster head briefs the dynamicity of the traffic.
- *Cluster Computation Time* - The average of time taken by AP to compute clusters in a run helps to evaluate the time complexity of the clustering approach implemented in the prototype.
- *Data collection* - The total data collected at AP at different traffic densities.

- *Different aggregation percentage* - The amount of data uploaded to AP by CHs at different aggregation percentages used.
- *CPU utilization* - This is an important metric to measure system resource usage - the CPU utilization at different traffic densities.
- *Memory usage* - Another metric helps to understand the system resource usage - the memory consumption at different traffic densities.

4.2 Experimental Setup

For all the experiments, we have used virtual machines (except for system resource usage measurement) to run vehicle nodes, AP nodes and Helper. As shown in Figure 4.1 we ran Helper in one virtual machine, AP node(s) in one virtual machine and all other vehicle nodes were distributed among the multiple virtual machines.

4.2.1 Manhattan Grid Scenario

We have conducted all our experiments under Manhattan grid scenario. The scenario details in which most of the experiments conducted is given in Table 4.1. If the values are changed for any experiment they are explained in the appropriate section. The communication between the nodes is clear from any sort of noise or packet drops since we do not consider the wireless channel properties.

4.2.2 Configuration Parameters

The configurable parameters as mentioned in Table 4.2 were set to constant values (The values are chosen approximately closer to as specified in [8]) unless there is a requirement for a change in an experiment which is explained.

4.2.3 System setup

In all the experiments conducted except in measuring CPU utilization and Memory usage (where we have used 3 machines), 6 virtual machines have been used as

properties	value
Area	490 m^2
Grid size	3 × 3
Traffic densities	50, 100, 150 and 200 vehicles
Micro cloud position	(170m,174m)

Table 4.1 – Manhattan grid scenario details used in the experiments.

properties	value
Vehicle to AP Local Info update interval	1s
Cluster computation interval	5s
CM to CH data transfer interval	2s
CH to AP aggregated data transfer Interval	4s
Data size	5kB
Data fragment size	1kB
Experiment duration	2000s

Table 4.2 – The configurable parameters that are set to constant values in the experiments.

shown in Figure 4.1. 1 machine for the Helper, 1 for the AP node and 4 for the vehicle nodes. The vehicle nodes are distributed in a Round-robin fashion from the given pool of virtual machines as shown in Figure 4.2.

4.3 Validation

4.3.1 The Cluster Size

Cluster size - the number of vehicles in the cluster keeps varying with time because at different point of time at different intersections the vehicle density keeps varying. The experiment was conducted at a traffic density of 200 vehicles. In this experiment,

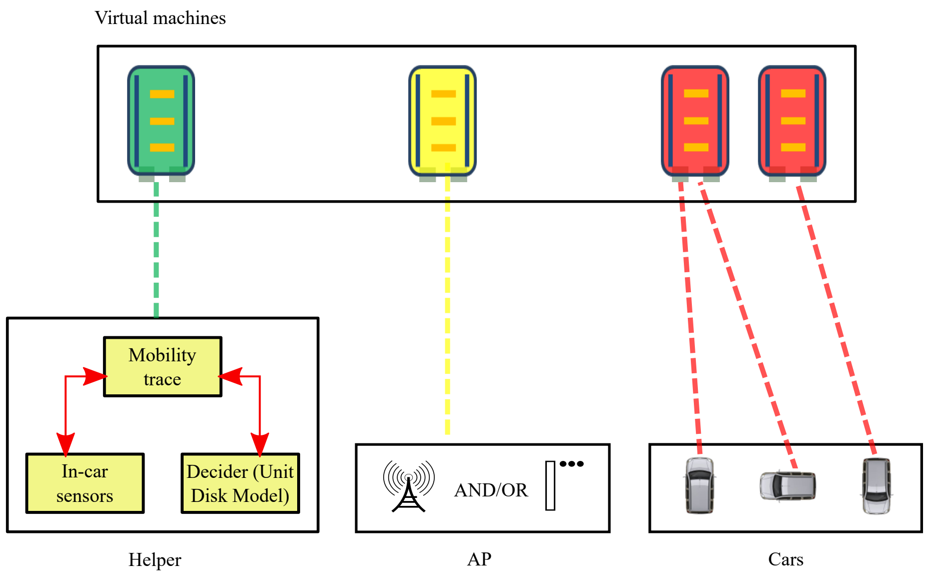


Figure 4.1 – System setup - all experiments are conducted by running Helper on 1 machine, AP node on 1 machine and vehicle nodes on multiple machines.

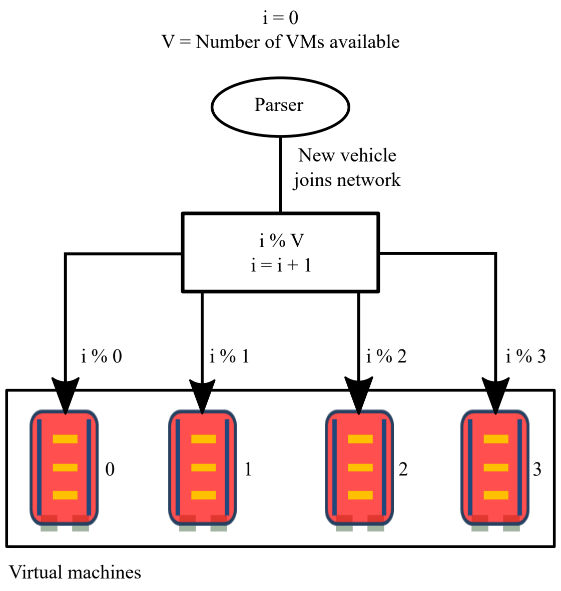


Figure 4.2 – Multiple vehicle nodes distributed on multiple virtual machines in a Round-robin fashion.

we first calculated the number of vehicles in the cluster at every timestep directly from the mobility trace with the time precision of one second. Then, we calculated the number of vehicles in the cluster at every cluster computation interval during the experiment with the time precision equal to one cluster computation interval (5s). The AP node logs this information as it computes clusters.

The plot shown in Figure 4.3 showcase the number of vehicles in the cluster in the y-axis and the time(s) over x-axis. The two lines in the plot are the values calculated from the mobility trace and the experimental results. We can interpret two things from the plot. First, the number of vehicles in the cluster varies over time differently. Second, the two lines are almost overlapping meaning the cluster computation at each cluster computation interval is correct.

In order to validate the multiple micro clouds at higher traffic density in one scenario, we ran the same experiment by deploying 3 micro clouds managed by 3 RSUs positioned exactly as the micro clouds. For that we increased the scenario size and the traffic density. The scenario size was increased to 1390 m^2 (the road length and width remained the same) and traffic density to 500 vehicles. The resulting plots are shown in Figure 4.4. In all the 3 micro clouds the lines are observed to be overlapping and hence validates cluster computation in 3 micro clouds scenario at a higher traffic density.

When we first conducted this experiment, the implementation line in the plot was lagging with some delay compared to the line corresponding to mobility trace. The reason is explained here. Mobility trace has timesteps in the interval of 0.1s.

Hence in the parser, after each timestep, a delay of 0.1s was specified. This caused the experiments to run with some delay because the parser at each timestep also spends some time on processing the vehicles details. In order to fix it, we specified parser delay to be $(0.1 - S_t)s$ where S_t is the time in seconds that parser spends on processing vehicle details at timestep t, and then we observed zero or negligible delay in runs.

4.3.2 Expected Amount Of Data Collected

From the previous experiment, we get the statistics of the number of vehicles in a cluster at every cluster interval. Based on this we can calculate the expected amount of data to be collected at a micro cloud or AP in one run. We calculated the expected amount of data using the equation

$$A = \sum_{i=1}^{n} (N_i \times D_S \times C)$$

where A is the expected total amount of data, n is the total number of cluster computation intervals, N_i is the number of vehicles in a cluster at cluster computation interval i, D_S is the size of the data CM transfers to the AP and C is the number of times the CM transfers the data to CH in one cluster interval. For easy theoretical calculations in this experiment, the cluster interval was set to 6s (multiples of CM to CH data transfer interval) and CM to CH data upload interval to 2s.

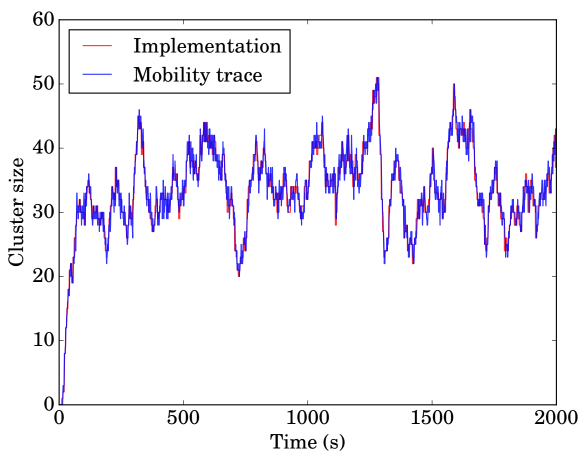
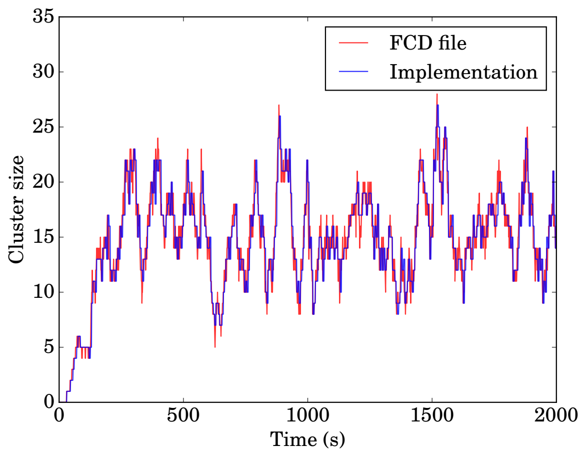
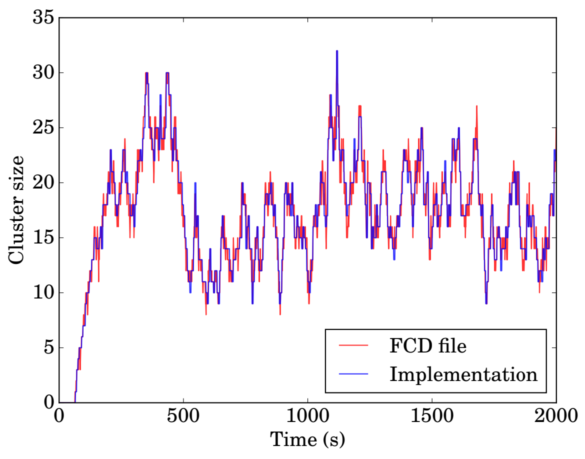


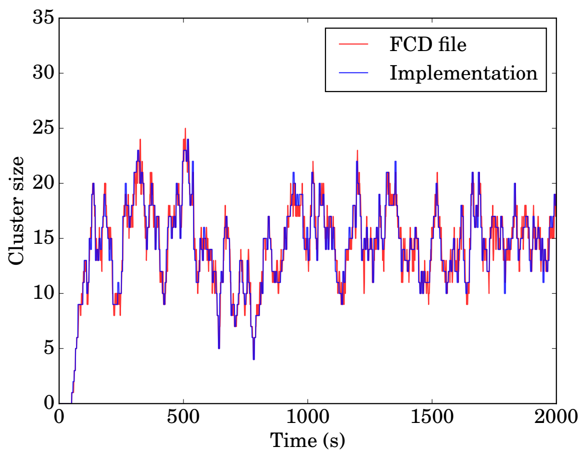
Figure 4.3 – Plot showing the number of cars in cluster computed from mobility trace (time precision = 1s) and experiment (time precision = cluster interval(5s)) at traffic density 200 vehicle nodes. The lines almost overlapping validates the cluster computation in the implementation.



(a) Micro cloud 1



(b) Micro cloud 2



(c) Micro cloud 3

Figure 4.4 – Subfigures showing the number of cars in 3 different micro clouds - Micro cloud 1, Micro cloud 2 and Micro cloud 3 managed by 3 RSUs in a scenario, computed from mobility trace and from experiment vs time (traffic density = 500 vehicle nodes). In all 3 subfigures, the lines almost overlapping validates the cluster computation in multiple micro clouds scenario.

Later, the amount of data collected at AP in experiments was determined by running experiments with different traffic densities while AP was logging the amount of total data collected. In the experiment, CH at each interval will upload all the data it collected from the CMs to AP without aggregation.

Figure 4.5 shows the results plot from which we can observe that there is a slight difference between the expected and the experimentally collected amount of data. This is because within an interval there is always a possibility that CM or CH will go out of communication range from each other. In that case, the Unit Disk Model will drop CM's unicast data packets which is not accounted while calculating the expected amount of data.

We can also infer from the plot that, as the traffic density increases the difference in the amount of data between expected and actually collected also increases. This is because when the traffic density increases there are more chances of such incidents to occur and packets gets dropped. The error bars in the plot shows the 95% confidence interval calculated over 3 runs.

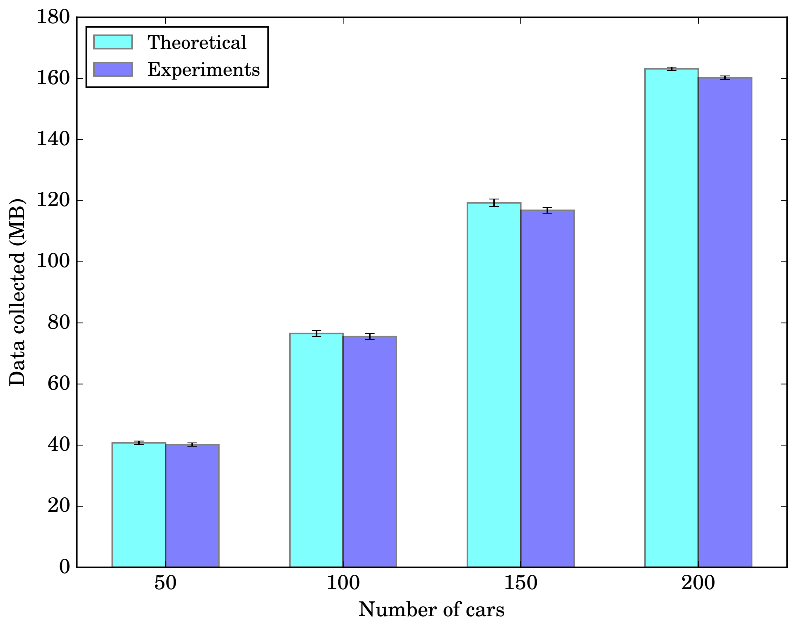


Figure 4.5 – Graph showing the difference between the expected amount of data collected and the actual amount of data collected at a micro cloud in an experiment. The difference is because CM or CH may go out of communication range within an interval and the decider drops the data packets in that case. The error bars in the plot shows the 95% confidence interval calculated over 3 runs.

4.4 Evaluations

4.4.1 Dynamicity of the traffic

In this experiment, we evaluated the dynamicity of the traffic by determining the consecutive times a car is elected as CH at different traffic densities - 50, 100 and 200 vehicles. The required information was logged by AP node.

The plot shown in Figure 4.6 underlines that as the traffic density level increases the consecutive times a vehicle elected as CH decreases. It can be observed that at traffic density level 200, a vehicle has never elected as CH for 3 consecutive times and at the traffic density level 100, a vehicle has never elected as CH for 4 consecutive times. But, at the traffic density level 50, a vehicle for once has been elected as CH for 6 consecutive times. The reason is that either the vehicle is waiting in a signal and remained closest to the intersection for a long time or there were no other vehicles inside cluster which were closer to the intersection at that point of time. This shows the dynamicity of the traffic at different traffic density levels.

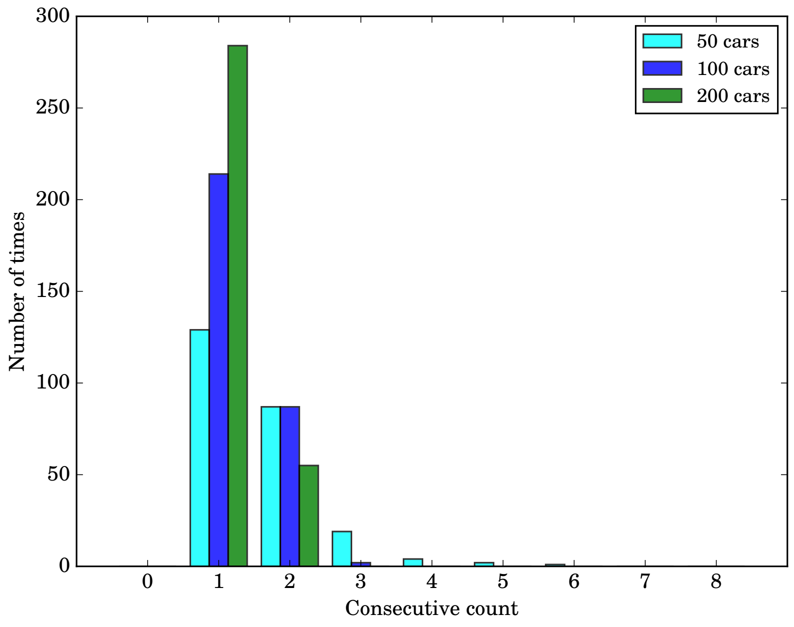


Figure 4.6 – The distribution of the consecutive times the cars re-elected as CH defines the dynamicity of the traffic.

4.4.2 Cluster Computation Time

In order to evaluate the clustering approach, we conducted the experiments at different traffic densities and logged the time taken for cluster computation at every cluster computation interval and then calculated an average of all.

The resulting plot shown in Figure 4.7 showcase average time in the y-axis and the traffic density levels in the x-axis. We can underline two results in the plot. First, the time taken for cluster computation increases linearly with the linear increase in the traffic density showing that the time complexity of the clustering approach implemented is linear. Second, the time taken to compute the clusters is still in microseconds even at the traffic density level of 200 vehicles. The error bars in the plot shows the 95% confidence interval calculated over 400 samples.

4.4.3 Data Collection At Different Traffic Densities

We conducted an experiment to find out how the total amount of data collected at micro clouds vary depending on the traffic densities and data sizes. In order to run this experiment, we chose four linearly increasing traffic densities 50, 100, 150 and

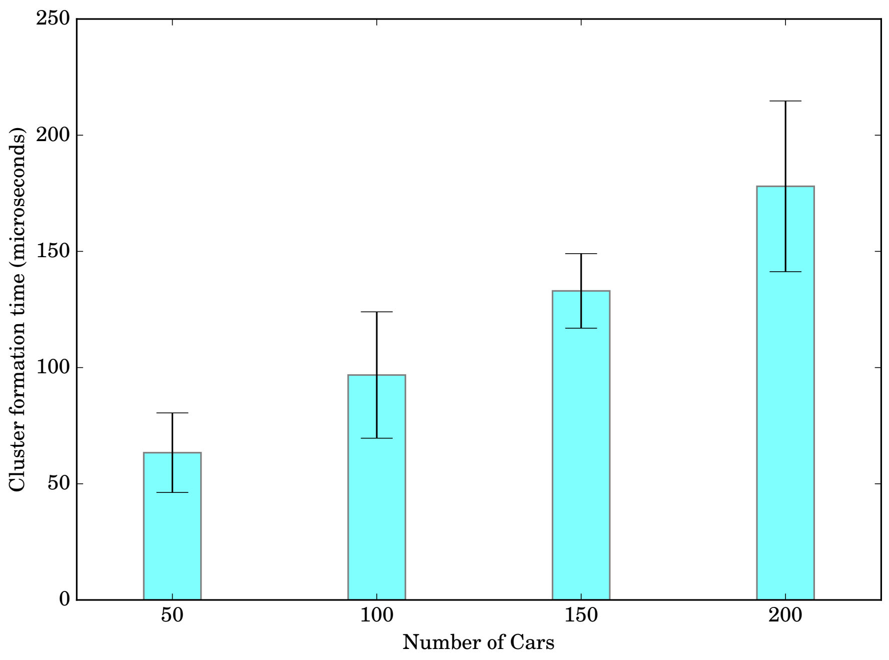


Figure 4.7 – The average cluster computation time increases linearly with the linear increase in traffic density. The error bars in the plot shows the 95% confidence interval calculated over 400 samples.

200. CM transfers 5kB of data every 2s to CH and CH transfers the collected data to AP without aggregation every 4s. The experiment was repeated with data size 10kB.

During the experiment, the total amount of data collected at the micro cloud was logged for each traffic density at AP. The plot shown in Figure 4.8 shows that the data collected at different traffic densities increases almost linearly with linear increase in traffic densities in both 5kB and 10kB data size experiments. Also, we can observe from the plot that, when the same mobility trace is used, the data collected at AP with 10kB data size is approximately twice in comparison to 5kB data size. The error bars in the plot shows the 95% confidence interval calculated over 3 runs. This evaluates the functionality of the data collection micro cloud service.

4.4.4 Data Aggregation

We conducted an experiment to evaluate the functionality of data aggregation at CH. For this, we chose a traffic density of 150 vehicle nodes in the scenario and aggregation percentages 25, 50, 75 and 100. The aggregation percentage 25

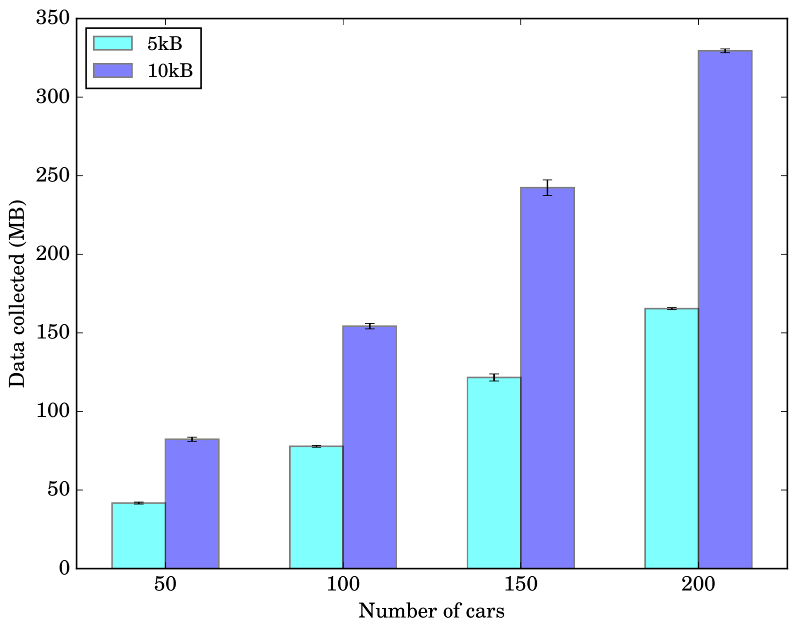


Figure 4.8 – The total amount of data collected at micro cloud increases in linear with the linear increase in traffic densities in both 5kB and 10kB data size experiments. Also, when the same mobility trace is used, the data collected at AP with 10kB data size is approximately twice in comparison to 5kB data size. The error bars in the plot shows the 95% confidence interval calculated over 3 runs.

indicates that the 25 percent of the data collected at CH is uploaded to AP and aggregation percentage 100 indicates that no aggregation has been performed at CH, in other words, all the data collected at CH is uploaded to AP. The experiment was conducted at two data sizes 5kB and 10kB.

The purpose of this experiment is to understand how the total amount of data collected at AP varies as the aggregation percentage adapted at CH varies. The plot in Figure 4.9 shows that when same mobility trace is used the amount of data collected at AP with aggregation percentage 50 is approximately double the amount of data collected with aggregation percentage 25 which was expected. The same pattern is observed for aggregation percentages 50 and 100. Also, the bars pattern shows that the amount of data collected increases proportionately with a proportionate increase in the aggregation percentage. The error bars in the plot shows the 95% confidence interval calculated over 3 runs. This evaluates the data aggregation service in the implementation.

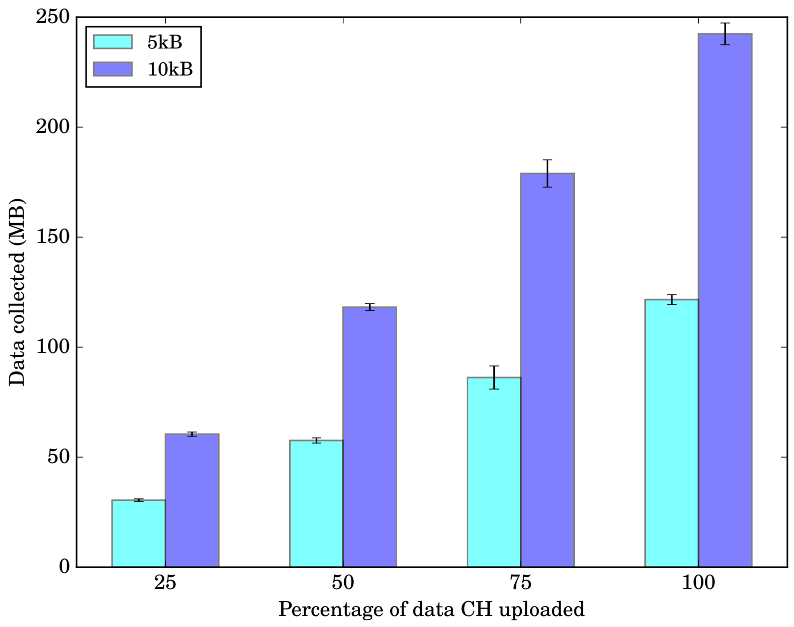


Figure 4.9 – The total amount of data collected at AP increases proportionately with the increase in the aggregation percentage (for example the aggregation percentage 25 indicates that the 25 percent of the data collected at CH is uploaded to AP). The error bars in the plot shows the 95% confidence interval calculated over 3 runs.

4.4.5 Task Distribution

We conducted an experiment to evaluate the task distribution application. The experiment was conducted at different traffic densities 50, 100, 150 and 200 vehicles. In this experiment, CH assigns subtasks to CMs periodically at an interval of 3s. The plot in Figure 4.10 shows the total number of tasks completed by all cluster heads in the micro cloud in the duration of 2000s. From the results, we can observe that, the total number of tasks completed by CHs is decreasing as the traffic density increases. This is expected because from the results in Section 4.4.1 we know that the consecutive times the vehicles elected as CH decreases as the traffic density increases.

Hence, if the vehicles do not elect as CH consecutively, at every interval only one task will be completed by CH as by next interval a different vehicle is elected as CH. For example - if the cluster computation interval is 5s, after two cluster computation intervals, if the same vehicle was elected as CH at both the intervals it had chance to complete 3 tasks (as CH assign subtasks to its CMs every 3s). But, if two different vehicles were elected as CH, each CH had chance to complete only one subtask - in total 2.

We also observe from the results that, at higher traffic densities, the difference in the total number of tasks completed by CHs is negligible. This is because at higher traffic densities, at every interval the CH changes. The error bars in the plot shows the 95% confidence interval calculated over 3 runs.

4.4.6 System Resource Usage

We conducted an experiment, to measure the usage of system resources such as memory and CPU. To do so, we chose to run all the vehicle nodes in 1 physical machine with system configuration as shown in Table 4.3. The data size used is 5kB. In this experiment the scenario size was increased to 1390 m^2 (the road length and width remained the same) for all the traffic densities - 100, 200 and 500 vehicle nodes. The scenario size was increased because the smaller scenario (used for smaller traffic densities) seemed unrealistic for 500 vehicle nodes.

During the experiment, the CPU utilization and memory consumption were captured using "top" command. The plot in Figure 4.11 shows the average CPU utilization percentage for different traffic densities. Though there is a slight increase in the CPU utilization percentage as the traffic density increases, we can observe from the plot that there is more room for higher traffic densities in the same machine. The error bars in the plot shows the 95% confidence interval calculated over 1300 samples (sampled at every second between 501s to 1800s).

Similarly, from the memory usage plot shown in Figure 4.12 we can observe that, as the traffic density increases memory consumption also increases. Also, we can see

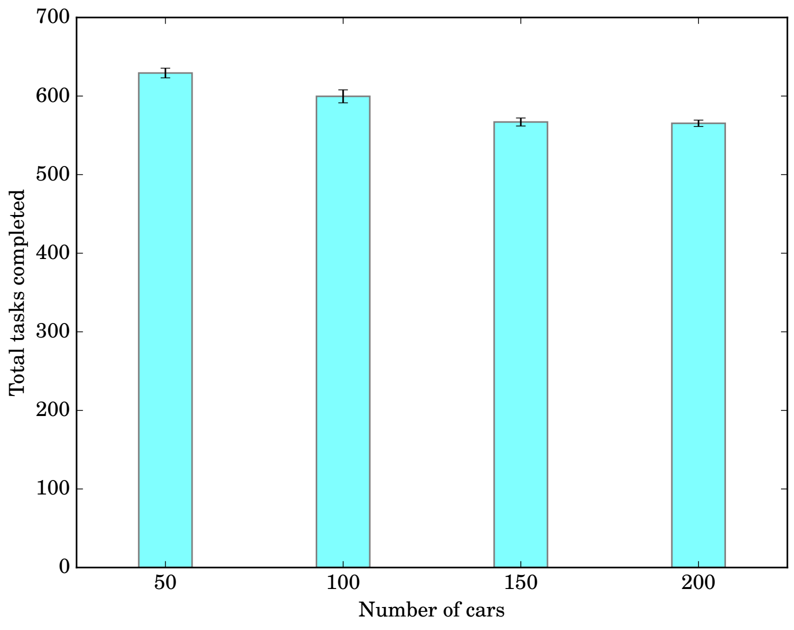


Figure 4.10 – The total number of tasks completed by all CHs at different traffic densities. As the traffic density increases the number of tasks completed decreases because as the traffic density increases the consecutive times the vehicles elected as CH decreases. The error bars in the plot shows the 95% confidence interval calculated over 3 runs.

properties	value
Architecture	x86_64
CPU(s)	8
Model name	Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz
Main Memory	16GB
Swap Memory	0K

Table 4.3 – The details of the machine where the vehicle nodes were run to conduct system resources usage measurement experiment.

from the plot that the memory used by 100, 200, 500 vehicle nodes is approximately 700MB, 1400MB and 3490MB. This underlines that the memory usage increases linearly with the linear increase in traffic density and one vehicle node consumes approximately 7 MB. The error bars in the plot shows the 95% confidence interval calculated over 1300 samples (sampled at every second between 501s to 1800s).

In this experiment, we tried to run 500 vehicle nodes in one machine. Initially, with the system default settings, approximately 155 to 160 vehicle nodes were able to run in one machine. This is because, by default, the system parameter `pids.max` was set to 12288 in the file `"/sys/fs/cgroup/pids/user.slice/user-1067.slice/pids.max"`. That means, one user can only run 12288 tasks or threads per machine. The implementation requires 67 threads per vehicle node. This includes the threads defined for different tasks within a node and the threads spawned by all ZeroMQ connections in a node. Each ZeroMQ connection spawns 2 threads in the background. Thus, 160 vehicles occupy approximately 11000 threads (remaining threads used and reserved by the system).

If more than the allowed number of threads were spawned, the system would print error message "Resource temporarily unavailable" and threads failed to get spawned. Later, when the parameter was increased to a sufficiently large num-

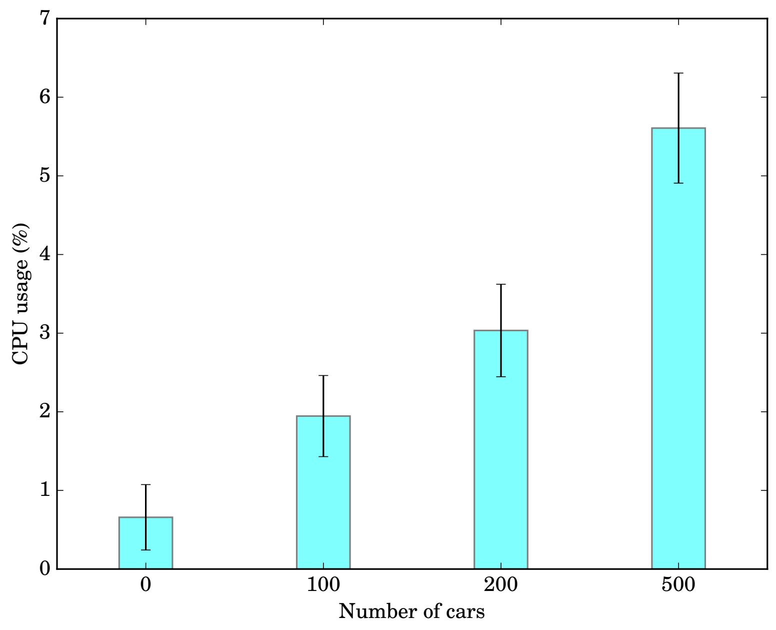


Figure 4.11 – The CPU usage at different traffic densities run in one machine. The error bars in the plot shows the 95% confidence interval calculated over 1300 samples (sampled at every second between 501s to 1800s).

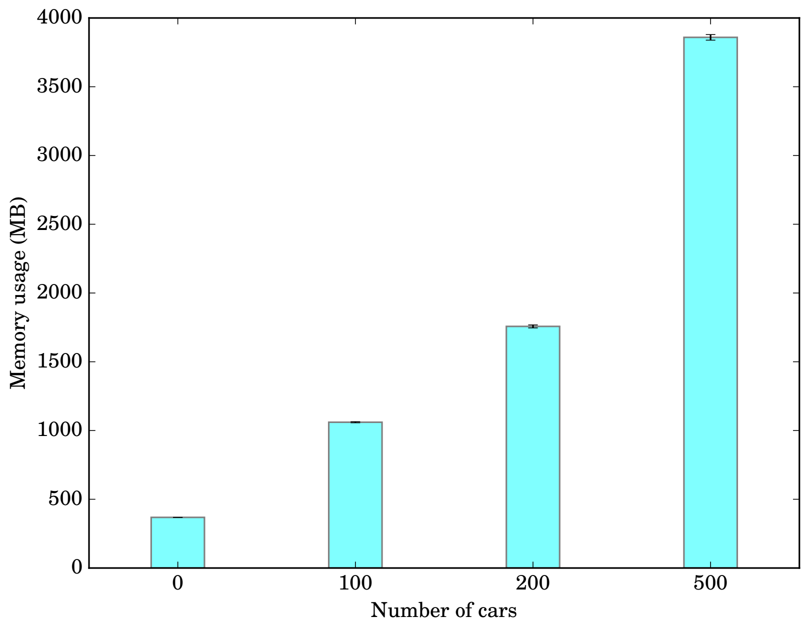


Figure 4.12 – The Memory usage at different traffic densities run in one machine. Memory usage increases linearly with the linear increase in traffic density. The error bars in the plot shows the 95% confidence interval calculated over 1300 samples (sampled at every second between 501s to 1800s).

ber 128000, the number of vehicle nodes allowed increased to 435 but not 500. This is because increase in the parameter `pids.max` (per user) was constrained by the total number of pids allowed in the system. Then, the system parameter "`kernel.pids.max=128000`" was added to file "`/etc/sysctl.conf`", after which we could run 500 nodes successfully in the machine.

In the other side, the number of nodes in a machine is also constrained by the number of ports available in the machine for use. Each vehicle node requires 5 distinct ports for ZeroMQ communications. The ports from 32768 to 61000 in the system are used as ephemeral ports in Linux. Hence, these ports are unavailable for ZeroMQ communications. In the experiments, we have used port numbers 15000 and above. In the implementation, once the vehicle exits from the network, there is no mechanism of reusing the same ports used by that vehicle in the same run. Hence, if we start our experiments with starting port 15000, theoretically we can have 3500 new vehicle nodes in one run. We can increase this number if we run our experiments with starting port number 10000 or less, but we observed in the interval 10000 to 15000 some ports would be already used by the system.

Chapter 5

Conclusion

The necessity of the vehicular micro clouds arises from the fact that as the traffic density increases, without the micro clouds the vehicles share the available cellular bandwidth to communicate with the data server. This increases the communication latency. In this thesis, we have implemented the vehicular micro cloud prototype and evaluated the implementation in Manhattan grid scenario by conducting experiments at different traffic densities and data sizes.

We validated the cluster computation in the implementation using cluster size as a metric in one and three micro clouds scenarios. Then, we conducted an experiment to evaluate average cluster formation time at different traffic densities. The results showed that the cluster formation time increases linearly with the linear increase in traffic density. Later, we verified the data collection and aggregation micro cloud service by comparing the expected amount of data collected at micro cloud with the amount of data collected at micro cloud from the experiment. The results showed a slight difference between the expected and the experimental result which is due to a possibility of CH or CM going out of communication range within an interval. Further, we evaluated data collection and aggregation service at different traffic densities and data sizes and presented the results. Additionally, we also evaluated task distribution service at different traffic densities. At last, we evaluated the system resources usage - the CPU and memory consumption by running all vehicle nodes in one machine. The results showed that one virtual machine could accommodate more than 500 vehicle nodes from the CPU and memory perspective.

As future work, the prototype can be improvised by incorporating a channel modelling algorithm to study realistic results. The prototype could be integrated into [14] for conducting experiments based on vehicular micro cloud field tests. Also, there is a scope for improvement in terms of visualization for better user experience if the support is extended to the Unity development environment.

List of Abbreviations

AP	Access Point
CH	Cluster Head
CM	Cluster Member
MEC	Mobile Edge Computing
RSU	Roadside Unit
UDM	Unit Disk Model
VMC	Vehicular Micro Cloud

List of Figures

List of Tables

Bibliography

[1] D. Jiang and L. Delgrossi, "IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments," in *67th IEEE Vehicular Technology Conference (VTC2008-Spring)*, Marina Bay, Singapore, May 2008, pp. 2036–2040. DOI: 10.1109/VETECS.2008.458.

[2] C. Hoymann, D. Astely, M. Stattin, G. Wikström, J. (Cheng, A. Höglund, M. Frenne, R. Blasco, J. Huschke, and F. Gunnarsson, "LTE release 14 outlook," *IEEE Communications Magazine*, vol. 54, no. 6, pp. 44–49, Jun. 2016. DOI: 10.1109/MCOM.2016.7497765.

[3] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil, "Vehicular Networking: A Survey and Tutorial on Requirements, Architectures, Challenges, Standards and Solutions," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 4, pp. 584–616, Nov. 2011. DOI: 10.1109/SURV.2011.061411.00019.

[4] M. Otomo, K. Hashimoto, N. Uchida, and Y. Shibata, "Mobile cloud computing usage for onboard vehicle servers in collecting disaster data information," in *2017 IEEE 8th International Conference on Awareness Science and Technology (iCAST)*, Taichung, Taiwan, Nov. 2017, pp. 475–480. DOI: 10.1109/ICAwST.2017.8256504.

[5] E. Uhlemann, "Introducing Connected Vehicles [Connected Vehicles]," *IEEE Vehicular Technology Magazine*, vol. 10, no. 1, pp. 23–31, Mar. 2015. DOI: 10.1109/MVT.2015.2390920.

[6] J. E. Siegel, D. C. Erb, and S. E. Sarma, "A Survey of the Connected Vehicle Landscape—Architectures, Enabling Technologies, Applications, and Development Areas," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, Aug. 2018. DOI: 10.1109/TITS.2017.2749459.

[7] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1628–1656, Mar. 2017. DOI: 10.1109/COMST.2017.2682318.

[8] F. Hagenauer, C. Sommer, T. Higuchi, O. Altintas, and F. Dressler, "Vehicular Micro Clouds as Virtual Edge Servers for Efficient Data Collection," in *23rd ACM International Conference on Mobile Computing and Networking (MobiCom 2017), 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services (CarSys 2017)*, Snowbird, UT: ACM, Oct. 2017, pp. 31–35. DOI: 10.1145/3131944.3133937.

[9] M. Gerla, "Vehicular Cloud Computing," in *11th IFIP/IEEE Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2012)*, Ayia Napa, Cyprus: IEEE, Jun. 2012, pp. 152–155. DOI: 10.1109/MedHocNet.2012.6257116.

[10] T. Higuchi, J. Joy, F. Dressler, M. Gerla, and O. Altintas, "On the Feasibility of Vehicular Micro Clouds," in *9th IEEE Vehicular Networking Conference (VNC 2017)*, Torino, Italy: IEEE, Nov. 2017, pp. 179–182. DOI: 10.1109/VNC.2017.8275621.

[11] F. Kordon and Luqi, "An introduction to rapid system prototyping," *IEEE Transactions on Software Engineering*, vol. 28, no. 9, pp. 817–821, Sep. 2002. DOI: 10.1109/TSE.2002.1033222.

[12] Z. Meng, Z. Wu, C. Muvianto, and J. Gray, "A Data-Oriented M2M Messaging Mechanism for Industrial IoT Applications," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 236–246, Feb. 2017. DOI: 10.1109/JIOT.2016.2646375.

[13] S. Gougeaud, S. Zertal, J.-C. Lafoucriere, and P. Deniel, "Using ZeroMQ as communication/synchronization mechanisms for IO requests simulation," in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, Seattle, WA, USA: IEEE, Jul. 2017, pp. 90–97. DOI: 10.23919/SPECTS.2017.8046773.

[14] S. Laux, G. S. Pannu, S. Schneider, J. Tiemann, F. Klingler, C. Sommer, and F. Dressler, "OpenC2X - An Open Source Experimental and Prototyping Platform Supporting ETSI ITS-G5," in *8th IEEE Vehicular Networking Conference (VNC 2016), Demo Session*, Columbus, OH: IEEE, Dec. 2016, pp. 152–153. DOI: 10.1109/VNC.2016.7835955.

[15] H. Jemal, Z. Kechaou, M. B. Ayed, and A. M. Alimi, "Cloud computing and mobile devices based system for healthcare application," in *2015 IEEE International Symposium on Technology and Society (ISTAS)*, Dublin, Ireland: IEEE, Nov. 2015, pp. 1–5. DOI: 10.1109/ISTAS.2015.7439407.

[16] P. Thornton and C. Houser, "Using mobile phones in education," in *The 2nd IEEE International Workshop on Wireless and Mobile Technologies in Education, 2004*, JungLi, Taiwan, Taiwan: IEEE, Mar. 2004, pp. 3–10. DOI: 10.1109/WMTE.2004.1281326.

[17] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, and A. Qureshi, “Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions,” *Journal of Network and Computer Applications*, vol. 48, p. 19, Feb. 2015. DOI: `https://doi.org/10.1016/j.jnca.2014.09.009`.

[18] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog Computing and Its Role in the Internet of Things,” in *1st Workshop on Mobile Cloud Computing (MCC 2012)*, Helsinki, Finland: ACM, Aug. 2012, pp. 13–16. DOI: `10.1145/2342509.2342513`.

[19] A. Ahmed and E. Ahmed, “A survey on mobile edge computing,” in *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, Coimbatore, India: IEEE, Jan. 2016, pp. 1–8. DOI: `10.1109/ISCO.2016.7727082`.

[20] S. Olariu, T. Hristov, and G. Yan, “The Next Paradigm Shift: From Vehicular Networks to Vehicular Clouds,” in *Mobile Ad Hoc Networking*, Wiley, 2013, pp. 645–700. DOI: `10.1002/9781118511305.ch19`.

[21] M. Abuelela and S. Olariu, “Taking VANET to the Clouds,” in *The 8th International Conference on Advances in Mobile Computing & Multimedia*, Paris: ACM, Nov. 2010, pp. 6–13. DOI: `10.1145/1971519.1971522`.

[22] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, “Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds,” in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, Seoul, South Korea: IEEE, Mar. 2014, pp. 241–246. DOI: `10.1109/WF-IoT.2014.6803166`.

[23] M. Eltoweissy, S. Olariu, and M. Younis, “Towards Autonomous Vehicular Clouds,” in *2nd International ICST Conference on Ad Hoc Networks (ADHOC-NETS 2010)*, J. Zheng, D. Simplot-Ryl, and V. C. M. Leung, Eds., Victoria, BC, Canada: Springer, Aug. 2010, pp. 1–16. DOI: `10.1007/978-3-642-17994-5_1`.

[24] L. Gu, D. Zeng, and S. Guo, “Vehicular Cloud Computing: A Survey,” in *IEEE Global Telecommunications Conference (GLOBECOM 2013), Workshop on Cloud Computing Systems, Networks, and Applications*, Atlanta, GA, USA: IEEE, Dec. 2013. DOI: `10.1109/GLOCOMW.2013.6825021`.

[25] E. Lee, E.-K. Lee, M. Gerla, and S. Oh, “Vehicular Cloud Networking: Architecture and Design Principles,” *IEEE Communications Magazine*, vol. 52, no. 2, pp. 148–155, Feb. 2014. DOI: `10.1109/MCOM.2014.6736756`.

[26] F. Ahmad, M. Kazim, A. Adnane, and A. Awad, “Vehicular Cloud Networks: Architecture, Applications and Security Issues,” in *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, Limassol, Cyprus: IEEE, Dec. 2015, pp. 571–576. DOI: `10.1109/UCC.2015.101`.

[27] F. Hagenauer, F. Dressler, and C. Sommer, “A Simulator for Heterogeneous Vehicular Networks,” in *6th IEEE Vehicular Networking Conference (VNC 2014), Poster Session*, Paderborn, Germany: IEEE, Dec. 2014, pp. 185–186. DOI: 10.1109/VNC.2014.7013339.

[28] L. Codeca, R. Frank, and T. Engel, “Luxembourg SUMO Traffic (LuST) Scenario: 24 Hours of Mobility for Vehicular Networking Research,” in *7th IEEE Vehicular Networking Conference (VNC 2015)*, Kyoto, Japan: IEEE, Dec. 2015. DOI: 10.1109/VNC.2015.7385539.

[29] C. Cooper, D. Franklin, M. Ros, F. Safaei, and M. Abolhasan, “A Comparative Survey of VANET Clustering Techniques,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 657–681, Feb. 2017. DOI: 10.1109/COMST.2016.2611524.

[30] E. Souza, I. Nikolaidis, and P. Gburzynski, “A New Aggregate Local Mobility (ALM) Clustering Algorithm for VANETs,” in *12th IEEE International Conference on Communication Technology (ICCT 2010)*, Cape Town, South Africa: IEEE, May 2010. DOI: 10.1109/ICC.2010.5501789.

[31] Z. Wang, L. Liu, M. Zhou, and N. Ansari, “A Position-Based Clustering Technique for Ad Hoc Intervehicle Communication,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 38, no. 2, p. 8, Mar. 2008. DOI: 10.1109/TSMCC.2007.913917.

[32] M. M. Caballeros Morales, C. S. Hong, and Y.-C. Bang, “An Adaptable Mobility-Aware Clustering Algorithm in vehicular networks,” in *13th Asia-Pacific Network Operations and Management Symposium (APNOMS 2011)*, Taipei, Taiwan: IEEE, Sep. 2011. DOI: 10.1109/APNOMS.2011.6077004.

[33] T. Higuchi, F. Dressler, and O. Altintas, “How to Keep a Vehicular Micro Cloud Intact,” in *87th IEEE Vehicular Technology Conference (VTC2018-Spring)*, Porto, Portugal: IEEE, Jun. 2018. DOI: 10.1109/VTCSpring.2018.8417759.

[34] D. Kwak, R. Liu, D. Kim, B. Nath, and L. Iftode, “Seeing Is Believing: Sharing Real-Time Visual Traffic Information via Vehicular Clouds,” *IEEE Access*, vol. 4, p. 15, May 2016. DOI: 10.1109/ACCESS.2016.2569585.

[35] T. Häberle, L. Charissis, C. Fehling, J. Nahm, and F. Leymann, “The Connected Car in the Cloud: A Platform for Prototyping Telematics Services,” *IEEE Software*, vol. 32, no. 6, p. 8, Nov. 2015. DOI: 10.1109/MS.2015.137.

Appendices

Appendix A

Sequence Diagrams

This chapter gives a brief idea about how the messages flow between the different nodes or the modules using sequence diagrams.

A.1 Vehicle Local Info

The sequence diagram shown in Figure A.1 briefs about how vehicles fetch their updated GPS and Speed information. This communication requires ZeroMQ REQ_-REP socket pair. The vehicles periodically send the request through their REQ socket for their local information. The In-car sensors have its REP socket always waiting for the vehicle's requests and reply them with the updated values whenever the request comes. By default, vehicles send the request to in-car sensors every one second.

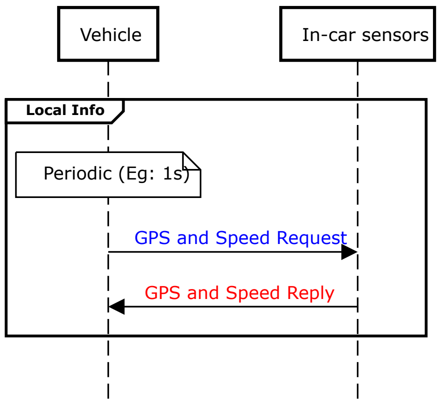


Figure A.1 – In-car sensors fetch GPS and Speed data from Mobility Trace. The updated information is sent to vehicle on request.

A.2 Clustering Info

The sequence diagram in Figure A.2 briefs about how the clustering service messages exchange happens in the system. When a vehicle wants to send its local information message to AP, it pushes message to Unit Disk Model (UDM) and UDM publishes the message to AP. When AP wants to send computed cluster information message to a vehicle, it pushes message to UDM and UDM publishes the message to the vehicle.

A.3 Data Application Info

The sequence diagram shown in the Figure A.4 briefs about how the data packets flow in the system. Here we see two message sequences - one, periodically CM pushes data message to UDM and UDM publishes it to CH and two, periodically CH pushes aggregated data message to UDM and UDM publishes it to AP.

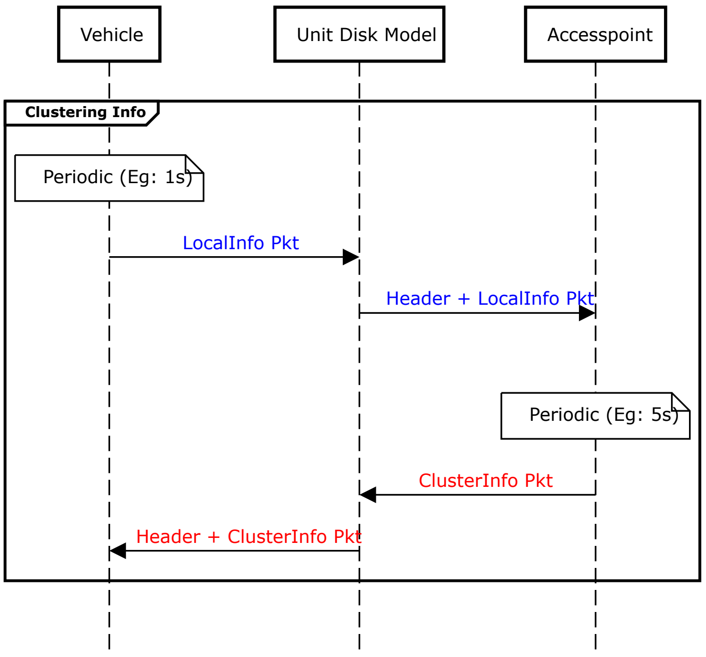


Figure A.2 – Each vehicle periodically sends local info (GPS and speed) to AP. AP periodically computes clusters and send it to Cluster Head and Cluster Member(s).

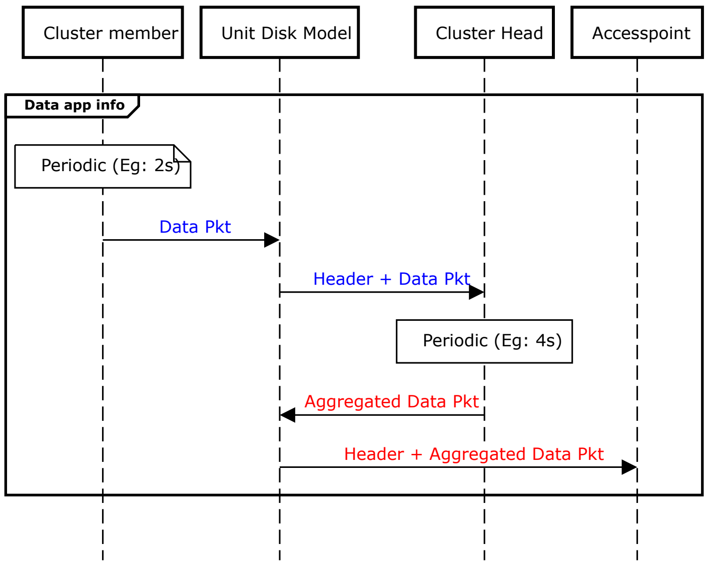


Figure A.3 – Each CM periodically sends data to CH. CH periodically aggregates the data and send it to AP.

A.4 Task Application Info

The sequence diagram shown in the **??** briefs about how the task distribution application packets flow in the system. Here we see two message sequences - one, periodically CH pushes subtask assignment message to UDM and UDM publishes it to CM and two, CM pushes subtask result message to UDM and UDM publishes it to CH.

A.5 Visual Component Info

The sequence diagram as shown in Figure A.5 briefs about the modules that provide their information to the visual component module. Here two different modules send messages to visual component - one, periodically AP publishes the cluster status message to visual component and two, vehicles periodically pushes application info message to visual component.

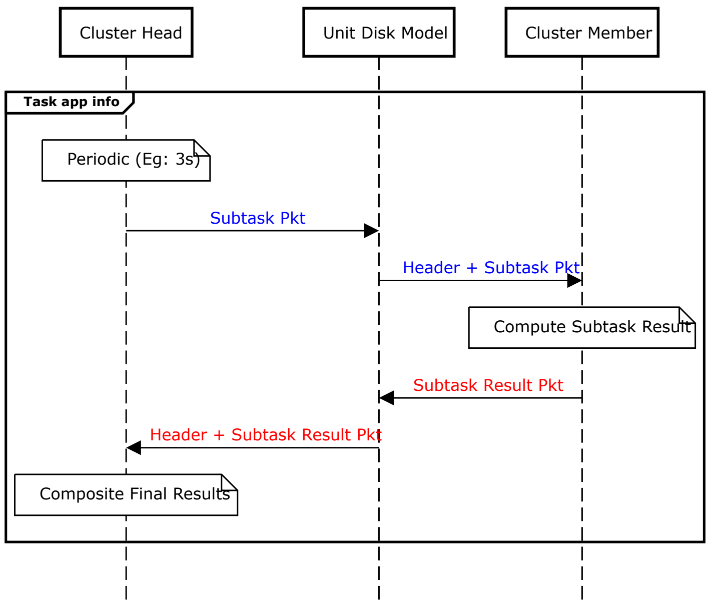


Figure A.4 – Each CH periodically sends data to CM. CH periodically aggregates the data and send it to AP.

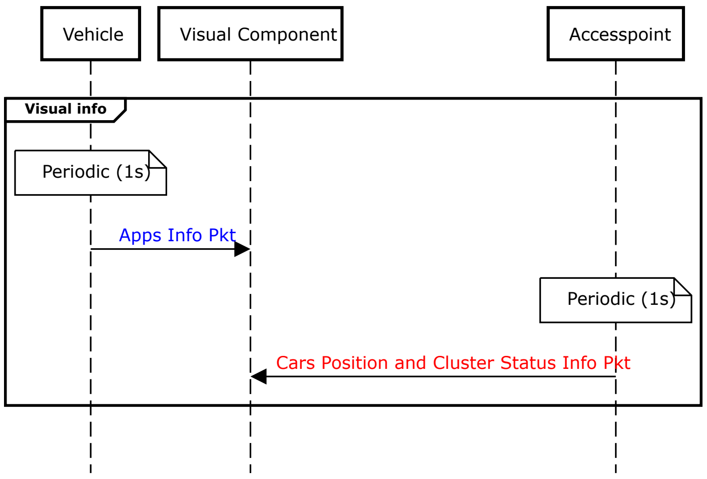


Figure A.5 – Each Car periodically sends application related information (Ex: Total data sent to AP). AP sends Cars position and cluster status information to visual component.