NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTEMENT OF INFORMATICS AND TELECOMMUNICATIONS

POSTGRADUATE PROGRAM

THESIS

Clustering algorithms in Vehicular Ad-hoc Networks: Design and Performance Evaluation

Lampros P. Katsikas

Supervisors: **Athanasia Alonistioti,** Assistant Professor

ATHENS

JUNE 2015

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και αξιολόγηση συστημάτων συσταδοποίησης κόμβων σε ασύρματα περιβάλλοντα μετάδοσης

Λάμπρος Π. Κατσίκας

Επιβλέποντες: **Αθανασία Αλωνιστιώτη,** Επίκουρος Καθηγητής

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2015

THESIS

Clustering algorithms in Vehicular Ad-hoc Networks: Design and Performance Evaluation

Lampros P. Katsikas

R.N.: M1232

SUPERVISORS: Athanasia Alonistioti, Assistant Professor

ADVISORY COMMITTEE: Dimitrios Gizopoulos, Associate Professor

June 2015

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και αξιολόγηση συστημάτων συσταδοποίησης κόμβων σε ασύρματα περιβάλλοντα μετάδοσης

Λάμπρος Π. Κατσίκας
Α.Μ.: Μ1232

ΕΠΙΒΛΕΠΟΝΤΕΣ: **Αθανασία Αλωνιστιώτη,** Επίκουρος Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Δημήτριος Γκιζόπουλος,** Αναπληρωτής Καθηγητής

Ιούνιος 2015

ABSTRACT

Vehicles communication is a new type of communication that arises the last years. The communication overhead that could be generated due to the fast mobility changes could lead to communication failures and clustering is a promising technique to solve this kind of problems by providing a multilevel hierarchy structure. We provide an implementation of 3 clustering algorithms in NS-3 simulation environment. The evaluation of the clustering algorithms is done by testing the set of algorithms in realistic urban and rural topology scenarios build with SUMO traffic simulation using real maps. The results show the existence of both simple clustering schemes that could lead to fast formation of the clusters but in pour efficiency in cluster stability and of more costly schemes in terms of messages exchanged that provide higher cluster stability.

SUBJECT AREA: Vehicle Ad-hoc NETworks (VANETs)

KEYWORDS: VANETs, Clustering Algorithm, NS-3 Simulator, SUMO Traffic Simulator. Cluster Head

ΠΕΡΙΛΗΨΗ

Η επικοινωνία μεταξύ των αυτοκινήτων είναι μια νέου τύπου επικοινωνία που έχει αναδυθεί τα τελευταία χρόνια. Ο μεγάλος φόρτος δεδομένων επικοινωνίας που μπορεί να δημιουργηθεί λόγω της αυξημένης κινητικότητας των στοιχείων ενός τέτοιου τυπου δικτύου μπορεί να αντιμετωπιστεί με την συσταδοποίηση, μία πολλά υποσχόμενη τεχνική για την επίλυση τέτοιου είδους προβλημάτων παρέχοντας μια πολυεπίπεδη δομή ιεραρχίας. Στα πλαίσια της παρούσας διπλωματικής εργασίας υλοποιήθηκαν 3 αλγόριθμοι συσταδοποίησης οχημάτων στον προσομοιωτή NS-3. Η αξιολόγηση των αλγορίθμων που υλοπιήθηκαν έγινε σε ρεαλιστικά αστικά και αγροτικά περιβάλλοντα που κατασκευάστηκαν με την βοήθεια του προσομοιωτή κίνησης SUMO και με τη χρήση πραγματικών χαρτών. Τα αποτελέσματα δείχνουν αφενός την ύπαρξη απλών αλγορίθμων που οδηγούν στη γρήγορη δημιουργία συστάδων αυτοκινήτων αλλα ταυτόχρονα στην έλλειψη σταθερότητας ως προς την δομή αυτών των συστάδων. Αφετέρου, δείχνουν την ύπαρξη πιο πολύπλοκων και πιο δαπανηρών αλγορίθμων απο πλευράς μηνυμάτων που ανταλλάσσονται, που όμως οδηγούν σε πιο σταθερές συστάδες αυτοκινήτων.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Ασύρματα δίκτυα οχημάτων τύπου Ad-hoc

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ασύρματα δίκτυα οχημάτων τύπου Ad-hoc, Αλγόριθμος Συσταδοποίησης, Προσωμοιωτής NS-3, Προσωμοιοτής Κίνησης SUMO, Επικεφαλής Ομάδας

ACKNOWLEDGEMENTS

I would like to especially thank my supervisor Professor Nancy Alonistioti for providing me the means and the support to perform this study.

I would also like to express my kindest greetings to Dr. Apostolos Kousaridas for the excellent cooperation we have had during the last year in order to reach this important milestone.

Finally, I would like to thank the Ph.D. student Konstantinos Chatzikokolakis for his contribution-as a co-author-to the paper submitted to the Workshop on ns-3, Barcelona, Spain, May 2015.

LIST OF PUBLICATIONS

[1].L.Katsikas, K. Chatzikokolakis, N. Alonistioti, “Implementing clustering for vehicular ad-hoc networks in ns-3”. In the Proceedings of the Workshop on ns-3, Barcelona, Spain, May 2015.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

The current thesis is designed and implemented by M.Sc. student Lampros P. Katsikas member of the Self-evolving Cognitive and Autonomic Networking (SCAN) Lab of the department of Informatics and Telecommunications, National and Kapodistrian University of Athens during 2015. The supervisor of this thesis is Professor Nancy Alonistioti. Important contribution to this work is being done by Dr. Apostolos Kousaridas.

1. INTRODUCTION

Recent advances in hardware, software, and communication technologies have enabled the design and implementation of a variety of networks. One such network that has become of great interest in the last years is the Vehicular Ad-Hoc Network (VANET). VANETs have been created by applying basic principles of Mobile Ad-hoc Networks (MANETs) in vehicles domain and has become an active area of research and standardization showing great potentials to improve vehicular communication in terms of road safety and traffic efficiency.

Many studies have shown that the number of wireless devices that are part of or reside in a vehicle is ever increasing. Such devices are wireless sensors, GPS, smart phones, laptops, tablets, etc. Moreover, rapid changes in a VANET topology due to high mobility introduce high communication overhead in order to exchange the updated topology information. Thus, a large number of devices may attempt to cooperate in the Dedicated Short Range Communications (DSRC) band leading to communication failure due to congestion. Clustering schemes could be a solution to this type of problems by reducing data volumes exchanged and by partitioning the network into sub-networks according to an appropriate metric value.

Current study firstly aims to present clustering schemes that have been proposed in the literature in order to overcome the problem of the ever increasing number of devices trying to cooperate in VANET environment. Clustering schemes propose methods for the grouping of the vehicles into a hierarchical structure instead of a flat topology. In the scope of the current study we provide in the community both an open source implementation of three state of the art clustering algorithms in the area of VANETs using NS-3 simulator and an extensive evaluation of those clustering schemes.

The remainder of this document is organized as follows: Section 2 introduces background information for this type of networks and related work. In section 3 we present the three state of the art algorithms we have chosen for implementation. Simulation environment and implementation details are described in section 4 while, section 5 describes the configuration of the simulation scenarios. The evaluation of the simulation scenarios is provided in section 6 and finally in section 7 are the conclusions of this work.

An NS-3 instance (version 3.21) containing the code implementation of this study is available in a public git repository at https://git.scanlab.gr/lkatsikas/wns3-2015.git.

2. BACKGROUND WORK

In the first part of this section we provide the most important characteristics of VANETs and the basic principles of the applications used in such environments, while the second subsection presents VANET clustering schemes from the literature.

2.1 Internet Of Vehicles

Key features of VANETs are summarized below:

- VANETs are considered as networks of high mobility. Thus a vehicular communication network is characterized as a dynamic network where vehicles movement highly depends on the topology, road and traffic conditions.
- Unlike to other types of networks, VANETs have no power constraints since vehicles could provide endless power to the devices participating in those networks.
- Vehicle-to-Vehicle (V2V) communication networks are formed by moving vehicles, equipped with wireless interfaces that allow them to exchange communication messages. This type of communication is most suited for short-range communication networks.
- Vehicle-to-Infrastructure (V2I) communication networks make use of preexisting network infrastructure such as Road-Side Units (RSUs). Communication between vehicles and RSUs is supported by protocols best suited for longer communication range.

Test case scenarios cover a wide range in this type of networks including security and road conditions awareness. Specifically, VANET applications could be divided, mainly into 3 categories, according to [1]:

- Safety applications (e.g. Incident notification, road hazard notification) in which messages should be exchanged quickly and reliably and thus such applications are characterized as delay critical.
- Traffic management applications (e.g. collision avoidance, root trip management, etc.) which aim to reduce time and fuel consumption of the vehicles by changing routes.
- Infotainment applications (e.g. video games, emails), the so called value added services.

As stated in the application scenarios above, VANET applications require low latency especially for the safety application scenarios, high data rate requirements and consistency in high mobility environment. Wireless Access in Vehicular Environments (WAVE) protocol [2] is a first definition of the IEEE 802.11p protocol that aims to fulfill the needs for VANET communication in the future. According to the specification, communication takes place in the frequency range [5.850 – 5.925] GHz, the Dedicated Short Range Communications (DSRC) spectrum band in the United States. The band is of 75 MHz in total which is further divided in 7 channels, one central control channel (CCH) and six service channels (SCH) as depicted in Figure 2-1. WAVE protocol is a combination of other protocols of the 802.11 family. In the MAC protocol layer is used the Enhanced Distributed Channel Access (EDCA) protocol that is based on the IEEE 802.11e provided with some extra modifications. The physical layer of IEEE 802.11p protocol is Orthogonal Frequency Division Modulation (OFDM) based on IEEE 802.11a.

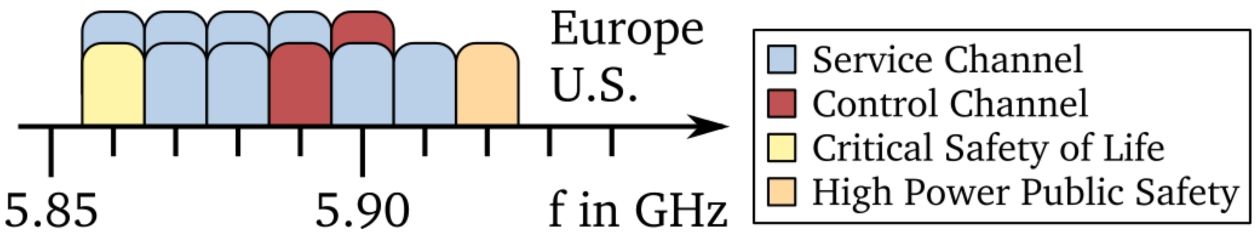


Figure 2-1: Channels in WAVE protocol

The main idea of the clustering is to group a subset of vehicles moving into the same area, having a very close relative position to each other and having similar mobility patterns. Clustering schemes take advantage of the fact that vehicles tend to move in an organized way rather than moving randomly in the area. Conceptually, clustering process assigns two distinct roles in the members of the group, the Cluster Head (CH) and the Cluster Member (CM). Each group has at least one CH that is responsible for a number of CMs and acts as the representative of the group. Those entities consist a cluster. Figure 2-2 shows a graphical representation of 2 clusters.

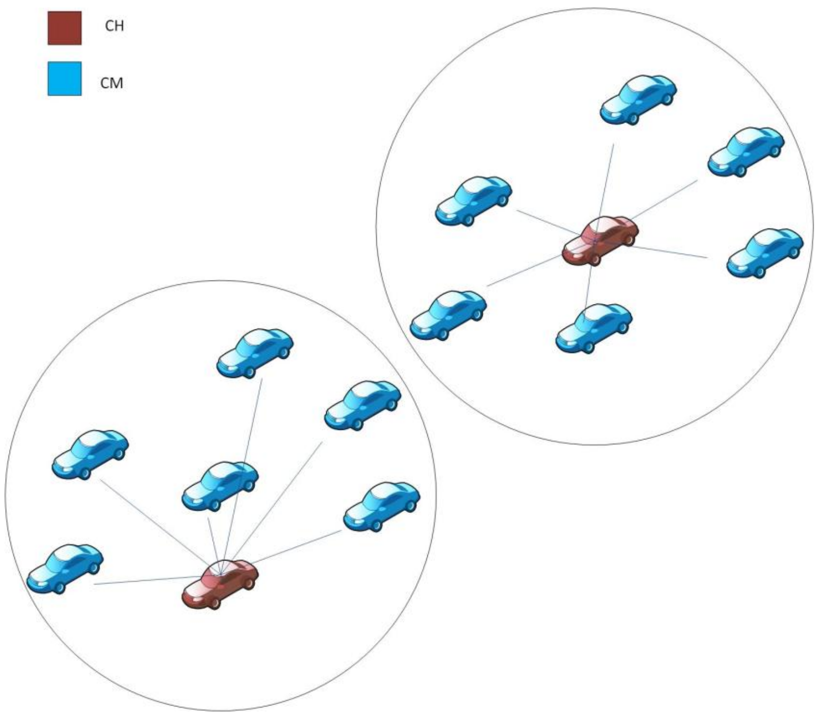


Figure 2-2: Clustering representation

2.2 VANET Clustering

In this section we provide the state of the art approaches of V2V Clustering Schemes that appear in the literature. In [3] a novel algorithm to form stable clusters in vehicular ad hoc networks on highways is introduced. The method is based on speed-overlapped clustering method for highways. Neighboring vehicles are defined as stable and unstable clustering neighbors based on their speed and relative movement direction and only stable neighbors can form clusters. This scheme results into vehicles moving in the fast speed lanes being grouped in different clusters than slow moving vehicles. In [4] a novel approach using Affinity Propagation algorithm as clustering criterion that minimizes both relative mobility and distance between Cluster Head (CH) and Cluster-Members (CMs) is introduced. The information exchange among neighboring nodes involves packets regarding two parameters: i) responsibility and ii) availability. Responsibility represents node’s suitability to become CH and availability indicates the desire of a node to become a CH. When Global Positioning Systems (GPS) measurements are available, stability is further increased based on moving direction of vehicles based on a similarity function that evaluates current and future vehicles’ positions. In [5] the authors propose an algorithm designed for aeronautical ad hoc networks, which, however, can also be applied in highway scenarios. Direction and velocity of nodes are being considered as inputs for the clustering algorithm. Their criterion for cluster formation is based on the time period nodes are expected to lie within communication range.

On the one hand, when nodes’ location is not available, they rely on the Doppler shift from control packets exchanged among nodes so as to calculate this time period. On the other hand, when nodes can obtain their current location, this time period is calculated based on Link Expiration Time (LET) [6]. Clusters consist of nodes that lie within communication range for at least a minimum threshold. The proposed solution in [7] focuses on long cluster lifetime where the participating nodes calculate their average velocity and acceleration and through message broadcasting all nodes receive the corresponding values of their neighbors. Based on the exchanged information, each node is able to calculate its Spatial Dependency [8] with each neighbor, its total Spatial Dependency regarding its entire neighborhood and its Cluster Relation with its neighbors. The last metric is used as criterion for the cluster formation. In [9] the vehicles are equipped with digital maps split into smaller regions and are able to obtain their direction which is then used as input in cluster formation. Vehicles with the same direction are grouped in the same cluster; however, clusters are recalculated at intersections due to vehicles moving towards different directions, while exiting an intersection. Packets/messages are successfully exchanged only among vehicles travelling to the same direction (i.e., vehicles within the same cluster). The Clustering Algorithm for VANETs introduced in [10] is based on Basagni’s Distributed and Mobility-Adaptive Clustering (DMAC) algorithm presented in [11]. In this VANET-oriented version, each vehicle calculates a weighted value indicating its suitability to become CH (higher value represents higher suitability). This value is calculated based on parameters such as position, velocity, and connectivity. In general, vehicles that become CH, retain this value high as clusters move and consequently re-clustering is avoided and clusters’ lifetime is extended. In [12], the communication among vehicles is unreliable in sparse networks and thus the algorithm’s behavior is affected. There are algorithms [13] that also consider the behavior of vehicles during message forwarding but such metric increases significantly the complexity of the system. In [14] overlapping clusters are created but this approach leads to unnecessary traffic exchanged. In [15], the authors propose a mechanism that focuses on fast cluster formation rather than focusing on clusters’ stability. In [16] the authors assume that the number of lanes of a road and the lane each vehicle resides is known, which however, is not always the case in real environments.

3. CLUSTERING ALGORITHMS

In VANETs, a clustering algorithm is mainly a distributed process controlled by the vehicles. Any clustering algorithm for VANETs has to solve 2 issues. The first one is the definition of a metric according to which, vehicles will form clusters (formation process) and the second is to specify the maintenance of the clustering structure since VANETs are dynamic networks characterized by rapid changes in position and velocity of the participating vehicles (maintenance process).

After an extensive search in the literature for the clustering schemes, introduced in section 2, we provide a deep analysis of 3 very promising clustering schemes that have been subject of numerous studies in the VANETs area so far.

3.1 A novel algorithm to form stable clusters in vehicular ad hoc networks on highways

The degree of the speed deviation among neighboring vehicles is the key criterion in order to construct stable clusters according to this algorithm [3]. Neighboring vehicles are characterized as stable or unstable neighbors based on their velocity vector (i.e. speed and direction) and only neighbors that are stable may form clusters together. Neighbors that are not moving into the same direction are ignored. In addition, any two vehicles are divided into r-Stable and 2r-Stable according to their distance factor *r* shown in Figure 3-1. This distance factor reflects the transmission range of the service channel, while *R* is the transmission range of the Control channel. The units for both factors are meters.

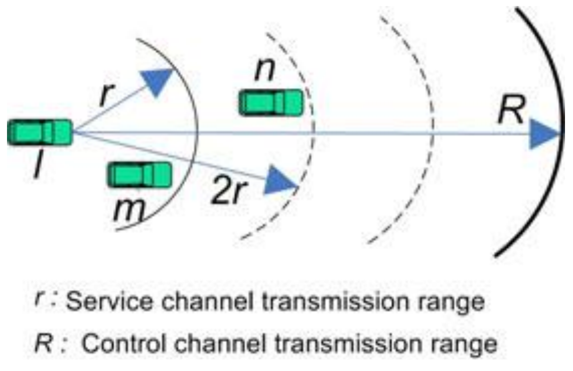

r
n
R
l
2r
m
r : Service channel transmission range
R : Control channel transmission range

Figure 3-1: Service channel VS Control channel

In more detail, vehicles broadcast their current mobility state using periodic messages. Neighbors with relative speed less than a predefined threshold could be considered as stable neighbors. The cluster formation process is initiated when the slowest vehicle sends a cluster-originating message. All 2r-Stable neighbors with greater speed reply to this message by changing their cluster identifier (ID) temporarily to the ID of the originating node and by calculating their suitability value to become CH. The suitability value of each vehicle is proportional to the deviation of its position and velocity compared to the mean values calculated by the vehicle’s stable neighbors. This value reflects the time that the vehicle has to wait before announcing its eligibility to become CH by sending a message to form the cluster. Vehicles that receive such message before their waiting time expires, quit the CH election competition, set their state to CM and change their temporary cluster ID to the new cluster ID encapsulated in the received message. The rest of the neighboring vehicles not being part of this process will go through the same cluster formation process to create other clusters.

Due to high mobility, communication messages in VANETs may lead to excessive signaling. It is of great importance to minimize the number of cluster changes by minimizing vehicle transitions among clusters thus minimizing the number of exchanged messages. Therefore, the stability of the cluster is considered as a key parameter in the algorithm evaluation. In this section we analyze the basic rules for the cluster maintenance according to the following events:

- Joining a cluster: This event is divided into two sub-events. If there is only one CH in the vicinity then a potential CH will accept the vehicle in case their speed deviation is lower than a predefined threshold *th*. This threshold is calculated as the mean value of the relative speed of all the stable neighbors of the vehicle. In case of more than one CH found in the vicinity area, the vehicle joins the cluster that remains for the longest time period. This is also called greatest remaining time (RT). This metric is calculated based on speed, direction and position information that nodes already have for their neighbors.
- Leaving a cluster: If neither a CH is in the vicinity, nor standalone vehicles to form new cluster, the vehicle changes its state to standalone and leaves the cluster.
- Cluster merging: In case that two CHs appear within each other’s range and their speed deviation is within a predefined threshold *th* then a cluster merge event is triggered. Specifically, the CH with fewer members gives up its CH role and becomes CM in the other cluster. Its CMs also join if they are within the new CH’s range, or they calculate the greatest RT before joining a cluster in case of more than one CHs existing in the vicinity. Otherwise, if there is no cluster within the range for some standalone vehicles, then those vehicles start a clustering process to form other clusters.

3.2 Modified DMAC Clustering Algorithm for VANETs

Modified Distributed Mobility Adaptive Clustering (MDMAC) algorithm has been proposed from Wolny [10] as an extension of the original Distributed Mobility Adaptive Clustering (DMAC) algorithm described from Basagni [11]. DMAC was initially proposed for Mobile Ad-hoc Networks (MANETs), while MDMAC applies the same idea into VANETs which is a different and completely dynamic type of network compared to MANETs. Both algorithms are characterized as generic frameworks for clustering since they are based on node’s weight for the cluster formation and node weight could be any metric that characterizes a node (i.e. number of neighbors, relevant position compared to the neighbors).

In MDMAC, any vehicle uses a periodic mechanism to send Hello messages to its neighbors reporting its current position, velocity, direction, its current status (if a CH or CM) and its current CH. If one of its neighbors is CH with greater node weight then the vehicle will join neighbor’s cluster. Otherwise, the vehicle will become CH itself. Moreover, when a message is received from a CH, instead of attaching directly to the new cluster as described in DMAC algorithm, MDMAC proposes a new mechanism that estimates the time that will be part of the cluster. Using this new feature, vehicles select the cluster with the greatest time in order to minimize cluster changes. Another feature of the MDMAC compared to DMAC is that before a vehicle is attached to a CH, verifies that the two vehicles are moving into the same direction. The idea here is that vehicles with different angle will be soon out of each other’s range. Finally, the last addition compared to the DMAC algorithm is the support for multi-hop clusters (i.e. CMs can be k hops away from CH).

3.3 Mobility-Based Clustering in VANETs using Affinity Propagation

Affinity PROpagation for VEhicular Networks (APROVE) clustering algorithm uses affinity propagation technique where data points pass messages to one another, describing the current affinity that one data point has for choosing another data point as its exemplar [4]. APROVE implemented affinity propagation idea into a VANET environment in a distributed fashion.

Vehicles, according to this algorithm, periodically broadcast Hello messages containing their mobility information (i.e. position, velocity, direction) and their current CH. A vehicle considers as neighbors, only vehicles moving in the same direction. On the reception of those messages, vehicles calculate similarity function for each one of their neighbors. Similarity function has been defined as a combination of the negative Euclidean distance between vehicle positions now and negative Euclidean distances in the future as shown in Figure 3-2. Additionally, except for the periodic messages, APROVE algorithm periodically broadcasts availability and responsibility messages. Responsibility messages represent node's suitability to become CH and availability messages indicate the desire of a node to become a CH. Current availability and responsibility value for each neighbor is damped with the previous transmitted value so as to give memory to the algorithm. Figure 3-3, shows the equations for availability and responsibility messages. On the reception of those messages, vehicles update the responsibility and availability values received from their neighbors.

$$s(i,j) = -\left(\|\mathbf{x_i} - \mathbf{x_j}\| + \|\mathbf{x_i'} - \mathbf{x_j'}\|\right)$$

$$\mathbf{x_i} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \mathbf{x_i'} = \begin{bmatrix} x_i + v_{x,i}\tau_f \\ y_i + v_{y,i}\tau_f \end{bmatrix}$$

Figure 3-2: Similarity function equation

$$r(i,j) \leftarrow s(i,j) - \max_{j'\, s.t. j' \neq j} \left\{ a(i,j') + s(i,j') \right\}$$

$$\underset{i \neq j}{a(i,j)} \leftarrow \min \left\{ 0, r(j,j) + \sum_{\forall i' \notin \{i,j\}} \max \left\{ 0, r(i',j) \right\} \right\}$$

$$a(j,j) \leftarrow \sum_{i'\, s.t. i' \neq j} \max \left\{ 0, r(i',j) \right\}$$

Figure 3-3: Availability and responsibility values equations

$$CH_i = \arg\max_j \{a(i,j) + r(i,j)\}$$

Figure 3-4: Cluster Head calculation equation

After the reception of all those messages, the maintenance phase starts and the vehicles purge old entries in their neighbor lists, while checking if their CH is still in range. If the CH has been lost, vehicles search in the existing neighbor list to find another suitable CH or become CH themselves if there is no CH available in the vicinity. Vehicles use equation in Figure 3-4 to periodically calculate their new CH according to an interval period of time (i.e. indicatively every 10-20 periodic messages).

4. SIMULATION ENVIRONMENT

In this section, we provide the implementation details of this work. The first part of this section contains a detailed description of the environment and the tools we used, while the second part consists of the technical details of the algorithms that we have implemented as described in section 3. The final part of this section describes the integration activities of several tools used in the implementation.

4.1 Environment Configuration

In the scope of this work, we have set a completely new, LINUX based environment downloading and installing the latest versions of ns-3 simulator (3.21) [17], and SUMO traffic simulator (0.23) [18]. Both of those tools have excellent support in LINUX operating systems.

4.1.1 NS-3 Simulator

NS-3 Simulator is a discrete-event network simulator for Internet systems, targeted primarily for research and educational use. It is an ongoing project since 2011, it is free software, licensed under the GNU GPLv2 license, and is publicly available for research, development, and use. The community except from the major development releases (1-2 per year) organizes annually workshops. Our publication was accepted in the workshop that took place in Barcelona as part of publications for 2015.

NS-3 is written in C++ programming language, with bindings also available for Python language integration and thus, simulation programs could be either C++ executables or Python programs. Figure 4-1 presents the sequence to generate python bindings from C++ source code. NS-3 actually in not backwards-compatible with NS-2 Simulator since the community was intended to create a new strong tool avoiding the weakness of its predecessor. Although, due to the fact that both tools have been developed using C++ language some features have been imported from NS-2 to NS-3 platform (i.e. SUMO integration classes).

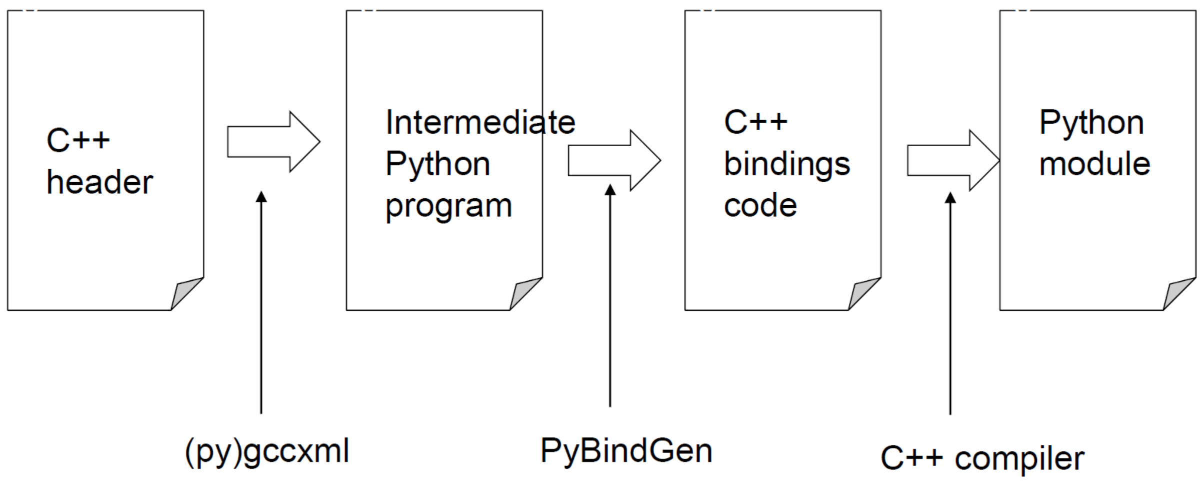


Figure 4-1: PyBindGen to generate python bindings

Current status of NS-3 contains plenty of modules. An overall overview of the modules is shown in Figure 4-2. Modules could be divided into 4 main categories:

1. Core: Those modules are the core functionality of the simulator (i.e. events, queues, schedulers)
2. Device: Those modules simulate a specific type of device (i.e. wifi, wimax, lte)
3. Protocols: Protocol modules consist of the implementation of several protocols such as routing, openflow e.t.c.
4. Utilities: Utilities modules that are used mainly for gathering statistics or provide a Graphical User Interface for the topology visualization (i.e. netanim, pyviz). A netanim example topology is shown in Figure 4-3.

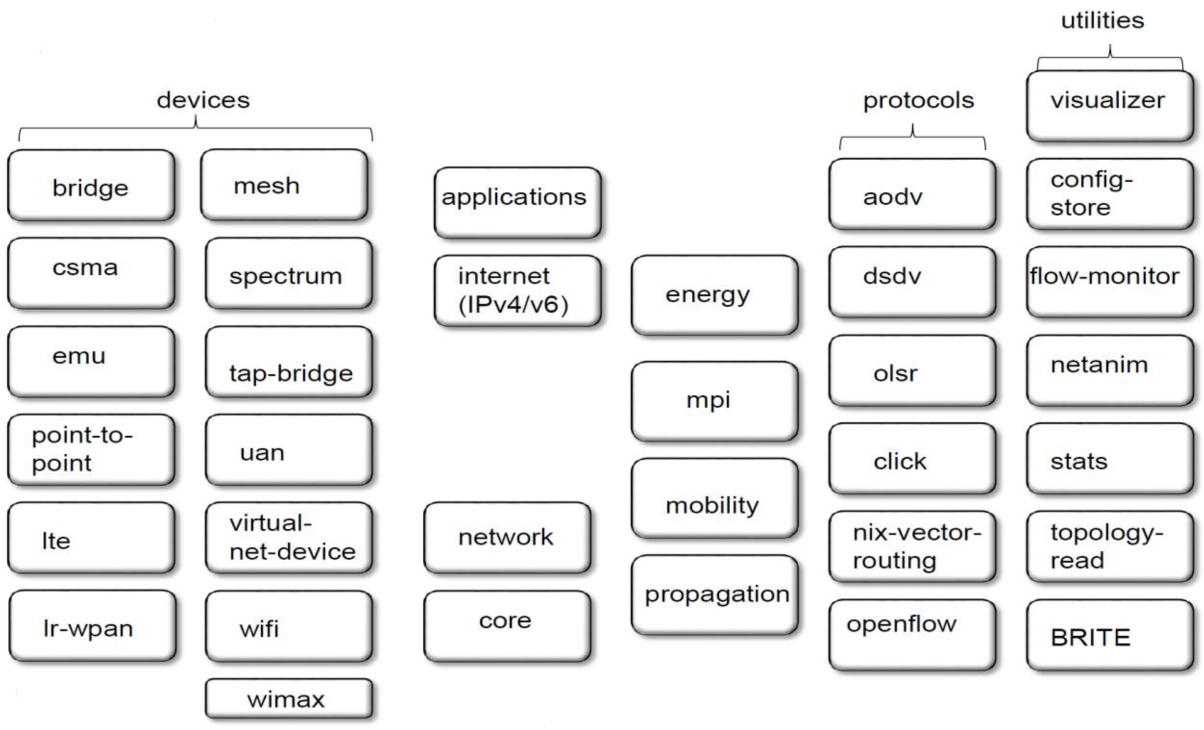


Figure 4-2: NS-3 modules overview

Some of those modules have been imported into NS-3 as part of other open source projects such as ndn Simulator and mptcp-ns3. Other modules also could be supported both for simulation and for emulation. It is possible to run real time experiments using TapBridge integration from Mininet for example. Mininet is popular in SDN (Software Defined Radio) community for building realistic virtual networks, running in real kernels, switches and application code, on a single machine (VM, cloud or native).

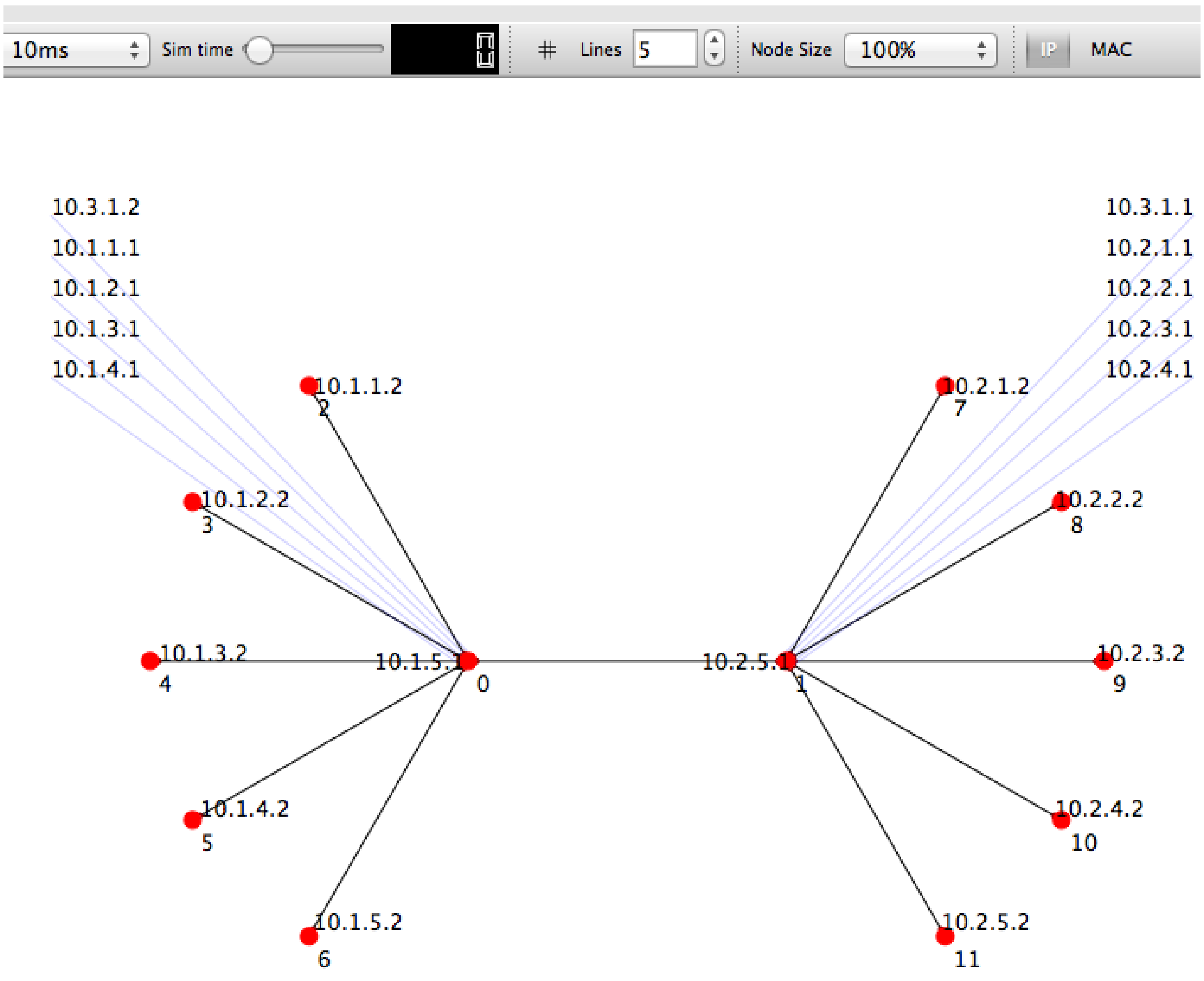


Figure 4-3: Netanim visualization tool

4.1.2 SUMO traffic Simulator

SUMO is a purely microscopic traffic simulator designed to handle large road networks. It is an open source project started at 2001 from the German Aerospace Center. SUMO provides several command line tools to create your own topology examples and custom trips and flows.

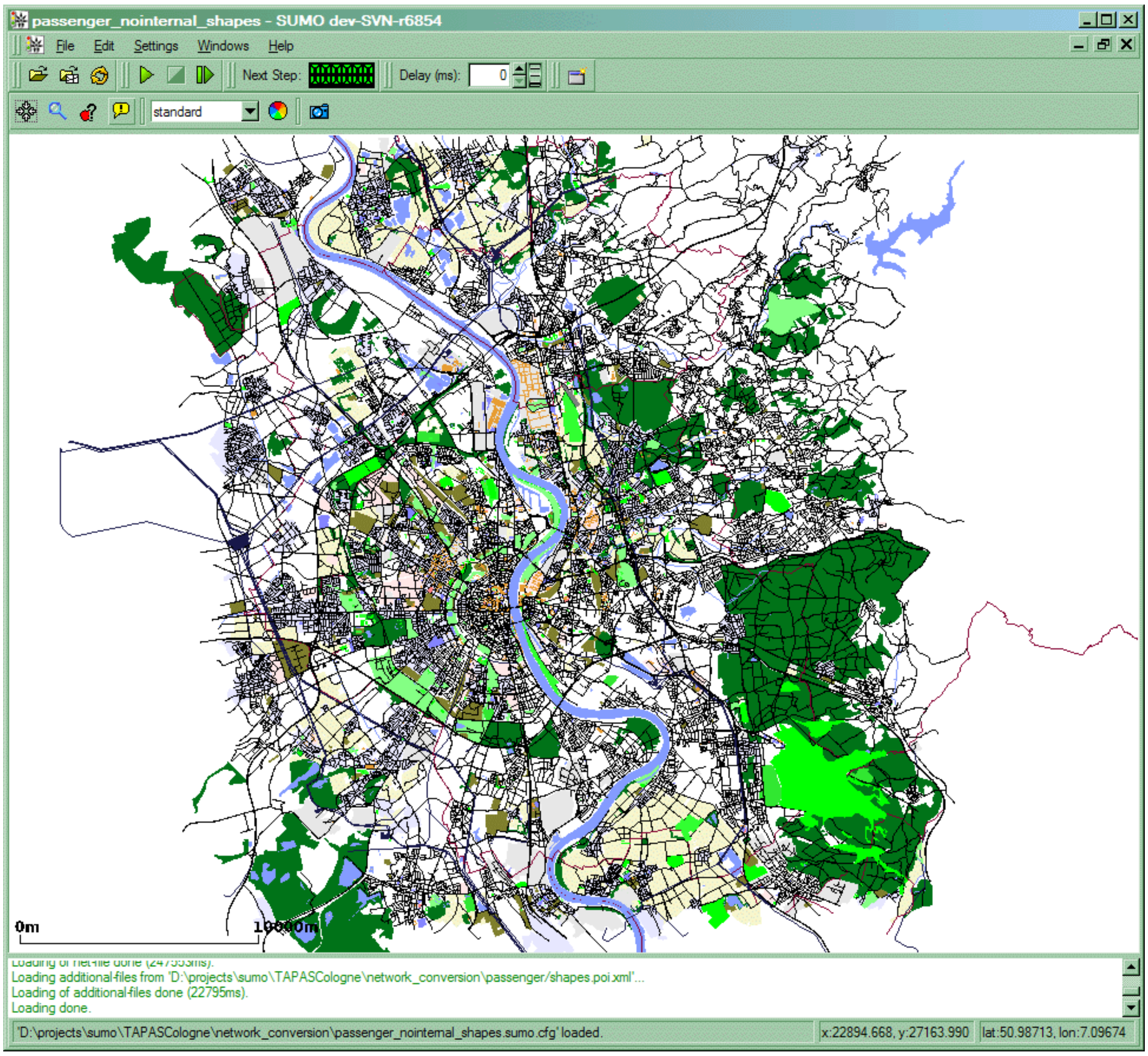
Internally in SUMO [20], each vehicle is represented by a unique identifier (name), the departure time, and the vehicle's route through the network. Optionally, each vehicle can be described more detailed using additional properties. Parameters such as departure and arrival time, departure and arrival position, road lane, and velocity of the vehicles could be easily define through xml description files. There are also additional variables allow the definition of the vehicle's appearance within the simulation's graphical user interface. Topology is created by connecting nodes and edges. A number of attributes is also available of the nodes and edges configuration such as number of lanes, priority value, upper speed value, one way edge or not e.t.c

The simulation is time-discrete with a default simulation step length of 1 second. It is space-continuous and internally, each vehicle's position is described by the lane the vehicle is on and the distance from the beginning of this lane. Each vehicle's speed is computed using a so-called car-following model, while moving through the network topology. A pure command line application for efficient batch simulation is available as well as a graphical application which renders the performed simulation using openGL.

SUMO could be characterized as a powerful tool for vehicle simulation since the community provides tools that help in the integration with non-SUMO networks except from defining custom xml networks. Here is a list with supported integration with non-SUMO networks:

- OpenStreetMap
- VISUM
- Vissim
- OpenDRIVE
- MATsim
- ArcView (shapefiles)
- Elmar's GDF
- Robocup Simulation League

OpenStreetMap is of great interest since it is also an open source project providing maps representation of the real world. In the current thesis we have used this framework in order to retrieve a realistic map topology for our simulation scenarios in order to evaluate the clustering algorithms that we have implemented as much as possible closer to real life scenarios. Using the SUMO converter tool provided from the SUMO community, we restructure files taken from OpenstreetMap in a form compatible with SUMO simulator. Afterwards, we use the random trip generator tool of SUMO in order to push a number of vehicles into the map topology taken from OpenStreetMap. Random generator could receive a great number of parameters that helps generate realistic vehicle movements in a given topology. In the annex we provide details on how to build the SUMO topology. An example of a real topology using OpenStreetMap of Cologne region in Germany could be seen in **Error! Reference source not found.**.



```
Loading of net-file done (247553ms).
Loading additional-files from 'D:\projects\sumo\TAPASCologne\network_conversion\passenger/shapes.poi.xml'...
Loading of additional-files done (22795ms).
Loading done.
```

```
'D:\projects\sumo\TAPASCologne\network_conversion\passenger_nointernal_shapes.sumo.cfg' loaded.    x:22894.668, y:27163.990    lat:50.98713, lon:7.09674
```

Figure 4-4: Cologne Germany, using OpenStreetMap and SUMO

4.2 Algorithm implementation in NS-3 Simulator

In this section we provide technical details for the clustering algorithms presented in section 3. In ANNEX I, we provide example code for running and testing the three clustering algorithms. In order to implement the aforementioned algorithms, we have extended the default application class on the NS-3 simulator for each one of the three algorithms so as NS-3 nodes to be able to run the algorithms as standard applications. Figure 4-5 shows the inheritance diagram of the new application classes.

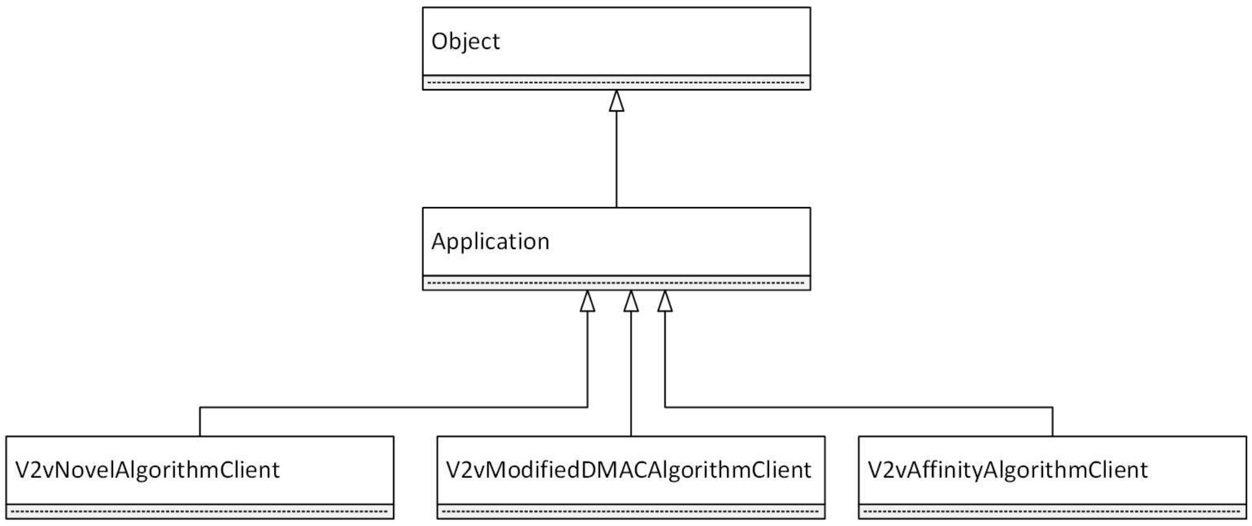


Figure 4-5: Application inheritance diagram

Except from the different clustering process that is unique for each algorithm, the three algorithm implementations have some common structure and features. In more details, each of the three algorithms has 2 sockets. The first one is responsible to handle packets received from the neighboring vehicles, while the second one is responsible to send the appropriate packet at each step of the algorithm. Another common feature of the three algorithms is the channel access scheme used. Although, 802.11p protocol is available in the latest releases of the NS3 simulator, since version 3.21 there was no channel access scheme implemented yet. In order to avoid problems occurred from collisions and retransmission mechanism, we have implemented a simple TDMA access scheme that provides to each vehicle the appropriate slots to transmit the packets required from the algorithms. A limitation to this implementation is that currently supports up to 100 vehicles.

In order to support a vehicle communication, we had to define the messages that are exchanged for each type of algorithm. For this purpose, we have extended the built-in header class of ns-3 by implementing all the messages required for each algorithm. Figure 4-6 shows the inheritance diagram of the header messages.

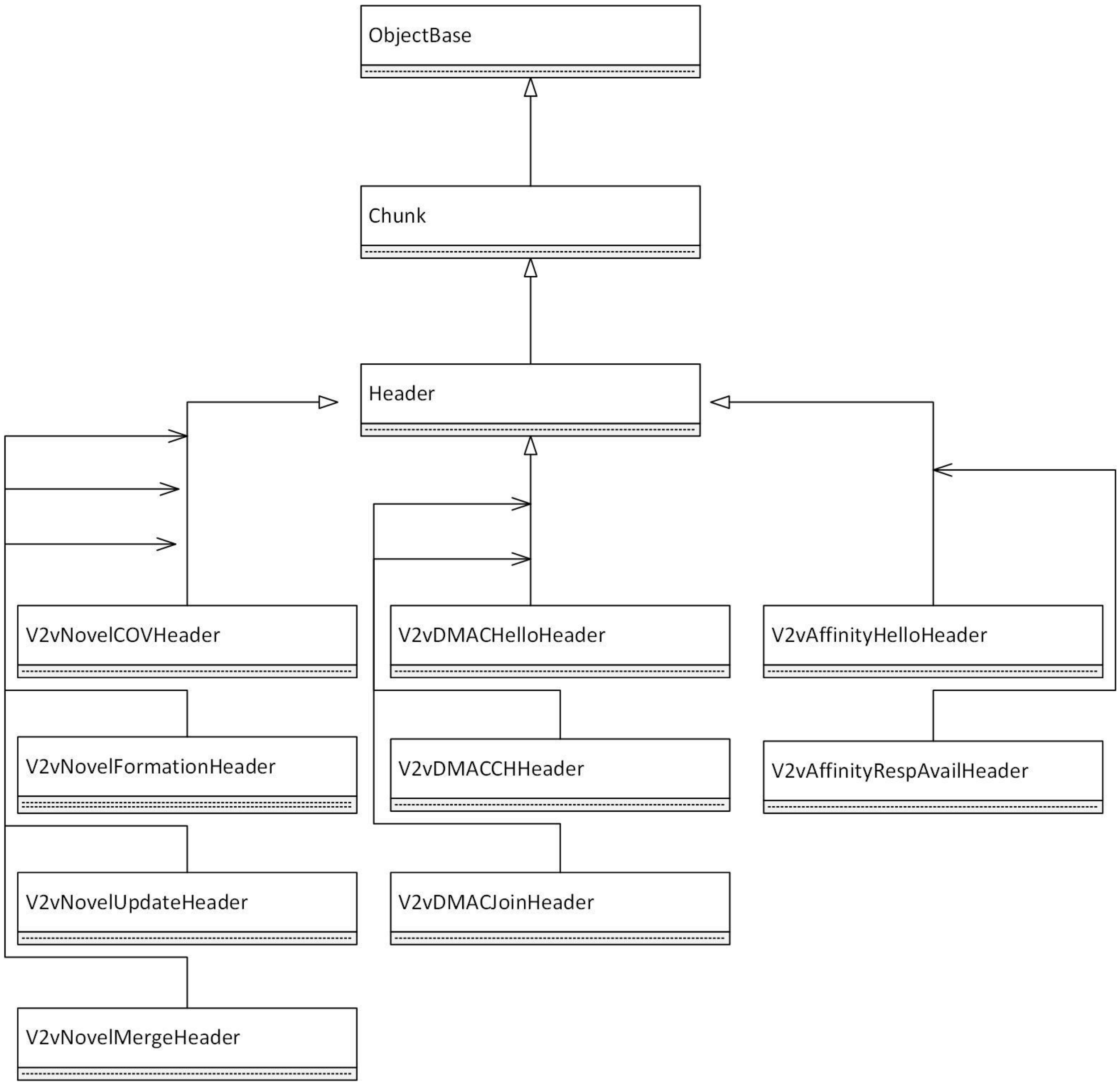


Figure 4-6: Header message inheritance diagram

Each one algorithm has been modeled as a Finite State Machine (FSM). In the following sections we present the states of each algorithm and we also provide the content of the header messages in each case.

4.2.1 V2vNovelAlgorithmClient

Novel algorithm has been modeled as a 4 state Finite State Machine. In the algorithm start up, vehicles are in INITIAL state starting the periodic messages process. Immediately after receiving neighbor's information, vehicles enter in COV state where the slowest vehicle in the vicinity area starts the formation algorithm by sending its temporary cluster id to its neighbors. Stable neighbors temporarily become part of the cluster starting the FORMATION process where the CH election process takes place. Non-stable neighbors that are not part of this process start again this process from scratch. It is important to notice here that periodic update messages that report vehicle's current state and mobility information continue till the end of the simulation broadcasting the periodic messages. If a vehicle is not receiving update messages from a neighbor anymore, means that this neighbor is currently out of range and thus is removed from the neighbors map. Although when a vehicle changes cluster is still in the same state, but if a vehicle turns in a standalone state with no neighboring vehicles then is set to INITIAL state in order to be ready to start formation process again when a neighboring will be in range.

In the rest of this paragraph we provide the structure and the information of the packet headers implemented in the scope of this algorithm.

V2vNovelCOVHeader message was initiated from the slowest vehicle in range providing its temporary cluster identifier during the cluster formation process. Its structure is shown in Table 1.

Table 1: V2vNovelCOVHeader fields

Timestamp(64bits)	Temporary Cluster Id(64bits)

V2vNovelFormationHeader message is used from the newly elected CH to announce its new state to the other vehicles in range. This message follows it time V2vNovelCOVHeader message described above. A representation of the header structure is shown in Table 2.

Table 2: V2vNovelFormationHeader fields

Timestamp(64bits)	Cluster Id(64bits)	Temporary Cluster Id (64bits)

V2vNovelUpdateHeader message is sent during the periodic update process of the vehicles containing information about their current state. It is useful to note here that Vector attributes are 3D vectors of size (3*64 = 192 bits). A representation of this header message is shown in Table 3.

Table 3: V2vNovelUpdateHeader fields

Timestamp(64bits)	Id(64bits)	Cluster Id(64bits)
Temporary Cluster Id (64bits)	Cluster Members(64bits)	Position(Vector)
Velocity(Vector)	Direction(Vector)	Degree(64bits)

When two CH come to each other’s range, a V2vNovelMergeHeader message is sent from the CH having the lowest number of CMs to inform its CMs for the cluster change before will be attached to the new cluster. Table 4 shows the structure of such a message.

Table 4: V2vNovelMergeHeader fields

Timestamp(64bits)	Old Cluster Id(64bits)	New Cluster Id(64bits)

4.2.2 V2vModifiedDMACAlgorithmClient

MDMAC algorithm starts by sending the periodic update messages. After vehicles become aware of their neighbors, calculate the weight of each one their neighbors, before deciding to send a CH or Join message according to their current state. Periodic messages of course are being broadcasted continuously after the algorithm initialization. If a vehicle has not received and update message from a neighbor the last second, assumes that this neighbor is out of range and thus is removed from the neighbors map. When a CH change event occurs or when a vehicle loses its CH, recalculation of CH takes place before sending a CH or Join message. When the TTL value given to the algorithm is greater than one, the algorithm works as a multi-hop distributed process and directly after the reception of each type of message, the same messages are forwarded to the new neighborhood.

In the rest of this paragraph we provide the header packet structure related to MDMAC algorithm. Vector attributes are 3D vectors of size (3*64 = 192 bits) exactly the same with V2vNovelUpdateHeader vector fields.

V2vDMACHelloHeader message is used from MDMAC algorithm for the periodic update of the neighbors. Its structure is shown in Table 5.

Table 5: V2vDMACHelloHeader fields

Id(64bits)	TTL(16bits)	Weight(64bits)
Position(Vector)	Velocity(Vector)	Direction(Vector)
Role(64bits)		

V2vDMACCHHeader message is sent to the neighboring vehicles when the node turns to CH state. Its fields are shown in Table 6.

Table 6: V2vDMACCHHeader fields

Id(64bits)	TTL(16bits)

V2vDMACJoinHeader is sent to the neighboring vehicles when the node turns to CM state. A representation of such a message is shown in Table 7.

Table 7: V2vDMACJoinHeader fields

Id(64bits)	TTL(16bits)	CH index(64bits)

4.2.3 V2vAffinityAlgorithmClient

Affinity propagation algorithm is precisely divided into 3 states. Algorithm initially enters in TH state where is responsible to retrieve and broadcast the mobility characteristics of the vehicle plus the index of its current CH. On the reception of such a message, vehicles calculate the similarity value of each neighbor. Right after the periodic update message is sent, vehicles enter in TM state where they accumulate the responsibility and availability values of each one of their neighbors. This information is also broadcasted before the vehicles enter in TCM state where the cluster maintenance takes place. Vehicles remove neighbors that have not sent update message the last second. The structure of each one of the messages implemented for this algorithm are shown below.

V2vAffinityHelloHeader message is sent during the periodic update messages sent to neighboring vehicles. Vector attributes are 3D vectors of size (3*64 = 192 bits) exactly the same with V2vNovelUpdateHeader vector fields. Table 8 shows the structure of such type of message.

Table 8: V2vAffinityHelloHeader fields

Timestamp(64bits)	Id(64bits)	CHindex(64bits)
Position(Vector)	Velocity(Vector)	Direction(Vector)

V2vAffinityRespAvailHeader message is sent in TM state of the vehicles, while vehicles broadcast their responsibility and availability values. “Neighbors Number” field shows the number of tuples (id, Responsibility, Availability) that follow. Thus the size of this header message is not constant but depends on the number of neighboring vehicles. Table 9 contains a representation of this message.

Table 9: V2vAffinityRespAvailHeader

Id(64bits)	CH converge(1bit)	Neighbors Number(64bits)
Id(64bits)	Responsibility(64bits)	Availability(64bits)
Id(64bits)	Responsibility(64bits)	Availability(64bits)
….	…..	…..
Id(64bits)	Responsibility(64bits)	Availability(64bits)

4.3 NS-3 and SUMO integration

In order to stress the algorithms behavior into more realistic environment we had to generate advanced mobility patterns for the vehicles. NS-3 built-in mobility models could not fit in such a scenario since vehicles simulation needs advanced mobility patterns with multiple paths and realistic traffic movement. In this part we give some details on how to import the traces exported from SUMO into NS-3 simulator. Advanced description and screenshots of the topology that we have created could be found in section 5.

The solution to this integration task is to use NS2 mobility helper which is also available in NS-3 simulator to parse the traces exported from SUMO. Before this, it is important to remove any other mobility instances from the nodes of NS3. The rest of the integration activities are rather simple. Just pass as parameter the number of vehicles are used in the traces and the simulation time to avoid inconsistencies in the configuration of the environment.

5. SIMULATION SCENARIOS

After the information that we have provided for the SUMO simulator and the steps for the integration with NS-3, in this section we show how to create a new topology from scratch. For the evaluation of our algorithms, we produced two different topology scenarios since it is important to know how each algorithm reacts both in urban and in rural scenarios. The first scenario simulates vehicles movement in the Manhattan square in New York, while the second one simulates vehicles in Batavia a small city new New York.

More specifically, for the urban scenario representation, we have used OpenStreetMap to retrieve part of Manhattan square from New York. To our understanding this scenario is representative for the movement of the vehicles in the centre of the cities. As for the rural scenario, we used again OpenStreetMap to retrieve part of a small city in a rural area with highways near to the city of New York. The difference in two scenarios is of course the absolute value of velocity that the vehicles can reach. In the first scenario any intersection of the road is controlled from traffic lights which forces vehicles to start and stop their movement often enough. In the second scenario the traffic lights and the intersections in the roads are reduced considerably.

ANNEX II shows the steps on how to reproduce the topology scenarios starting from the maps downloaded from OpenStreetMap till the conversion from SUMO to NS-3 compatible traces.

Figure 5-1 and Figure 5-2 show the Manhattan square and Batavia city respectively, from the SUMO Graphical User Interface perception.

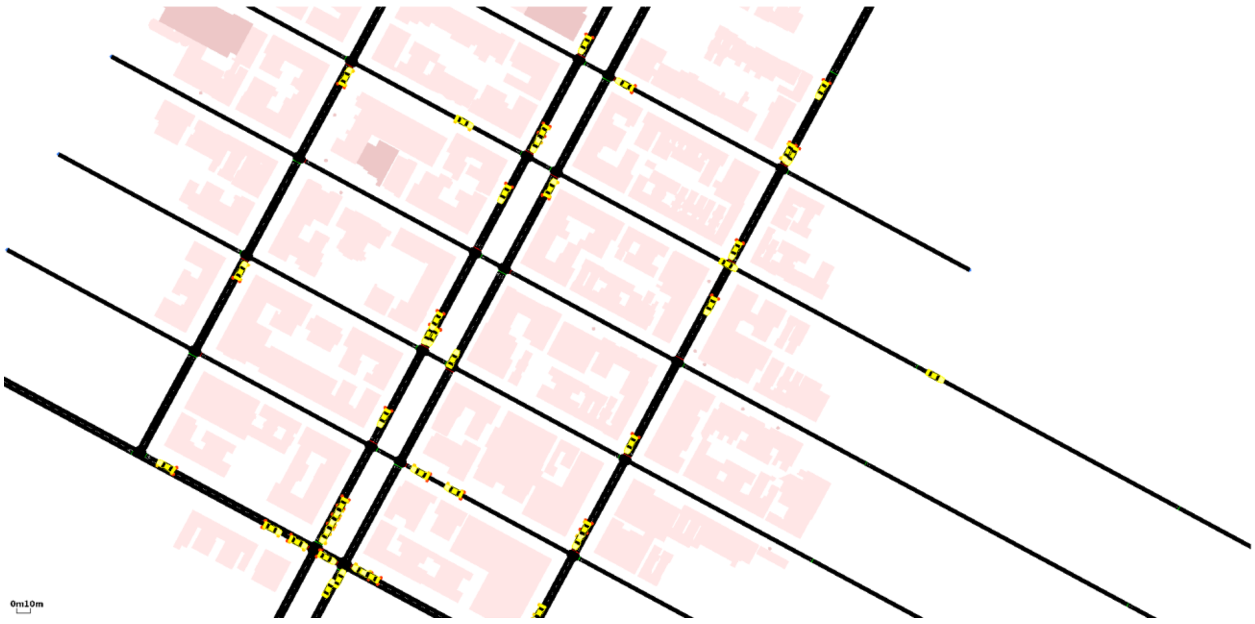


Figure 5-1: Urban scenario – Manhattan square

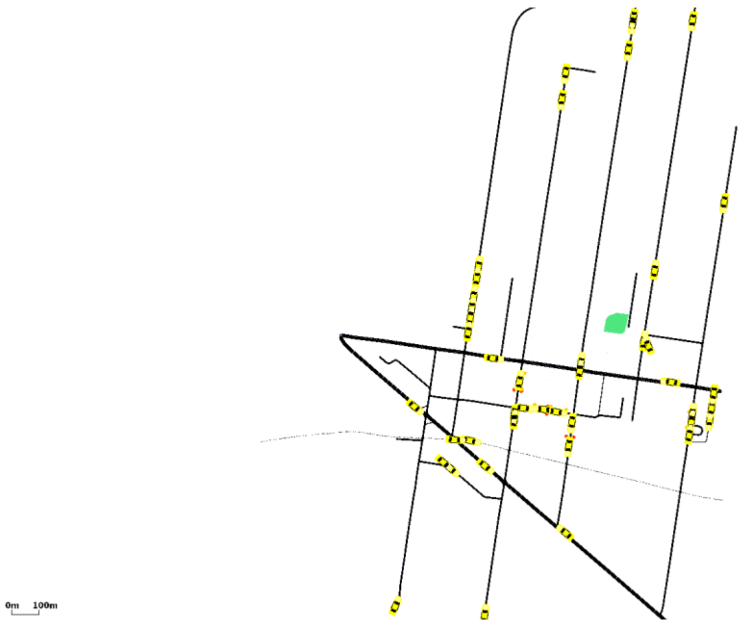


Figure 5-2: Rural scenario – Batavia rural area

The following table contains the configuration of all the parameters that were used in the simulation scenarios. Both in the urban and in the rural scenario we kept the same configuration in order to evaluate the properness of each algorithm in each of the two scenarios. It is useful to note that using the Tx power, Tx/Rx Gain and Energy Detection Threshold in the receiver, as shown in Table 10, the transmission range of the vehicles is approximately of 300 meters long.

Table 10: Configuration Parameters

Parameter	Value
Number of vehicles	90
Tx Power	32 dB
Tx/Rx Gain	12 dB
Energy Detection Threshold	-71.8 dB
Cluster Update messages interval	1 s
Inactive neighbor threshold	1.5 s
Training Period	80 s
Simulation Time	350 s
Overall Simulation Time	430 s

6. PERFORMANCE EVALUATION

This section contains the performance evaluation of the algorithms, providing a number of important metrics for both two scenarios as described in the previous section. The whole section is divided into two parts. The first subsection contains the results of the Manhattan square scenario (urban), while the second subsection contains the results of the Batavia rural scenario.

6.1 Manhattan square – Urban scenario evaluation

Using the configuration as shown in Table 10, we run the 3 algorithms for 350 seconds after a warm up period of 80 seconds. Warm up period on the one hand is used to gradually insert the vehicles into the simulator and on the other hand helps in the consolidation of the random variables used in the simulator environment.

In the first set of plots we evaluate the cluster changes that are triggered from each algorithm during the simulation time. Figure 6-1Figure 6-1 demonstrates the number of cluster changes per second during the simulation time, while Figure 6-2 demonstrates the overall cluster changes performed by also providing the average number of cluster changes per vehicle for each one the implemented algorithms. Clearly, the novel algorithm performs better from any other algorithm, while DMAC has the greatest overhead compared to the other algorithms.

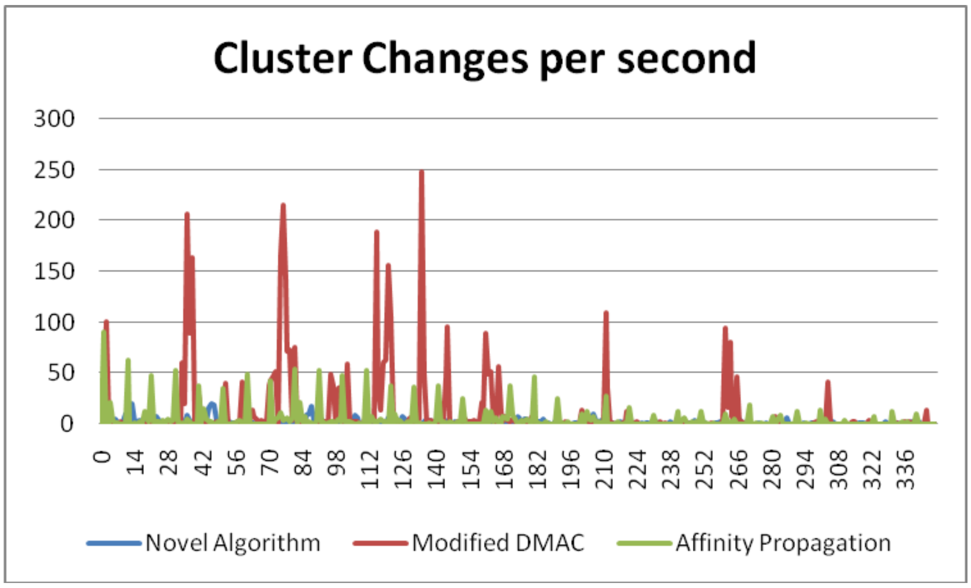


Figure 6-1: Cluster changes per second

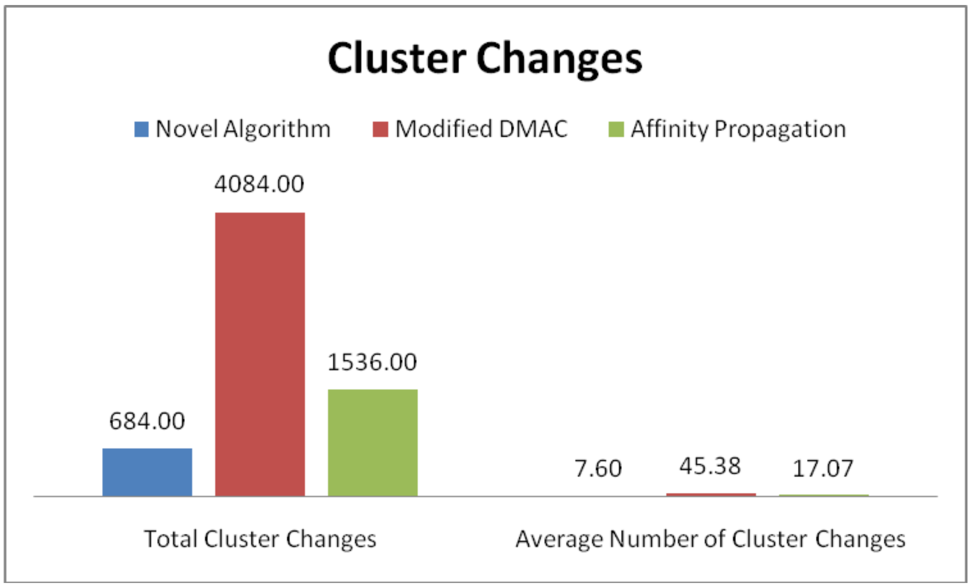


Figure 6-2: Cluster changes

In the second set of plots we focus on the number of clusters that exist during the simulation and in the average number of CMs reside into the clusters for each one of the algorithms. Figure 6-3 shows the number of clusters that exist in each second of the simulation time, while Figure 6-4 provides the average number of members in clusters. The performance of the 3 algorithms is quite close. Again Novel algorithm appears to be achieving the best distribution of the members in clusters.

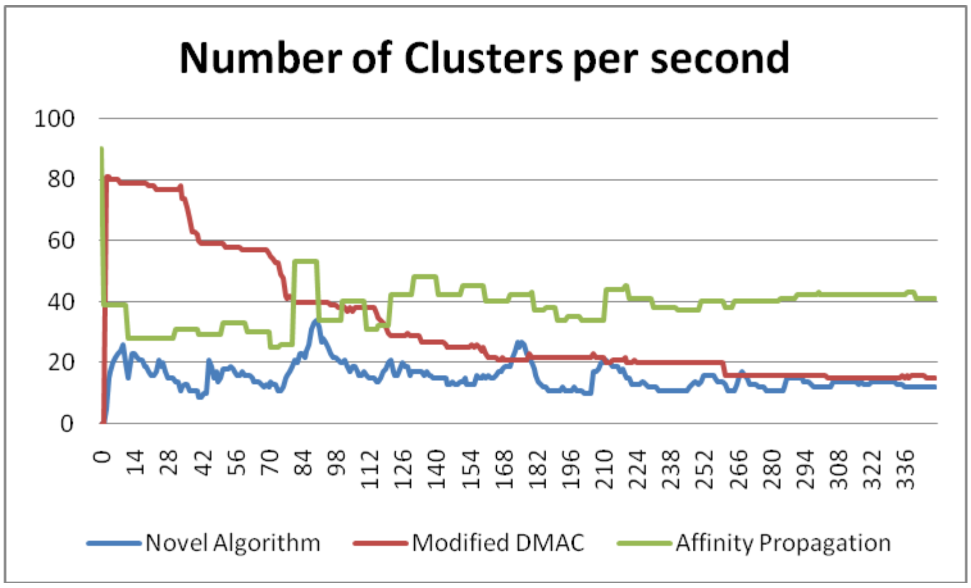


Figure 6-3: Number of clusters per second

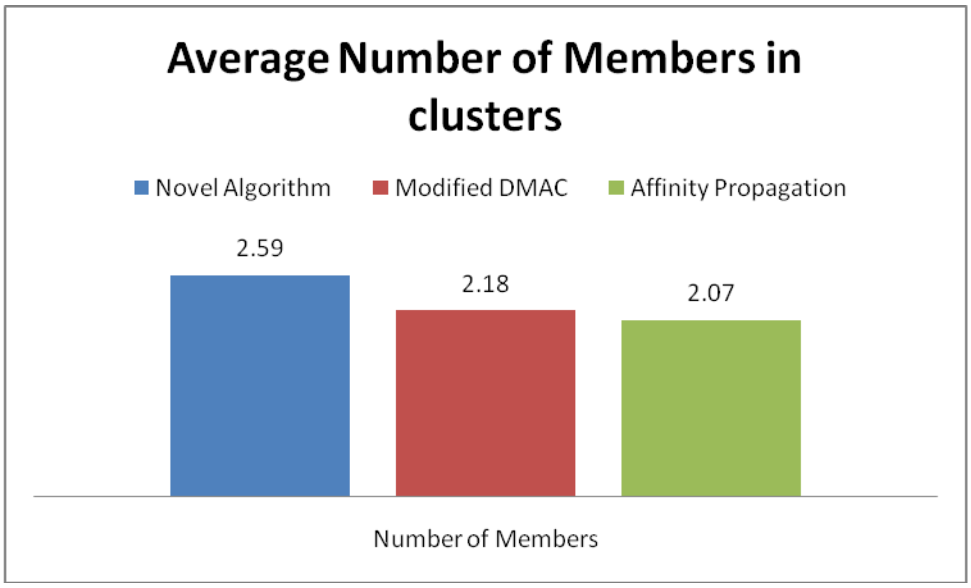


Figure 6-4: Average number of members in clusters

The next batch of plots compares the number of messages that are exchanged in each algorithm both for the formation and the maintenance of the clusters. Figure 6-5 demonstrates the number messages that are exchanged in the simulation environment each second. Figure 6-6 and Figure 6-7 show the average number of messages and the average number of messages per second respectively that have been exchanged for each algorithm in the simulation. We observe that APROVE has constantly the greatest number of exchanged messages since each algorithm's iteration requires the exchange of the responsibility and availability messages except from the periodic update message. In the other hand, the comparison of DMAC and Novel algorithm reveals that DMAC spends more messages than Novel algorithm. This is totally explained from the fact that DMAC reaches greatest number of cluster changes as we shown in Figure 6-2.

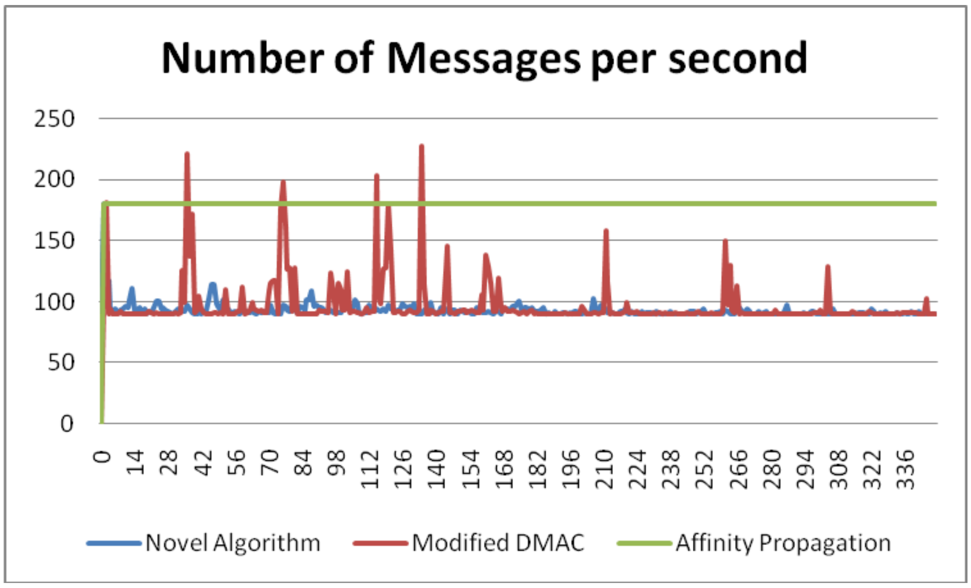


Figure 6-5: Numbers of messages per second

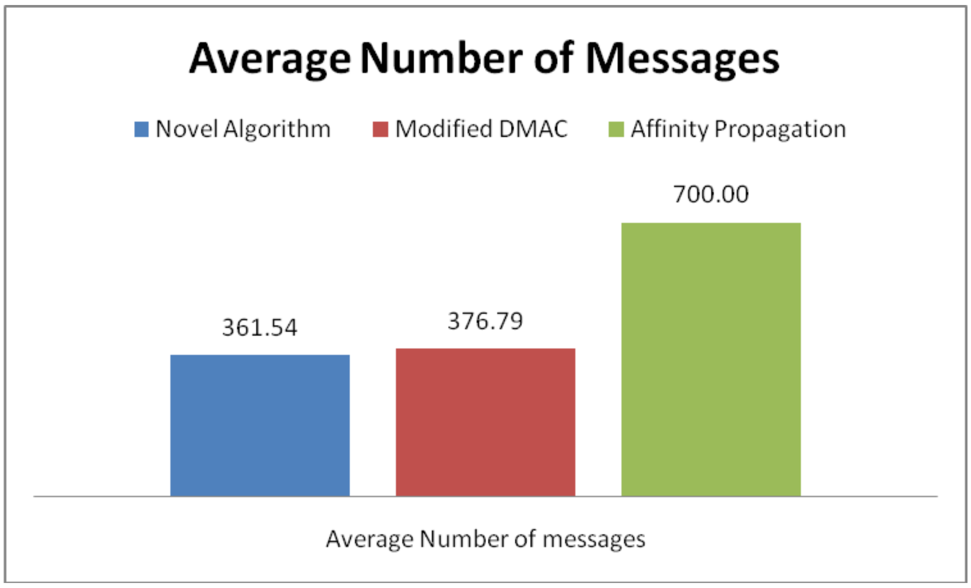


Figure 6-6: Average number of messages

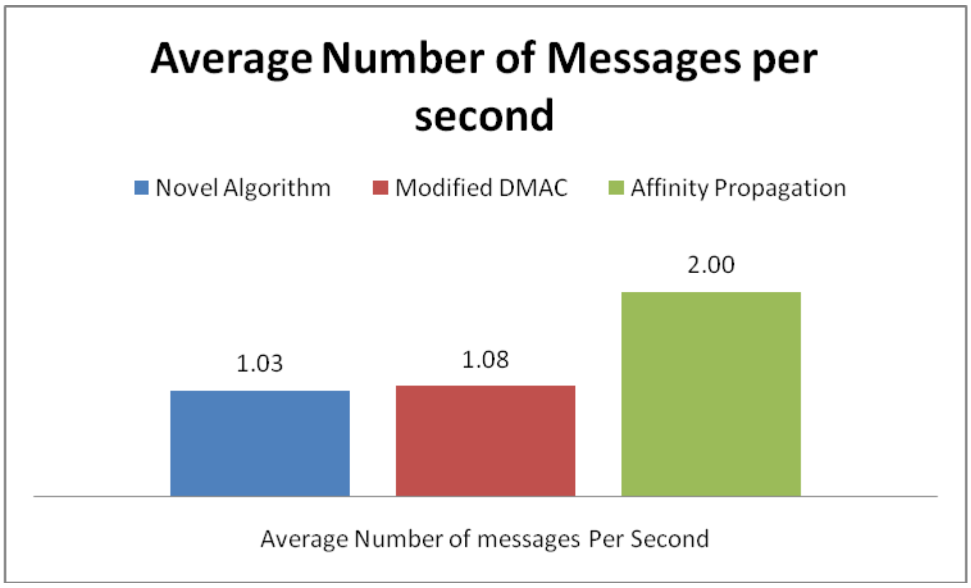


Figure 6-7: Average number of messages per second

The next metric for the clustering algorithm evaluation is the average time that is required for each algorithm to finish the clustering formation process. Because of its simplicity DMAC instantaneously performs cluster formation process. Novel algorithm shows important delay on cluster formation due to the cluster head election process. If the relative velocity of the neighboring vehicles is great enough, the waiting time before a vehicle announces that became CH be might many seconds.

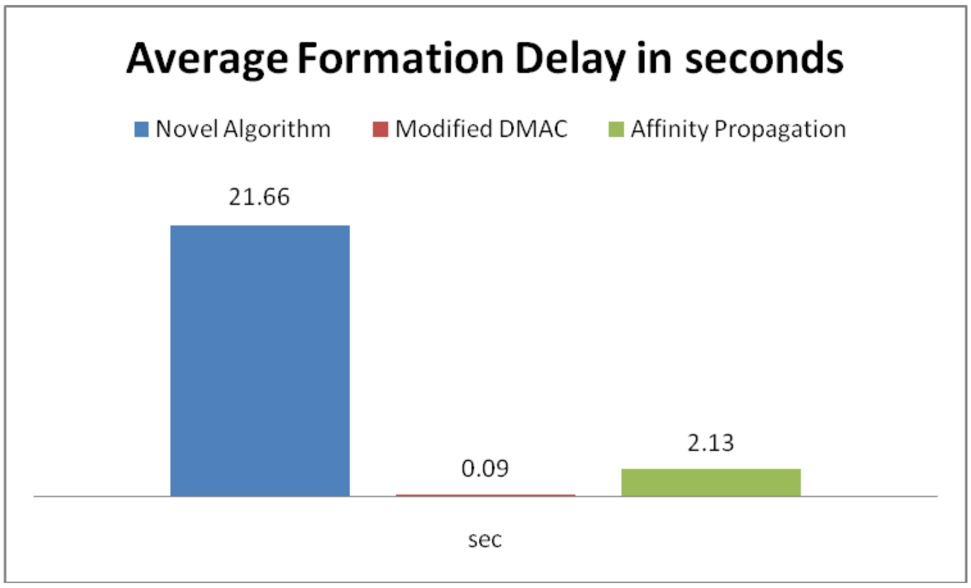


Figure 6-8: Average formation delay in seconds

The final set of 2 plots calculates the average time that vehicles act as CH and CM respectively. APROVE reveals great stability achieving great times both in CH and in CM average duration. APROVE achieves also the best minimum duration especially in CM duration. An important note here is low CH duration value of Novel algorithm which could be easily explained from the fact that is the only algorithm that supports cluster merge. Another important note here is the low average CM duration of the MDMAC algorithm that is relevant to great number of cluster changes shown in Figure 6-2.

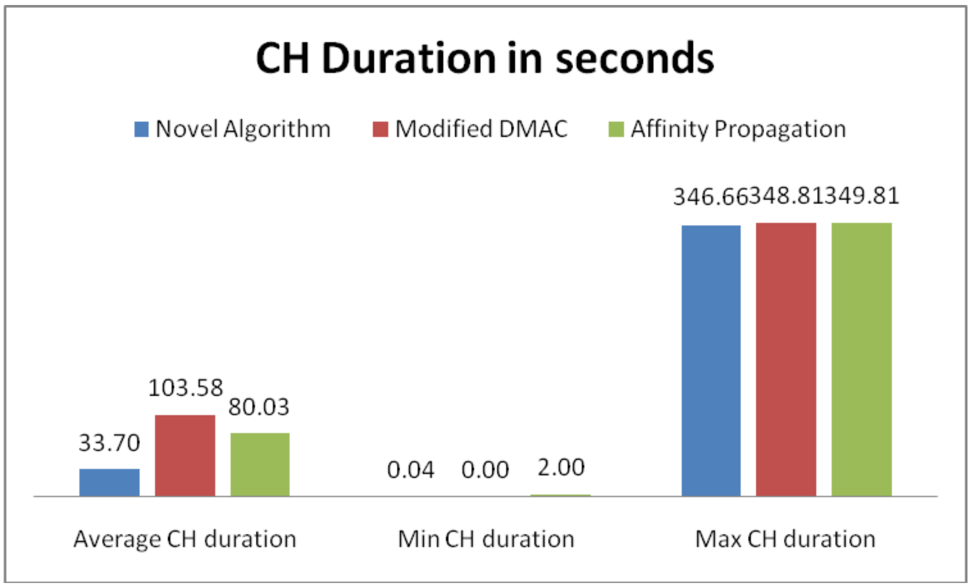


Figure 6-9: CH duration in seconds

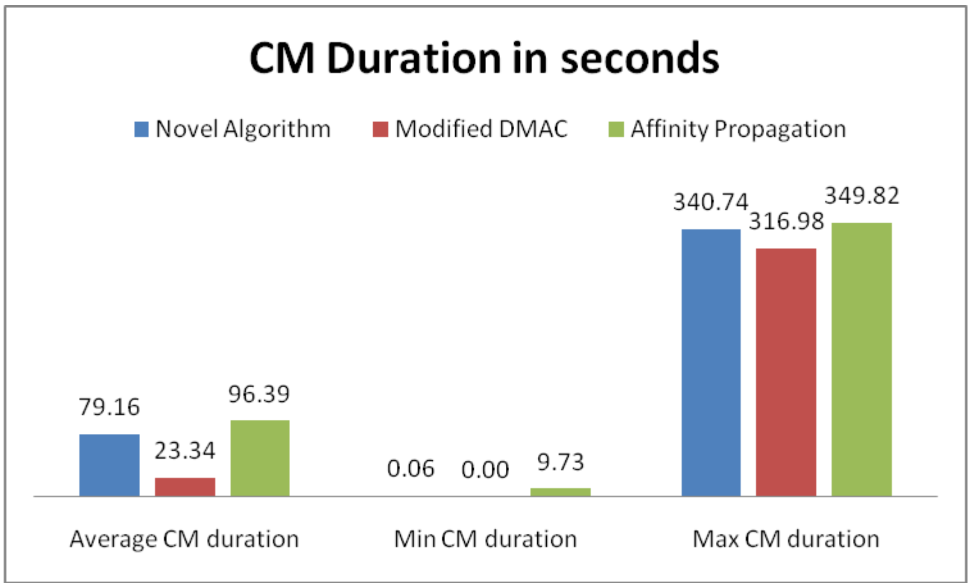


Figure 6-10: CM duration in seconds

6.2 Batavia – Rural scenario evaluation

This section contains the evaluation of the Batavia rural city of New York using exactly the same metrics as in Manhattan square scenario.

Figure 6-11 represents the number of cluster changes occurred during the simulation, while Figure 6-12 shows the total cluster changes occurred in each algorithm as well as the average number of cluster changes per vehicle. Comparing those plots with the relevant plots of Manhattan scenario we can observe that Novel algorithm is absolutely the most efficient scheme regarding the state changes of the vehicles, while the least efficient algorithm scheme is again MDMAC. Although, the performance of MDMAC and APROVE is improved, it is important to state that Novel algorithm has the same performance compared to Manhattan scenario.

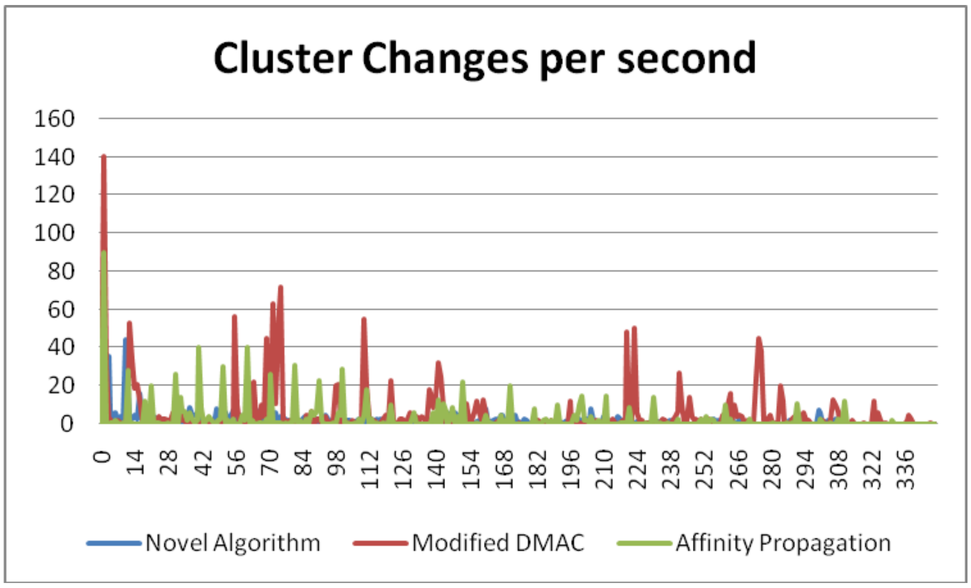


Figure 6-11: Cluster changes per second

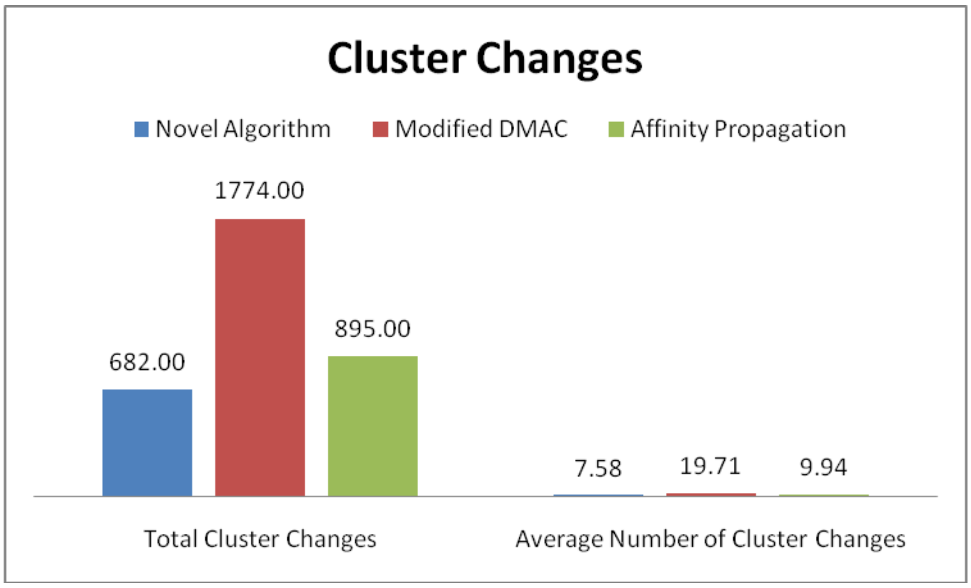


Figure 6-12: Cluster changes

The next two plots focus on the number of clusters that exist in the topology during the simulation and the average number of CMs in each algorithm. APROVE is again the algorithm with the greatest number of clusters in the topology which means that this scheme has the least average number of CMs as shown in Figure 6-14. From the one hand, APROVE seems to select its best neighbors as candidate CMs which means relatively stable cluster structure but in the other hand the existence of too many clusters is drawback for the efficiency of the mechanism. We also highlight the performance of the MDMAC algorithm. Its performance in terms of CMs distribution in clusters is remarkable in the rural scenario, while in the urban scenario is very close to the rest algorithms.

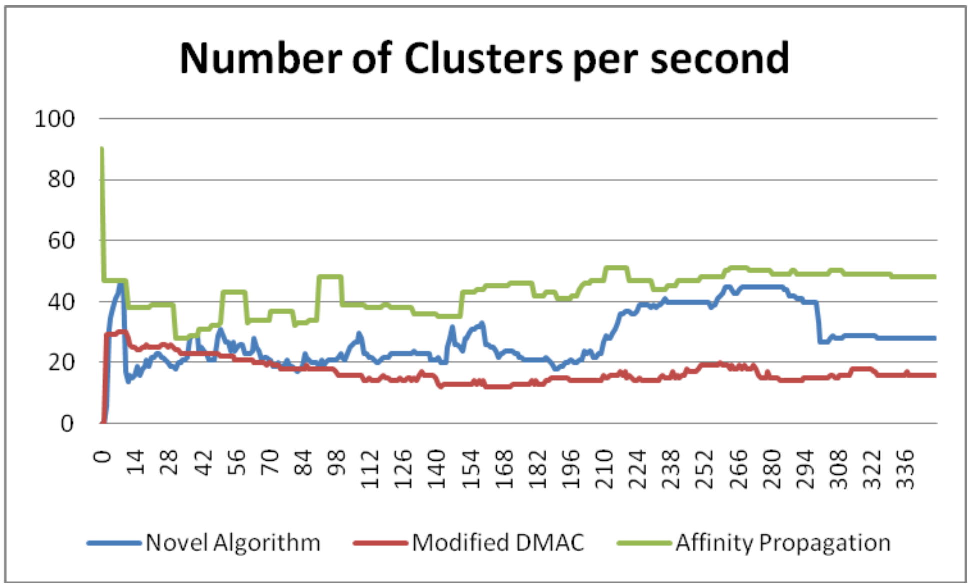


Figure 6-13: Number of cluster per second

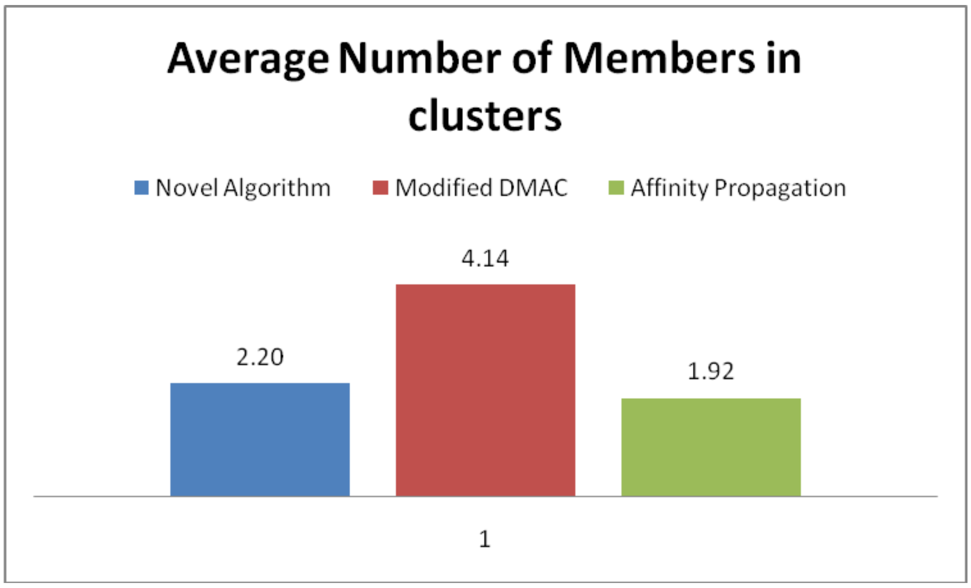


Figure 6-14: Average number of members in clusters

Figure 6-15, Figure 6-16 and Figure 6-17 demonstrate the messages exchanged in each algorithm per second, the in average and in average per second respectively. APROVE reaches the same number of messages since the number of messages sent is topology agnostic for this scheme. The better performance of MDMAC algorithm in the rural environment is reflected also in the reduced number of messages compared to the urban environment.

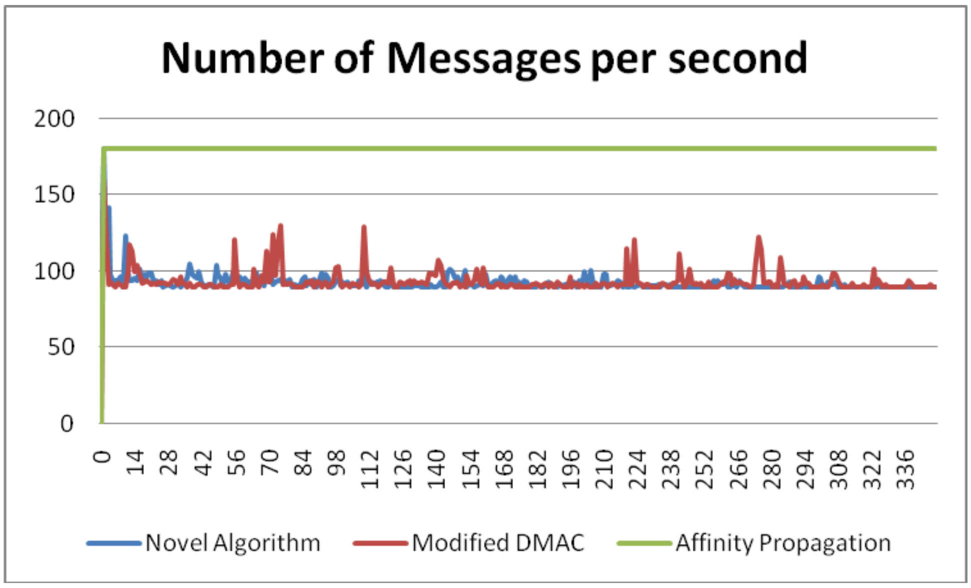


Figure 6-15: Number of messages per second

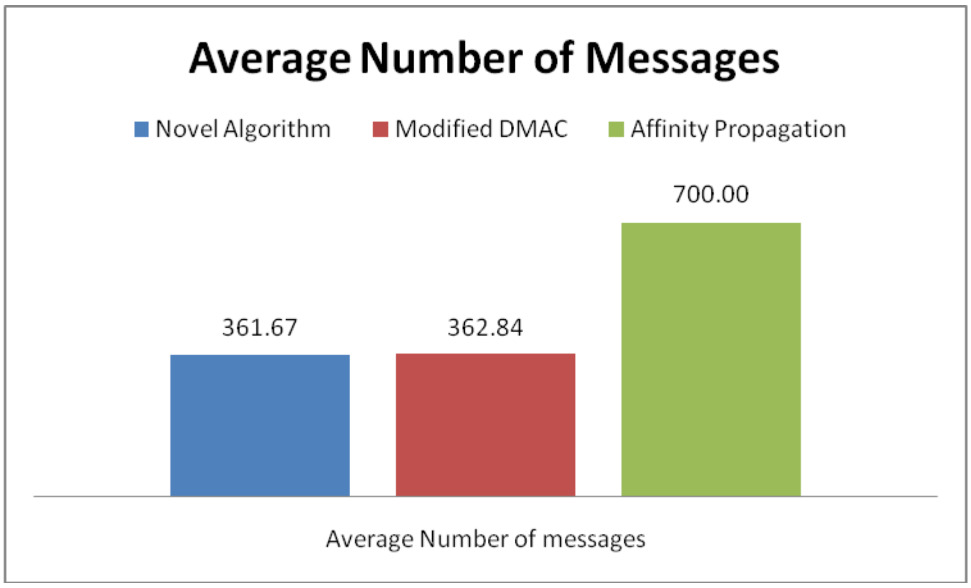


Figure 6-16: Average number of messages

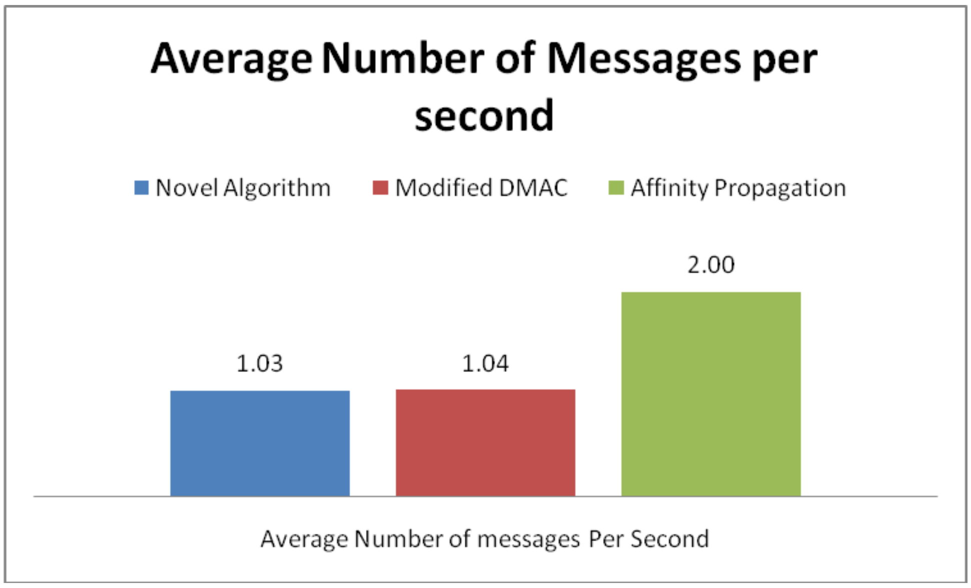


Figure 6-17: Average number of messages per second

Figure 6-18 shows the average formation delay of clusters for each one of the implemented algorithms. We can observe that in Novel algorithm, CH election process converges faster in rural scenario than in the urban scenario but still achieves the highest delay compared to the rest two schemes.

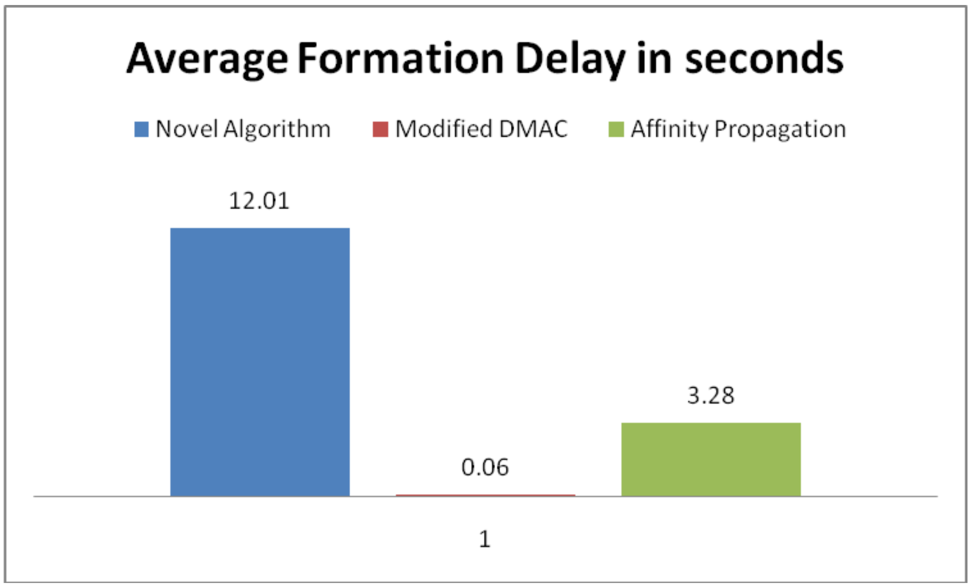


Figure 6-18: Average formation delay in seconds

In the final set of plots we measure the minimum, maximum and average duration of CH and CMs. For the CMs duration we observe that the reaction of the algorithms is not affected from the topology since APROVE is still better from any other clustering scheme, while the least efficient algorithm in terms of duration is scored from MDMAC. CH duration is affected in novel algorithm from the merge process as described in Figure 6-9.

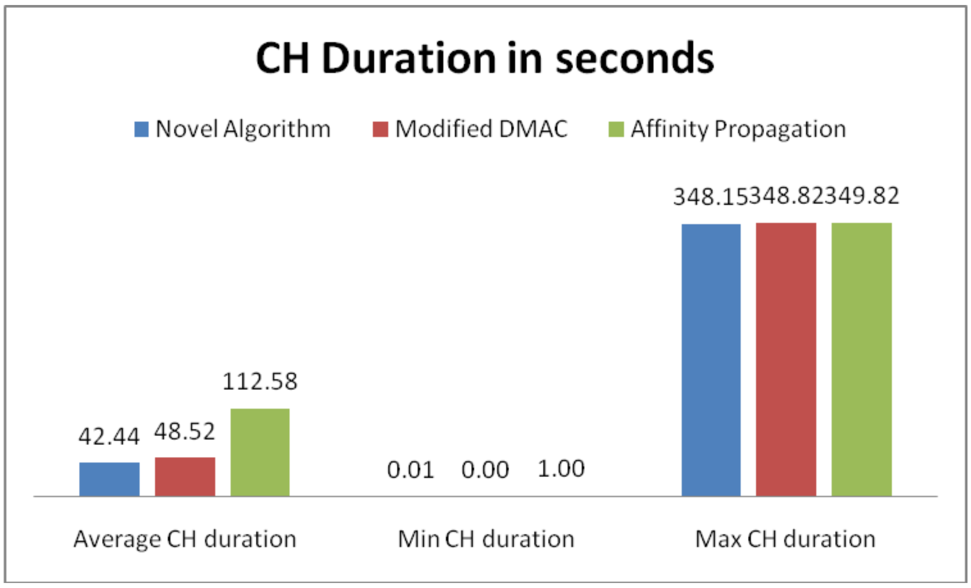


Figure 6-19: CH duration in seconds

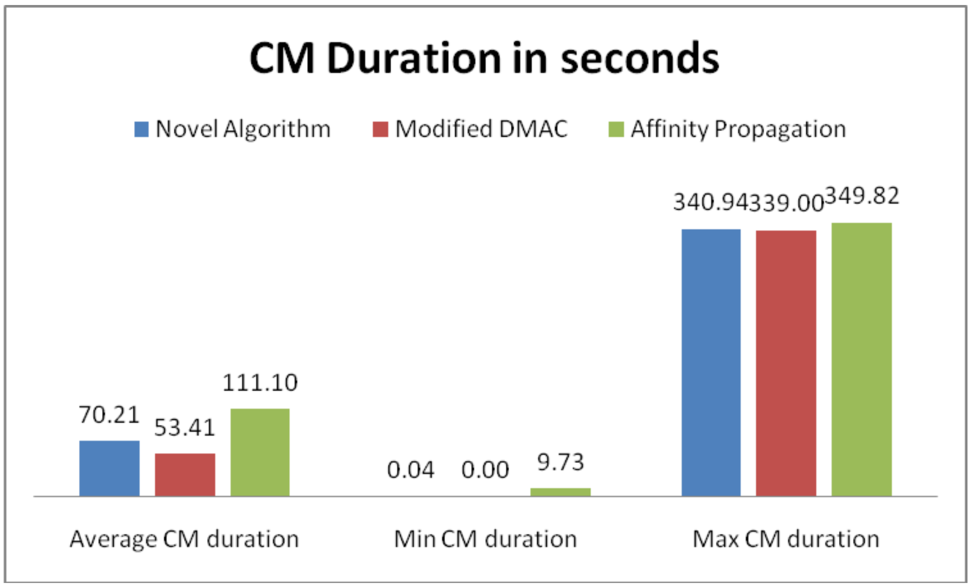


Figure 6-20: CM duration in seconds

7. CONCLUSIONS

In this paper, we present the main characteristics of VANETs. We address the problem of the communication efficiency due to the ever increasing number of devices reside in vehicles and the increased communication overhead due to the fast mobility changes in such type of networks. Clustering is a promising technique to tackle those issues and reduce the communication overhead in VANETs by dividing the current flat network topology into a hierarchical structure. After an extensive study of the clustering schemes for VANETs in the literature, we presented the implementation details of three state of the art approaches. We also provided an open source implementation of those clustering algorithms in NS-3 simulation environment which is available in a public repository. SUMO traffic simulator also used to create an urban and a rural environment to run and test the implemented algorithms.

After the evaluation of the simulations we have some important conclusions. The clustering scheme that was affected more from the different topology scenarios was MDMAC, while APROVE and Novel algorithms seem more robust to the different mobility traces tested. We highlight MDMAC algorithm for its simplicity which leads to fast formation of clusters and with a relative low number of messages in any environment. On the other side, MDMAC shows weakness compared to the other schemes in cluster stability performing great number of cluster changes during the simulations especially in the urban mobility scenario. APROVE, shows great performance in cluster stability achieving the highest duration of CH and CMs, while its average delay for the cluster formation is relatively low. A disadvantage of this scheme is the number of messages overhead. Except from the high formation delay compared to the other two implementations, Novel algorithm is dominant in the cluster stability metrics using the lowest number of messages but it reveals weakness in the cluster formation delay especially in an urban environment.

After this extensive study in VANETs we have a good perception of the limitations and weakness of the current clustering schemes found in the literature and thus our plans for the future are highly related to the proposal of a new clustering scheme. We are also interested in providing network assistance to such clustering schemes from a cellular network infrastructure in order to use part of network knowledge to the vehicle devices.

ACRONYMS

APROVE	Affinity PROpagation for VEhicular networks
CCH	Control CHannel
CH	Cluster Head
CM	Cluster Member
COV	Cluster Originating Vehicle
DMAC	Distributed and Mobility-Adaptive Clustering
DSRC	Dedicated Short Range Communication
EDCA	Enhanced Distributed Channel Access
FSM	Finite State Machine
GPLv2	General Public License version 2
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
LET	Link Expiration Time
LTE	Long Term Evolution
MAC	Medium Access Control
MANET	Mobile Ad-hoc NETwork
MDMAC	Modified Distributed and Mobility-Adaptive Clustering
NS-2	Network Simulator 2
NS-3	Network Simulator 3
OFDM	Orthogonal Frequency Division Modulation
RSU	Road-Side Unit
RT	Remaining Time
SCH	Service CHannel
SDN	Software Defined Radio
SUMO	Simulation of Urban Mobility
TDMA	Time Division Multiple Access
TTL	Time To Live
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
VANET	Vehicle Ad-hoc NETwork
VM	Virtual Machine
WAVE	Wireless Access in Vehicular Environments

ANNEX I

In this part of the document we provide a fully working application example for each one of the algorithms implemented in NS-3 environment written in C++ programming language.

Novel Algorithm Example Code

```
int main(int argc, char *argv[]) {

    /*--------------------- Logging System Configuration -------------------*/
    LogLevel logLevel = (LogLevel) (LOG_PREFIX_ALL | LOG_LEVEL_WARN);
    LogComponentEnable("V2vNovelAlgorithmExample", logLevel);
    LogComponentEnable("V2vNovelAlgorithmClient", logLevel);
    //LogComponentEnable ("Ns2MobilityHelper",logLevel);

    std::string exampleName ("V2vNovelAlgorithmClient");

    NS_LOG_UNCOND("/----------------------------------------------------------------------\\");
    NS_LOG_UNCOND(" - A novel algorithm for VANETs [Example] -> Cluster vehicles communication");
    NS_LOG_UNCOND("\\----------------------------------------------------------------------/");
    /*-----------------------------------------------------------------*/

    /*--------------------- Simulation Default Values ---------------------*/
    std::string phyMode ("OfdmRate6MbpsBW10MHz");

    uint16_t numberOfUes = 10;

    double clusterTimeMetric = 1.0;          /// Clustering Time Metric for Waiting Time calculation
    double minimumTdmaSlot = 0.01;       /// Time difference between 2 transmissions

    double simTime = 750.0;
    double trainingPeriod = 80.0;


    std::string traceFile;
    std::string logFile;
```

```
std::string range;
std::string scenario;
double gain;
double power;
/*--------------------------------------------------------------------*/


/*-------------------- Set explicitly default values -------------------*/
Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold",
                    StringValue ("2200"));
// turn off RTS/CTS for frames below 2200 bytes
Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold",
                    StringValue ("2200"));
// Fix non-unicast data rate to be the same as that of unicast
Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                    StringValue (phyMode));
/*--------------------------------------------------------------------*/


/*-------------------- Command Line Argument Values --------------------*/
CommandLine cmd;
cmd.AddValue("ueNumber", "Number of UE", numberOfUes);
cmd.AddValue("simTime", "Simulation Time in Seconds", simTime);
cmd.AddValue("range", "Transmission range of the vehicles", range);
cmd.AddValue("training", "Training period of Time", trainingPeriod);


cmd.AddValue ("traceFile", "Ns3 movement trace file", traceFile);
cmd.AddValue ("logFile", "Log file", logFile);
cmd.Parse(argc, argv);

NS_LOG_INFO("");
NS_LOG_INFO("|---"<< " SimTime -> " << simTime <<" ---|\n");
NS_LOG_INFO("|---"<< " Number of UE -> " << numberOfUes <<" ---|\n");
NS_LOG_INFO("|---"<< " Transmission Range -> " << range <<" ---|\n");
```

```
  if (traceFile.empty () || numberOfUes <= 0 || simTime <= 0 || logFile.empty ())
    {
      std::cout << "Usage of " << argv[0] << " :\n\n"
      "./waf --run \"ns2-mobility-trace"
      " --traceFile=src/mobility/examples/default.ns_movements"
      " --nodeNum=2 --duration=100.0 --logFile=ns2-mob.log\" \n\n"
      "NOTE: ns2-traces-file could be an absolute or relative path. You could use the file default.ns_movements\n"
      "      included in the same directory of this example file.\n\n"
      "NOTE 2: Number of nodes present in the trace file must match with the command line argument and must\n"
      "         be a positive number. Note that you must know it before to be able to load it.\n\n"
      "NOTE 3: Duration must be a positive number. Note that you must know it before to be able to load it.\n\n";

      return 0;
    }

  if(strcasecmp ((char*)range.c_str (), "High") == 0){
      NS_LOG_INFO("High transimssion range set.");
      power = 32;
      gain = 12;
  }
  else if(strcasecmp ((char*)range.c_str (), "Medium") == 0){
      NS_LOG_INFO("Medium transimssion range set.");
      power = 28;
      gain = 9;
  }
  else if(strcasecmp ((char*)range.c_str (), "Low") == 0){
      NS_LOG_INFO("Low transimssion range set.");
      power = 24;
      gain = 6;
  }
  else{
      std::cout << "Invalid transmission range. High/Medium/Low are supported options.";
      return 0;
```

```
}

if(traceFile.find ("Scenario1") != string::npos){
   scenario = "Scenario1";
}
else if(traceFile.find ("Scenario2") != string::npos){
   scenario = "Scenario2";
}
else if(traceFile.find ("Scenario3") != string::npos){
   scenario = "Scenario3";
}
else if(traceFile.find ("KarradaIn") != string::npos){
   scenario = "KarradaIn";
}
else if(traceFile.find ("Batavia") != string::npos){
   scenario = "Batavia";
}
else if(traceFile.find ("Manhattan") != string::npos){
   scenario = "Manhattan";
}
else{
   std::cout << "Invalid Scenario given in tracefile";
   return 0;
}

if(trainingPeriod < 0){
   std::cout << "Training period could not be negative";
   return 0;
}


// Create Ns2MobilityHelper with the specified trace log file as parameter
Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);

// open log file for output
std::ofstream os;
os.open (logFile.c_str ());
```

```
/*-------------------------------------------------------------------*/


Ipv4AddressHelper ipv4h;
ipv4h.SetBase("1.0.0.0", "255.0.0.0");
/*-------------------------------------------------------------------*/


/*------------------------ Create UEs-EnodeBs ------------------------*/
NodeContainer dummyNode;
dummyNode.Create(1);
NodeContainer ueNodes;
ueNodes.Create(numberOfUes);

ns2.Install (ueNodes.Begin (), ueNodes.End ());
/*-------------------------------------------------------------------*/


/*------------------ Install the IP stack on the UEs ------------------*/
InternetStackHelper internet;
internet.Install(ueNodes);
/*-------------------------------------------------------------------*/


/*------------------------- Setup Wifi nodes -------------------------*/
// The below set of helpers will help us to put together the wifi NICs we want
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
Ptr<YansWifiChannel> channel = wifiChannel.Create ();

YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
wifiPhy.SetChannel (channel);
wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11);
wifiPhy.Set ("TxPowerStart", DoubleValue(power));
wifiPhy.Set ("TxPowerEnd", DoubleValue(power));
wifiPhy.Set ("TxPowerLevels", UintegerValue(1));
wifiPhy.Set ("TxGain", DoubleValue(gain));
wifiPhy.Set ("RxGain", DoubleValue(gain));
```

```
   wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue(-71.8));
   wifiPhy.Set ("CcaMode1Threshold", DoubleValue(-74.8));

   NqosWaveMacHelper wifi80211pMac = NqosWaveMacHelper::Default ();
   Wifi80211pHelper wifi80211p = Wifi80211pHelper::Default ();
   //wifi80211p.EnableLogComponents ();

   wifi80211p.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                                "DataMode",StringValue (phyMode),
                                "ControlMode",StringValue (phyMode));
   NetDeviceContainer wifiDevices = wifi80211p.Install (wifiPhy, wifi80211pMac,
ueNodes);

   NS_LOG_INFO ("Assign IP Addresses.");
   ipv4h.SetBase ("10.1.1.0", "255.255.255.0");
   ipv4h.Assign (wifiDevices);

   uint16_t controlPort = 3999;
   ApplicationContainer controlApps;

   /**
    * Setting Control Channel
    */
   for (uint32_t u = 0; u < ueNodes.GetN(); ++u) {

      //!< Initial TDMA UE synchronization Function
      double vehicleTdmaSlot = (u+1)*minimumTdmaSlot;

      Ptr<MobilityModel>         mobilityModel         =         ueNodes.Get(u)-
>GetObject<MobilityModel>();
      V2vNovelAlgorithmHelper                    ueClient("ns3::UdpSocketFactory",
Address(InetSocketAddress(Ipv4Address::GetBroadcast(), controlPort)),
              "ns3::UdpSocketFactory",InetSocketAddress(Ipv4Address::GetAny(),
controlPort), mobilityModel);

      ueClient.SetAttribute ("TrainingPeriod", DoubleValue(trainingPeriod));
      ueClient.SetAttribute ("MaxUes", UintegerValue(numberOfUes));
      ueClient.SetAttribute ("MinimumTdmaSlot", DoubleValue(minimumTdmaSlot));
```

```
        ueClient.SetAttribute ("VehicleTdmaSlot", DoubleValue(vehicleTdmaSlot));
        ueClient.SetAttribute ("ClusterTimeMetric", DoubleValue(clusterTimeMetric));
        controlApps.Add(ueClient.Install(ueNodes.Get(u)));
    }

    controlApps.Start (Seconds(0.));
    controlApps.Stop (Seconds(simTime-0.1));

    AsciiTraceHelper ascii;
    wifiPhy.EnableAsciiAll(ascii.CreateFileStream ("src/v2v/examples/output/socket-
options-ipv4.txt"));
    wifiPhy.EnablePcapAll ("src/v2v/examples/output/socket.pcap", false);
    /*--------------------------------------------------------------------*/

    string numberofClustersFile = "NumberOfClusters[" + exampleName + scenario +
range + "].txt";
    string datasetContext = "Dataset/Context/String";

    Ptr<FileAggregator> aggregator1 = CreateObject<FileAggregator>
(numberofClustersFile, FileAggregator::FORMATTED);
    aggregator1->Set2dFormat ("%.3e\t%.0f");

    Simulator::Schedule(Seconds(simTime), printNumberofClusterMembers, ueNodes);
    Simulator::Schedule(Seconds(simTime), printFormationDelay, ueNodes);
    Simulator::Schedule(Seconds(simTime), printClusterChanges, ueNodes);
    Simulator::Schedule(Seconds(simTime), printNumberofMessages, ueNodes,
simTime, trainingPeriod);
    Simulator::Schedule(Seconds(trainingPeriod), calculateClustersNumber, ueNodes,
aggregator1, datasetContext, trainingPeriod);

    string numberofMessagesPerSecondFile = "NumberOfMessagesPerSecond[" +
exampleName + scenario + range + "].txt";
    Ptr<FileAggregator> aggregator2 = CreateObject<FileAggregator>
(numberofMessagesPerSecondFile, FileAggregator::FORMATTED);
    aggregator2->Set2dFormat ("%.3e\t%.0f");
    Simulator::Schedule(Seconds(trainingPeriod),
calculateNumberofMessagesPerSecond, ueNodes, aggregator2, datasetContext,
trainingPeriod);
```

```
    string clusterChangesPerSecondFile = "ClusterChangesPerSecond[" + exampleName + scenario + range + "].txt";
    Ptr<FileAggregator> aggregator3 = CreateObject<FileAggregator> (clusterChangesPerSecondFile, FileAggregator::FORMATTED);
    aggregator3->Set2dFormat ("%.3e\t%.0f");
    Simulator::Schedule(Seconds(trainingPeriod), calculateClusterChangesPerSecond, ueNodes, aggregator3, datasetContext, trainingPeriod);

    Simulator::Schedule(Seconds(simTime), printChDuration, ueNodes);
    Simulator::Schedule(Seconds(simTime), printCmDuration, ueNodes);
    Simulator::Schedule(Seconds(simTime), printMinChDuration, ueNodes);
    Simulator::Schedule(Seconds(simTime), printMaxChDuration, ueNodes);
    Simulator::Schedule(Seconds(simTime), printMinCmDuration, ueNodes);
    Simulator::Schedule(Seconds(simTime), printMaxCmDuration, ueNodes);

    /*---------------------- Simulation Stopping Time ----------------------*/
    Simulator::Stop(SIMULATION_TIME_FORMAT(simTime));
    /*--------------------------------------------------------------------*/

    /*--------------------------- Simulation Run ---------------------------*/
    Simulator::Run();
    Simulator::Destroy();
    /*--------------------------------------------------------------------*/

    return EXIT_SUCCESS;
}
```

Modified DMAC Algorithm Example Code

```
int main(int argc, char *argv[]) {

    /*--------------------- Logging System Configuration -------------------*/
    LogLevel logLevel = (LogLevel) (LOG_PREFIX_ALL | LOG_LEVEL_WARN);
    LogComponentEnable("V2vModifiedDMACExample", logLevel);
    LogComponentEnable("V2vModifiedDMACAlgorithmClient", logLevel);
    //LogComponentEnable ("Ns2MobilityHelper",logLevel);

    std::string exampleName ("V2vModifiedDMACAlgorithmClient");
```

```
   NS_LOG_UNCOND("/---------------------------------------------------------------------------
\\");
   NS_LOG_UNCOND(" - Modified DMAC Algorithm for VANETs [Example] -> Cluster
vehicles communication");
   NS_LOG_UNCOND("\\---------------------------------------------------------------------------
/");
   /*------------------------------------------------------------------*/

   /*--------------------- Simulation Default Values --------------------*/
   std::string phyMode ("OfdmRate6MbpsBW10MHz");

   uint16_t ttl = 1;
   uint16_t numberOfUes = 10;

   double beats = 1.0;
   double rangeEstimation = 200.0;
   double minimumTdmaSlot = 0.001;        /// Time difference between 2 transmissions

   double simTime = 750.0;
   double trainingPeriod = 80.0;

   std::string traceFile;
   std::string logFile;
   std::string range;
   std::string scenario;
   double gain;
   double power;
   /*------------------------------------------------------------------*/


   /*------------------- Set explicitly default values ------------------*/
   Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold",
                   StringValue ("2200"));
   // turn off RTS/CTS for frames below 2200 bytes
   Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold",
                   StringValue ("2200"));
   // Fix non-unicast data rate to be the same as that of unicast
```

```
Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                    StringValue (phyMode));
/*-------------------------------------------------------------------*/



/*------------------ Command Line Argument Values ------------------*/
CommandLine cmd;
cmd.AddValue("ueNumber", "Number of UE", numberOfUes);
cmd.AddValue("simTime", "Simulation Time in Seconds", simTime);
cmd.AddValue("range", "Transmission range of the vehicles", range);
cmd.AddValue("training", "Training period of Time", trainingPeriod);

cmd.AddValue ("traceFile", "Ns3 movement trace file", traceFile);
cmd.AddValue ("logFile", "Log file", logFile);
cmd.Parse(argc, argv);

NS_LOG_INFO("");
NS_LOG_INFO("|---"<< " SimTime -> " << simTime <<" ---|\n");
NS_LOG_INFO("|---"<< " Number of UE -> " << numberOfUes <<" ---|\n");
NS_LOG_INFO("|---"<< " Transmission Range -> " << range <<" ---|\n");



if (traceFile.empty () || numberOfUes <= 0 || simTime <= 0 || logFile.empty ())
  {
    std::cout << "Usage of " << argv[0] << " :\n\n"
    "./waf --run \"ns2-mobility-trace"
    " --traceFile=src/mobility/examples/default.ns_movements"
    " --nodeNum=2 --duration=100.0 --logFile=ns2-mob.log\" \n\n"
    "NOTE: ns2-traces-file could be an absolute or relative path. You could use the file default.ns_movements\n"
    "      included in the same directory of this example file.\n\n"
    "NOTE 2: Number of nodes present in the trace file must match with the command line argument and must\n"
    "        be a positive number. Note that you must know it before to be able to load it.\n\n"
    "NOTE 3: Duration must be a positive number. Note that you must know it before to be able to load it.\n\n";
```

```
        return 0;
      }

    if(strcasecmp ((char*)range.c_str (), "High") == 0){
        NS_LOG_INFO("High transimssion range set.");
        power = 32;
        gain = 12;
    }
    else if(strcasecmp ((char*)range.c_str (), "Medium") == 0){
        NS_LOG_INFO("Medium transimssion range set.");
        power = 28;
        gain = 9;
    }
    else if(strcasecmp ((char*)range.c_str (), "Low") == 0){
        NS_LOG_INFO("Low transimssion range set.");
        power = 24;
        gain = 6;
    }
    else{
        std::cout << "Invalid transmission range. High/Medium/Low are supported
options.";
        return 0;
    }

    if(traceFile.find ("Scenario1") != string::npos){
        scenario = "Scenario1";
    }
    else if(traceFile.find ("Scenario2") != string::npos){
        scenario = "Scenario2";
    }
    else if(traceFile.find ("Scenario3") != string::npos){
        scenario = "Scenario3";
    }
    else if(traceFile.find ("KarradaIn") != string::npos){
        scenario = "KarradaIn";
    }
    else if(traceFile.find ("Batavia") != string::npos){
```

```
    scenario = "Batavia";
  }
  else if(traceFile.find ("Manhattan") != string::npos){
    scenario = "Manhattan";
  }
  else{
    std::cout << "Invalid Scenario given in tracefile";
    return 0;
  }


  // Create Ns2MobilityHelper with the specified trace log file as parameter
  Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);

  // open log file for output
  std::ofstream os;
  os.open (logFile.c_str ());
  /*-----------------------------------------------------------------*/


  Ipv4AddressHelper ipv4h;
  ipv4h.SetBase("1.0.0.0", "255.0.0.0");
  /*-----------------------------------------------------------------*/


  /*------------------------ Create UEs-EnodeBs ------------------------*/
  NodeContainer dummyNode;
  dummyNode.Create(1);
  NodeContainer ueNodes;
  ueNodes.Create(numberOfUes);

  ns2.Install (ueNodes.Begin (), ueNodes.End ());
  /*-----------------------------------------------------------------*/


  /*------------------- Install the IP stack on the UEs ------------------*/
  InternetStackHelper internet;
```

```
    internet.Install(ueNodes);
    /*---------------------------------------------------------------------*/



    /*------------------------- Setup Wifi nodes -------------------------*/
    // The below set of helpers will help us to put together the wifi NICs we want
    YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
    Ptr<YansWifiChannel> channel = wifiChannel.Create ();

    YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
    wifiPhy.SetChannel (channel);
    wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11);
    wifiPhy.Set ("TxPowerStart", DoubleValue(power));
    wifiPhy.Set ("TxPowerEnd", DoubleValue(power));
    wifiPhy.Set ("TxPowerLevels", UintegerValue(1));
    wifiPhy.Set ("TxGain", DoubleValue(gain));
    wifiPhy.Set ("RxGain", DoubleValue(gain));
    wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue(-71.8));
    wifiPhy.Set ("CcaMode1Threshold", DoubleValue(-74.8));

    NqosWaveMacHelper wifi80211pMac = NqosWaveMacHelper::Default ();
    Wifi80211pHelper wifi80211p = Wifi80211pHelper::Default ();
    //wifi80211p.EnableLogComponents ();

    wifi80211p.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                              "DataMode",StringValue (phyMode),
                              "ControlMode",StringValue (phyMode));
    NetDeviceContainer wifiDevices = wifi80211p.Install (wifiPhy, wifi80211pMac,
ueNodes);

    NS_LOG_INFO ("Assign IP Addresses.");
    ipv4h.SetBase ("10.1.0.0", "255.255.0.0");
    ipv4h.Assign (wifiDevices);

    uint16_t controlPort = 3999;
    ApplicationContainer controlApps;
```

```
  /**
   * Setting Modified DMAC Client
   */
  for (uint32_t u = 0; u < ueNodes.GetN(); ++u) {

      //!< Initial TDMA UE synchronization Function
      double vehicleTdmaSlot = (u+1)*minimumTdmaSlot;

      Ptr<MobilityModel>         mobilityModel         =         ueNodes.Get(u)-
>GetObject<MobilityModel>();
      V2vModifiedDMACAlgorithmHelper          ueClient("ns3::UdpSocketFactory",
Address(InetSocketAddress(Ipv4Address::GetBroadcast(), controlPort)),
              "ns3::UdpSocketFactory",InetSocketAddress(Ipv4Address::GetAny(),
controlPort), mobilityModel);

      ueClient.SetAttribute ("TrainingPeriod", DoubleValue(trainingPeriod));
      ueClient.SetAttribute ("TTL", UintegerValue(ttl));
      ueClient.SetAttribute ("Beats", DoubleValue(beats));
      ueClient.SetAttribute ("Range", DoubleValue(rangeEstimation));
      ueClient.SetAttribute ("Freshness", DoubleValue(10*beats));
      ueClient.SetAttribute ("MaxUes", UintegerValue(numberOfUes));
      ueClient.SetAttribute ("MinimumTdmaSlot", DoubleValue(minimumTdmaSlot));
      ueClient.SetAttribute ("VehicleTdmaSlot", DoubleValue(vehicleTdmaSlot));
      controlApps.Add(ueClient.Install(ueNodes.Get(u)));
  }

  controlApps.Start (Seconds(0.));
  controlApps.Stop (Seconds(simTime-0.1));

  AsciiTraceHelper ascii;
  wifiPhy.EnableAsciiAll(ascii.CreateFileStream     ("src/v2v/examples/output/socket-
options-ipv4.txt"));
  wifiPhy.EnablePcapAll ("src/v2v/examples/output/socket.pcap", false);
  /*-------------------------------------------------------------------*/

  string numberofClustersFile     = "NumberOfClusters[" + exampleName + scenario +
range + "].txt";
  string datasetContext = "Dataset/Context/String";
```

```
   Ptr<FileAggregator> aggregator1 = CreateObject<FileAggregator> (numberofClustersFile, FileAggregator::FORMATTED);
   aggregator1->Set2dFormat ("%.3e\t%.0f");

   Simulator::Schedule(Seconds(simTime), printNumberofClusterMembers, ueNodes);
   Simulator::Schedule(Seconds(simTime), printFormationDelay, ueNodes);
   Simulator::Schedule(Seconds(simTime), printClusterChanges, ueNodes);
   Simulator::Schedule(Seconds(simTime), printNumberofMessages, ueNodes, simTime, trainingPeriod);
   Simulator::Schedule(Seconds(trainingPeriod), calculateClustersNumber, ueNodes, aggregator1, datasetContext, trainingPeriod);

   string numberofMessagesPerSecondFile = "NumberOfMessagesPerSecond[" + exampleName + scenario + range + "].txt";
   Ptr<FileAggregator> aggregator2 = CreateObject<FileAggregator> (numberofMessagesPerSecondFile, FileAggregator::FORMATTED);
   aggregator2->Set2dFormat ("%.3e\t%.0f");
   Simulator::Schedule(Seconds(trainingPeriod), calculateNumberofMessagesPerSecond, ueNodes, aggregator2, datasetContext, trainingPeriod);

   string clusterChangesPerSecondFile = "ClusterChangesPerSecond[" + exampleName + scenario + range + "].txt";
   Ptr<FileAggregator> aggregator3 = CreateObject<FileAggregator> (clusterChangesPerSecondFile, FileAggregator::FORMATTED);
   aggregator3->Set2dFormat ("%.3e\t%.0f");
   Simulator::Schedule(Seconds(trainingPeriod), calculateClusterChangesPerSecond, ueNodes, aggregator3, datasetContext, trainingPeriod);

   Simulator::Schedule(Seconds(simTime), printChDuration, ueNodes);
   Simulator::Schedule(Seconds(simTime), printCmDuration, ueNodes);
   Simulator::Schedule(Seconds(simTime), printMinChDuration, ueNodes);
   Simulator::Schedule(Seconds(simTime), printMaxChDuration, ueNodes);
   Simulator::Schedule(Seconds(simTime), printMinCmDuration, ueNodes);
   Simulator::Schedule(Seconds(simTime), printMaxCmDuration, ueNodes);

   /*--------------------- Simulation Stopping Time ----------------------*/
   Simulator::Stop(SIMULATION_TIME_FORMAT(simTime));
   /*-------------------------------------------------------------------*/
```

```
    /*--------------------------- Simulation Run ---------------------------*/
    Simulator::Run();
    Simulator::Destroy();
    /*----------------------------------------------------------------------*/

    return EXIT_SUCCESS;
}
```

APROVE Algorithm Example Code

```
int main(int argc, char *argv[]) {

    /*-------------------- Logging System Configuration -------------------*/
    LogLevel logLevel = (LogLevel) (LOG_PREFIX_ALL | LOG_LEVEL_WARN);
    LogComponentEnable("V2vAffinityExample", logLevel);
    LogComponentEnable("V2vAffinityAlgorithmClient", logLevel);
    //LogComponentEnable ("Ns2MobilityHelper",logLevel);

    std::string exampleName ("V2vAffinityAlgorithmClient");

    NS_LOG_UNCOND("/------------------------------------------------------------------------------
-----\\");
    NS_LOG_UNCOND(" - Affinity Propagation Algorithm for VANETs [Example] ->
Cluster vehicles communication");
    NS_LOG_UNCOND("\\------------------------------------------------------------------------------
------/");
    /*----------------------------------------------------------------------*/

    /*--------------------- Simulation Default Values ---------------------*/
    std::string phyMode ("OfdmRate6MbpsBW10MHz");

    uint16_t numberOfUes = 10;

    double lamda = 0.5;
    uint32_t CIperiod = 10;
    double minimumTdmaSlot = 0.001;        /// Time difference between 2 transmissions
    double positionTimeWindow = 1.0;
    double simTime = 750.0;
    double trainingPeriod = 80.0;
    double selfSimilarity = -150.0;

    std::string traceFile;
    std::string logFile;
    std::string range;
    std::string scenario;
    double gain;
    double power;
    /*-------------------------------------------------------------------*/


    /*------------------- Set explicitly default values ------------------*/
    Config::SetDefault ("ns3::WifiRemoteStationManager::FragmentationThreshold",
                        StringValue ("2200"));
    // turn off RTS/CTS for frames below 2200 bytes
    Config::SetDefault ("ns3::WifiRemoteStationManager::RtsCtsThreshold",
                        StringValue ("2200"));
    // Fix non-unicast data rate to be the same as that of unicast
    Config::SetDefault ("ns3::WifiRemoteStationManager::NonUnicastMode",
                        StringValue (phyMode));
    /*-------------------------------------------------------------------*/


    /*------------------- Command Line Argument Values -------------------*/
    CommandLine cmd;
    cmd.AddValue("ueNumber", "Number of UE", numberOfUes);
    cmd.AddValue("simTime", "Simulation Time in Seconds", simTime);
    cmd.AddValue("range", "Transmission range of the vehicles", range);
    cmd.AddValue("CIperiod", "Cluster Interval Period", CIperiod);
    cmd.AddValue("training", "Training period of Time", trainingPeriod);

    cmd.AddValue ("traceFile", "Ns3 movement trace file", traceFile);
    cmd.AddValue ("logFile", "Log file", logFile);
    cmd.Parse(argc, argv);
```

```
  NS_LOG_INFO("");
  NS_LOG_INFO("|---"<< " SimTime -> " << simTime <<" ---|\n");
  NS_LOG_INFO("|---"<< " Number of UE -> " << numberOfUes <<" ---|\n");
  NS_LOG_INFO("|---"<< " Transmission Range -> " << range <<" ---|\n");


  if (traceFile.empty () || numberOfUes <= 0 || simTime <= 0 || logFile.empty ())
   {
     std::cout << "Usage of " << argv[0] << " :\n\n"
     "./waf --run \"ns2-mobility-trace"
     " --traceFile=src/mobility/examples/default.ns_movements"
     " --nodeNum=2 --duration=100.0 --logFile=ns2-mob.log\" \n\n"
     "NOTE: ns2-traces-file could be an absolute or relative path. You could use the file
default.ns_movements\n"
     "      included in the same directory of this example file.\n\n"
     "NOTE 2: Number of nodes present in the trace file must match with the command
line argument and must\n"
     "         be a positive number. Note that you must know it before to be able to load
it.\n\n"
     "NOTE 3: Duration must be a positive number. Note that you must know it before
to be able to load it.\n\n";

     return 0;
   }

  if(strcasecmp ((char*)range.c_str (), "High") == 0){
     NS_LOG_INFO("High transimssion range set.");
     power = 32;
     gain = 12;
  }
  else if(strcasecmp ((char*)range.c_str (), "Medium") == 0){
     NS_LOG_INFO("Medium transimssion range set.");
     power = 28;
     gain = 9;
  }
  else if(strcasecmp ((char*)range.c_str (), "Low") == 0){
     NS_LOG_INFO("Low transimssion range set.");
```

```
        power = 24;
        gain = 6;
    }
    else{
        std::cout << "Invalid transmission range. High/Medium/Low are supported options.";
        return 0;
    }

    if(traceFile.find ("Scenario1") != string::npos){
        scenario = "Scenario1";
    }
    else if(traceFile.find ("Scenario2") != string::npos){
        scenario = "Scenario2";
    }
    else if(traceFile.find ("Scenario3") != string::npos){
        scenario = "Scenario3";
    }
    else if(traceFile.find ("KarradaIn") != string::npos){
        scenario = "KarradaIn";
    }
    else if(traceFile.find ("Batavia") != string::npos){
        scenario = "Batavia";
    }
    else if(traceFile.find ("Manhattan") != string::npos){
        scenario = "Manhattan";
    }
    else{
        std::cout << "Invalid Scenario given in tracefile";
        return 0;
    }

    if(trainingPeriod < 0){
        std::cout << "Training period could not be negative";
        return 0;
    }
```

```
// Create Ns2MobilityHelper with the specified trace log file as parameter
Ns2MobilityHelper ns2 = Ns2MobilityHelper (traceFile);

// open log file for output
std::ofstream os;
os.open (logFile.c_str ());
/*--------------------------------------------------------------------*/


Ipv4AddressHelper ipv4h;
ipv4h.SetBase("1.0.0.0", "255.0.0.0");
/*--------------------------------------------------------------------*/


/*------------------------ Create UEs-EnodeBs ------------------------*/
NodeContainer dummyNode;
dummyNode.Create(1);
NodeContainer ueNodes;
ueNodes.Create(numberOfUes);

ns2.Install (ueNodes.Begin (), ueNodes.End ());
/*--------------------------------------------------------------------*/


/*------------------ Install the IP stack on the UEs -----------------*/
InternetStackHelper internet;
internet.Install(ueNodes);
/*--------------------------------------------------------------------*/


/*------------------------- Setup Wifi nodes -------------------------*/
// The below set of helpers will help us to put together the wifi NICs we want
YansWifiChannelHelper wifiChannel = YansWifiChannelHelper::Default ();
Ptr<YansWifiChannel> channel = wifiChannel.Create ();

YansWifiPhyHelper wifiPhy =  YansWifiPhyHelper::Default ();
```

```
  wifiPhy.SetChannel (channel);
  wifiPhy.SetPcapDataLinkType (YansWifiPhyHelper::DLT_IEEE802_11);
  wifiPhy.Set ("TxPowerStart", DoubleValue(power));
  wifiPhy.Set ("TxPowerEnd", DoubleValue(power));
  wifiPhy.Set ("TxPowerLevels", UintegerValue(1));
  wifiPhy.Set ("TxGain", DoubleValue(gain));
  wifiPhy.Set ("RxGain", DoubleValue(gain));
  wifiPhy.Set ("EnergyDetectionThreshold", DoubleValue(-71.8));
  wifiPhy.Set ("CcaMode1Threshold", DoubleValue(-74.8));

  NqosWaveMacHelper wifi80211pMac = NqosWaveMacHelper::Default ();
  Wifi80211pHelper wifi80211p = Wifi80211pHelper::Default ();
  //wifi80211p.EnableLogComponents ();

  wifi80211p.SetRemoteStationManager ("ns3::ConstantRateWifiManager",
                          "DataMode",StringValue (phyMode),
                          "ControlMode",StringValue (phyMode));
  NetDeviceContainer wifiDevices = wifi80211p.Install (wifiPhy, wifi80211pMac,
ueNodes);

  NS_LOG_INFO ("Assign IP Addresses.");
  ipv4h.SetBase ("10.1.1.0", "255.255.255.0");
  ipv4h.Assign (wifiDevices);

  uint16_t controlPort = 3999;
  ApplicationContainer controlApps;

  /**
   * Setting Affinity Client
   */
  for (uint32_t u = 0; u < ueNodes.GetN(); ++u) {

    //!< Initial TDMA UE synchronization Function
    double vehicleTdmaSlot = (u+1)*minimumTdmaSlot;

    Ptr<MobilityModel> mobilityModel = ueNodes.Get(u)-
>GetObject<MobilityModel>();
```

```
      V2vAffinityAlgorithmHelper                              ueClient("ns3::UdpSocketFactory",
Address(InetSocketAddress(Ipv4Address::GetBroadcast(), controlPort)),
              "ns3::UdpSocketFactory",InetSocketAddress(Ipv4Address::GetAny(),
controlPort), mobilityModel);

      ueClient.SetAttribute ("TrainingPeriod", DoubleValue(trainingPeriod));
      ueClient.SetAttribute ("SelfSimilarity", DoubleValue(selfSimilarity));
      ueClient.SetAttribute ("CI", UintegerValue(CIperiod));
      ueClient.SetAttribute ("Tf", DoubleValue(positionTimeWindow));
      ueClient.SetAttribute ("Lamda", DoubleValue(lamda));
      ueClient.SetAttribute ("MaxUes", UintegerValue(numberOfUes));
      ueClient.SetAttribute ("MinimumTdmaSlot", DoubleValue(minimumTdmaSlot));
      ueClient.SetAttribute ("VehicleTdmaSlot", DoubleValue(vehicleTdmaSlot));
      controlApps.Add(ueClient.Install(ueNodes.Get(u)));
   }

   controlApps.Start (Seconds(0.));
   controlApps.Stop (Seconds(simTime-0.1));

   AsciiTraceHelper ascii;
   wifiPhy.EnableAsciiAll(ascii.CreateFileStream        ("src/v2v/examples/output/socket-
options-ipv4.txt"));
   wifiPhy.EnablePcapAll ("src/v2v/examples/output/socket.pcap", false);
   /*------------------------------------------------------------------*/

   string numberofClustersFile       = "NumberOfClusters[" + exampleName + scenario +
range + "].txt";
   string datasetContext = "Dataset/Context/String";

   Ptr<FileAggregator>        aggregator1        =        CreateObject<FileAggregator>
(numberofClustersFile, FileAggregator::FORMATTED);
   aggregator1->Set2dFormat ("%.3e\t%.0f");

   Simulator::Schedule(Seconds(simTime), printNumberofClusterMembers, ueNodes);
   Simulator::Schedule(Seconds(simTime), printFormationDelay, ueNodes);
   Simulator::Schedule(Seconds(simTime), printClusterChanges, ueNodes);
   Simulator::Schedule(Seconds(simTime),      printNumberofMessages,      ueNodes,
simTime, trainingPeriod);
```

```
    Simulator::Schedule(Seconds(trainingPeriod), calculateClustersNumber, ueNodes, aggregator1, datasetContext, trainingPeriod);


    string numberofMessagesPerSecondFile = "NumberOfMessagesPerSecond[" + exampleName + scenario + range + "].txt";
    Ptr<FileAggregator> aggregator2 = CreateObject<FileAggregator> (numberofMessagesPerSecondFile, FileAggregator::FORMATTED);
    aggregator2->Set2dFormat ("%.3e\t%.0f");
    Simulator::Schedule(Seconds(trainingPeriod), calculateNumberofMessagesPerSecond, ueNodes, aggregator2, datasetContext, trainingPeriod);


    string clusterChangesPerSecondFile = "ClusterChangesPerSecond[" + exampleName + scenario + range + "].txt";
    Ptr<FileAggregator> aggregator3 = CreateObject<FileAggregator> (clusterChangesPerSecondFile, FileAggregator::FORMATTED);
    aggregator3->Set2dFormat ("%.3e\t%.0f");
    Simulator::Schedule(Seconds(trainingPeriod), calculateClusterChangesPerSecond, ueNodes, aggregator3, datasetContext, trainingPeriod);


    Simulator::Schedule(Seconds(simTime), printChDuration, ueNodes);
    Simulator::Schedule(Seconds(simTime), printCmDuration, ueNodes);
    Simulator::Schedule(Seconds(simTime), printMinChDuration, ueNodes);
    Simulator::Schedule(Seconds(simTime), printMaxChDuration, ueNodes);
    Simulator::Schedule(Seconds(simTime), printMinCmDuration, ueNodes);
    Simulator::Schedule(Seconds(simTime), printMaxCmDuration, ueNodes);


    /*---------------------- Simulation Stopping Time ----------------------*/
    Simulator::Stop(SIMULATION_TIME_FORMAT(simTime));
    /*--------------------------------------------------------------------*/


    /*-------------------------- Simulation Run --------------------------*/
    Simulator::Run();
    Simulator::Destroy();
    /*--------------------------------------------------------------------*/


    return EXIT_SUCCESS;
}
```

ANNEX II

In this part of the document we provide the steps on how to reproduce a SUMO topology similar to the topology we used for the evaluation of the algorithms. Prerequisites in order to run the following commands are only a LINUX based machine and a sumo 0.23 version as described in section **Error! Reference source not found.**.

- Step 1: Download a topology of your interest from the openstreetmap.
- Step 2: The downloaded .osm file is used as parameter to the netconvert tool in order to produce the net.xml SUMO file using the following command.

```
netconvert --osm-files Manhattan.osm -o Manhattan.net.xml
```

- Step 3: Invoke polyconvert tool by providing both .osm file and net.xml file produced in the previous step and a typemap file, a template of which is provided directly from SUMO community. Polyconvert imports geometrical shapes (polygons or points of interest) from different sources, converts them to a representation that may be visualized using SUMO-GUI.

```
polyconvert --net-file Manhattan.net.xml --osm-files Manhattan.osm --type-
file typemap.xml -o Manhattan.poly.xml
```

- Step 4: Use python script to generate random vehicle trips into the topology we have produced. An example of this command is shown in the following text box.

```
python randomTrips.py -n Manhattan.net.xml -r Manhattan.rou.xml -e 55.0 -p
0.15 --trip-attributes="departLane=\"best\" departSpeed=\"max\"
departPos=\"free\"" --intermediate 4 -l
```

- Step 5: Invoke sumo-gui to see the topology created and the vehicles moving. The file given as parameter a template of which is also provided from SUMO, just invokes the 3 files we have produced.

```
sumo-gui -c Manhattan.sumo.cfg
```

- Step 6: Invoke sumo tool to import net.xml topology and vehicles routing into one file.

```
sumo -n Manhattan.net.xml -r Manhattan.rou.xml --fcd-output
Manhattan.xml
```

- Step 7: Use a python script file in order to convert the SUMO topology into a format compatible to NS-3 simulator. The output file is finally used from NS-3 parser to feed the simulator with vehicles traces.

```
traceExporter.py --fcd-input=Manhattan.xml --ns2mobility-output=mobility.tcl
```

REFERENCES

[1] L. Katsikas, K. Chatzikokolakis and N. Alonistioti, Implementing clustering for vehicular ad-hoc networks in ns-3, *Proceedings* of the 2015 Workshop on ns-3, May 2015, pp.25-31.
[2] L. Miao, k. Djouani, B. J. van Wyk and Y. Hamam, Evaluation and Enhancement of IEEE 802.11p Standard: A Survey, Mobile Computing, Vol. 1, Iss. 1, November 2012.
[3] Z. Y. Rawashdeh. A novel algorithm to form stable clusters in vehicular ad hoc networks on highways. EURASIP Journal on Wireless Communications and Networking, January 2012.
[4] B. Hassanabadi, C. Shea, L. Zhang and S. Valaee. Clustering in Vehicular Ad Hoc Networks using Affinity Propagation. Ad Hoc Networks, February 2014, 13(B), 535-548.
[5] E. Sakhaee and A. Jamalipour. Stable Clustering and Communications in Pseudolinear Highly Mobile Ad Hoc Networks. Vehicular Technology, IEEE Transactions, November 2008, 57(6), 3769-3777.
[6] W. Su, S. J. Lee and M. Gerla. Mobility prediction in wireless networks. MILCOM 2000, 21st Century Military Communications Conference Proceedings, 1, 491-495.
[7] W. Fan, Y. Shi, S. Chen and L. Zou. A mobility metrics based dynamic clustering algorithm for VANETs. Communication Technology and Application (ICCTA 2011), IET International Conference, October 2011, 752- 756.
[8] F. Bai, N. Sadagopan and A. Helmy. IMPORTANT: a framework to systematically analyze the Impact of Mobility on Performance of Routing Protocols for Adhoc Networks. INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies, April 2003, 2, 825-835.
[9] N. Maslekar, M. Boussedjra, J. Mouzna and L. Houda. Direction based clustering algorithm for data dissemination in vehicular networks. Vehicular Networking Conference (VNC), October 2009, 1-6.
[10] G. Wolny. Modified DMAC Clustering Algorithm for VANETs. Systems and Networks Communications. ICSNC '08. 3rd International Conference, October 2008, 268-273.
[11] S. Basagni. Distributed clustering for ad hoc networks. Parallel Architectures, Algorithms, and Networks. (I-SPAN '99) Proceedings. Fourth International Symposium, June 1999, 310-315.
[12] S. Kukliński and G. Wolny. Density based clustering algorithm for VANETs. Testbeds and Research Infrastructures for the Development of Networks & Communities and Workshops. TridentCom 2009. 5th International Conference, April 2009, 1-6.
[13] A. Daeinabi, A. G. P. Rahbar and A. Khademzadeh. VWCA: An efficient clustering algorithm in vehicular ad hoc networks. Journal of Network and Computer Applications, January 2011, 34(1), 207-222.
[14] R. T. Goonewardene, F.H. Ali and E. Stipidis. Robust mobility adaptive clustering scheme with support for geographic routing for vehicular ad hoc networks. Intelligent Transport Systems, June 2009, 3(2), 148-158.
[15] E. Dror, C. Avin and Z. Lotker. Fast randomized algorithm for hierarchical clustering in Vehicular Ad-Hoc Networks. Ad Hoc Networking Workshop (Med-Hoc-Net), The 10th IFIP Annual Mediterranean, June 2011,1-8.
[16] M.S. Almalag and M. C. Weigle. Using traffic flow for cluster formation in vehicular ad-hoc networks. Local Computer Networks (LCN), IEEE 35th Conference, October 2010, 631-636.
[17] Network Simulator-3 (NS-3), https://www.nsnam.org/.
[18] Simulation of Urban Mobility (SUMO), http://www.dlr.de/.
[19] An instant Virtual Network on your Laptop Mininet, http://mininet.org/.
[20] M. Behrisch, L. Bieker, J. Erdmann and D. Krajzewicz. SUMO-Simulation of Urban Mobility An Overview. The Third International Conference on Advances in System Simulation, 2011.