Dynamic Clustering Protocol based on Relative Speed in Mobile Ad Hoc Networks For Intelligent Vehicles

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Sundeep Gopalaswamy

Certificate of Approval:

David Umphress
Associate Professor
Computer Science and
Software Engineering

Alvin S. Lim, Chair
Associate Professor
Computer Science and
Software Engineering

Min-Te Sun
Assistant Professor
Computer Science and
Software Engineering

George T. Flowers
Interim Dean
Graduate School

Dynamic Clustering Protocol based on Relative Speed in Mobile Ad Hoc Networks For Intelligent Vehicles

Sundeep Gopalaswamy

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
December 17, 2007

Dynamic Clustering Protocol based on Relative Speed in Mobile Ad Hoc Networks For Intelligent Vehicles

Sundeep Gopalaswamy

Signature of Author

Date of Graduation

Vita

Sundeep Gopalaswamy, son of P.S.Gopalaswamy and P.K.Vijayalakshmi was born June 19, 1981, in Bangalore, India. He earned his Bachelor's degree in Computer Science from UVCE, Bangalore University, Bangalore, India in 2003. He worked for nearly two years in MindTree Consulting as Software Engineer.

Thesis Abstract

Dynamic Clustering Protocol based on Relative Speed in Mobile Ad Hoc Networks For Intelligent Vehicles

Sundeep Gopalaswamy

Master of Science, December 17, 2007
(B.E., UVCE, Bangalore University, 2003)

86 Typed Pages

Directed by Alvin S. Lim

A knowledge of real time traffic density on different roads has many applications such as real time navigation for driver, designing efficient vehicular routing protocol and building fully autonomous vehicles. Forming clusters of vehicle is the first step towards achieving these goals. Once clusters are formed, distributed servers could be built which would collect and store all the information. Querying the distributed servers would give density information at various roads. We propose a clustering protocol, RSDCP which adapts dynamically to high mobility of vehicles. Our protocol is able to form stable clusters by choosing the cluster head based on relative speed. To demonstrate this, we compare it with two other protocols, one which has same clustering mechanism as that of our protocol but is based on id instead of relative speed (IDDCP) and the second is a simple clustering protocol (IDS) where cluster head is elected periodically based on id, similar to the one proposed by Gerla et al. [1]. We did the analysis from three different perspectives, in terms of time, clustering, and network packets. For evaluating in terms of time, time spent as part of cluster was measured. From clustering perspective, number of clusters, and number

of vehicles per cluster were measured. To estimate the network performance, number of protocol packets, and application packets transmitted were measured. Results show that in RSDCP, vehicles are part of cluster 30% longer than IDS and nearly 5% longer than IDDCP. RSDCP on almost all the test scenarios has higher average number of clusters as well as higher number of vehicles per cluster. Comparing the overall packet overhead, we notice that IDS has nearly 50% higher overhead than RSDCP while IDDCP has nearly 10% higher overhead than RSDCP.

Acknowledgments

I would like to express my appreciation to Dr. Alvin Lim for the guidance he has provided throughout my study at Auburn. I would also like to express my gratitude to the advisory committee members, Dr. David Umphress and Dr. Min-Te Sun.

I would like to thank the Dr. Prathima Agrawal and the Wireless Engineering Board for its support through the Vodafone Fellowship. This fellowship was instrumental in supporting my research.

I would also like to thank my fellow students (especially Raghukisore Neelisetti) who have helped me in understanding the problems and finding feasible solutions.

Above all, I would like to thank my parents who though needed me the most let me come to Auburn to achieve this dream of mine. I would also like to thank my sisters and brother-in-laws whose moral support mattered the most.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliograpy follows van Leunen's *A Handbook for Scholars*.

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures

Chapter 1

Introduction

Starting from carriages of olden days to present day supersonic aeroplanes, man has made numerous inventions to make his travel comfortable and easy. Among these, roadways are the most widely used means of transportation. The Transportation Statistics Annual Report [2] published by Bureau of Transportation Statistics presents many statistics which highlight the importance of roadways in a comman man's life in the United States. If laid end to end, roads in United States would circle the earth 160 times. In 2001, the average number of vehicles per household was 1.90, higher than average licensed drivers per household, 1.75. In 2005, 88.4 percent of the workforce used cars to drive to their work everyday. According to the 2005 Urban Mobility Report [3], in 2003 congestion caused 3.7 billion hours of travel delay and 2.3 billion gallons of wasted fuel, an increase of 79 million hours and 69 million gallons from 2002 to a total cost of more than 63 billion dollars. All these indicate that there is a need for better design of vehicles and road infrastructure. In order to achieve this, the concept of Intelligent Vehicles was envisioned by pioneers like Bishop and Hahn.

Bishop and Hahn have described various problems in Intelligent Vehicles and the methods adopted to solve them. In [4], Hahn predicts the roadmap of Intelligent Vehicles as shown in Figure 1.1. It can be seen that initially the focus is on developing mechanisms to help the driver. Forward collision warning, lane departure warning, headway control, automatic braking, obstacle warning, drowsiness warning system, and nighttime pedestrian warning are some of the research which falls under this category. Many companies such as

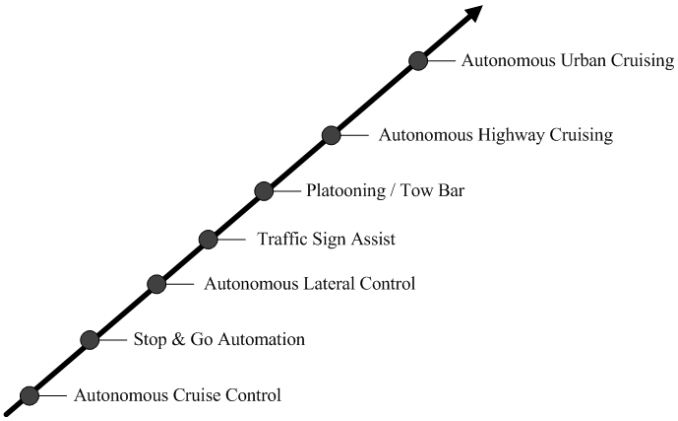


Figure 1.1: Intelligent Vehicles Roadmap [4]

Honda, Nissan, and Toyota are making efforts to construct reliable mechanical control system to have a safe driving experience [5]. As these mature, vehicles can start co-operating with their neighbors by exchanging mobility parameters and driver intentions. The ultimate goal is to allow vehicles to function autonomously. The University of Korea, the Mechanical Engineering Laboratory in Japan, the University of Pavia (Italy), Ohio State University and the University of California are making attempts to develop fully automated cars [5].

It is envisioned that the vehicles would communicate with each other using wireless technologies [6]. In order to exchange information with the neighbors, good protocol design is needed. If every vehicle starts to broadcast, there would be packet collisions leading to loss of information. Instead, vehicles could form clusters and establish a protocol to exchange information in an orderly manner. The concept of clustering has existed for a long time. Clustering helps in efficient management of nodes, be it with respect to addressing scheme, routing or load balancing [7]. Clustering also leads to a hierarchical organization so that the system becomes managable even when scaled to large numbers. In our present work, we propose a Dynamic Clustering Protocol based on Relative Speed, RSDCP which is suitable

for Intelligent Vehicles. We show that using this clustering algorithm, density information at various places can be collected. RSDCP forms stable clusters thereby making it easy for the application to retrieve the density information.

In Chapter 2, we discuss the motivations for our work and describe its relevant applications in Intelligent Vehicles. We give an overview of research related to our clustering protocol in Chapter 3. In Chapter 4 we discuss the design principles that guided the development of our protocol and describe it in detail in Chapter 5. Chapters 6 and 7 explain the simulation and performance evaluation of the clustering protocol and a simple application running on it. We conclude our research and discuss future work in Chapter 8.

Chapter 2

Motivations and Applications

In this chapter we discuss the motivation for proposing this new scheme of clustering as well as its potential applications.

2.1 Motivation

Many clustering protocols [1] [7] [8] [9] [10] [11] have been proposed in the past. Only some of them [8] [9] [10] [11] accounted for mobility of the nodes. These clustering protocols were proposed for MANETs where mobility is slow and non-continuous. In a traffic scenarios, the vehicles are in constant motion and at times reach high speeds of 70 miles per hour (or 31.11 meters per second). An algorithm catering to these needs should give high importance to mobility and should have less overhead to achieve real time performance. Two vehicles could be in the communication range of one another but could be travelling in opposite directions. In order to give a candid picture of the traffic pattern, it is required to distinguish vehicles moving in the same direction from vehicles moving in the opposite direction. None of the protocols mentioned previously have this "sense of direction". Once nodes are clustered and a cluster head is elected, density information about the cluster can be collected at regular intervals by cluster head. Querying the cluster head would give the density information of the area it covers.

2.2 Application

Forming clusters is the first step towards building a distributed service which would give density information of vehicles at various places. The potential applications of this distributed service are explained in the subsequent sections. The layer in the network stack which incorporates this distributed service is called Distributed Service layer. Its position in the network stack is as shown in Figure 2.1. It can be seen in the figure that the Distributed Service layer can be used by network layer as well as application layer. In the architecture

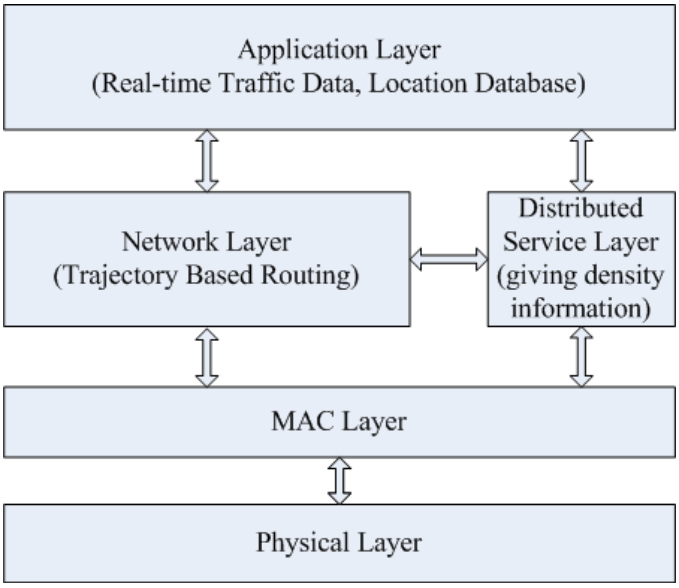


Figure 2.1: Distributed Service Layer in the network stack

of the distributed service there are several distributed servers called Distributed Density Servers (DDS) which hold information about density at various places. In order to realize this architecture (see Figure 2.2), the following problems have to be addressed:

1. Forming Clusters: Clusters have to be formed with a cluster head which collects data from all other vehicles and stores them temporarily.

2. Choosing DDS: Some of the cluster heads could become DDS which would store all the information of a given area. An algorithm must be designed that distributively decides certain cluster heads to take up the role of DDS.

3. Exchanging Information among DDS: Cluster heads would have partial data. These have to be aggregated and stored in different DDS. A mechanism has to be designed to achieve this.

4. Designing Query Protocol: When a vehicle needs density information of a particular area, it has to send a query to the DDS. DDS would respond back with the desired information. The challenge is that the querying vehicle would have moved after transmitting the query. A protocol has to be designed such that the query and reponse works efficiently as well as reliably.

The present work attempts to solve the first problem. The rest of the problems are typical to a Location based services or Moving Object database. In the past there has been much research [12] [13] [14] [15] which has proposed frameworks to build a moving database. Using these as starting points, the above mentioned problems can be addressed. Venturing into the domain of Mobile agents could provide solutions for third problem. Likewise, [16] which attempts to address the problems associated with querying these databases, can be used to solve the fourth problem.

2.2.1 Routing in Vehicular Ad hoc Networks

Vehicular Ad hoc Networks have many applications [17]. For example, it could be used to query the nearest department stores which have specific products for sale. It could be used to find the nearest parking lot with an empty parking space. To realize these applications,

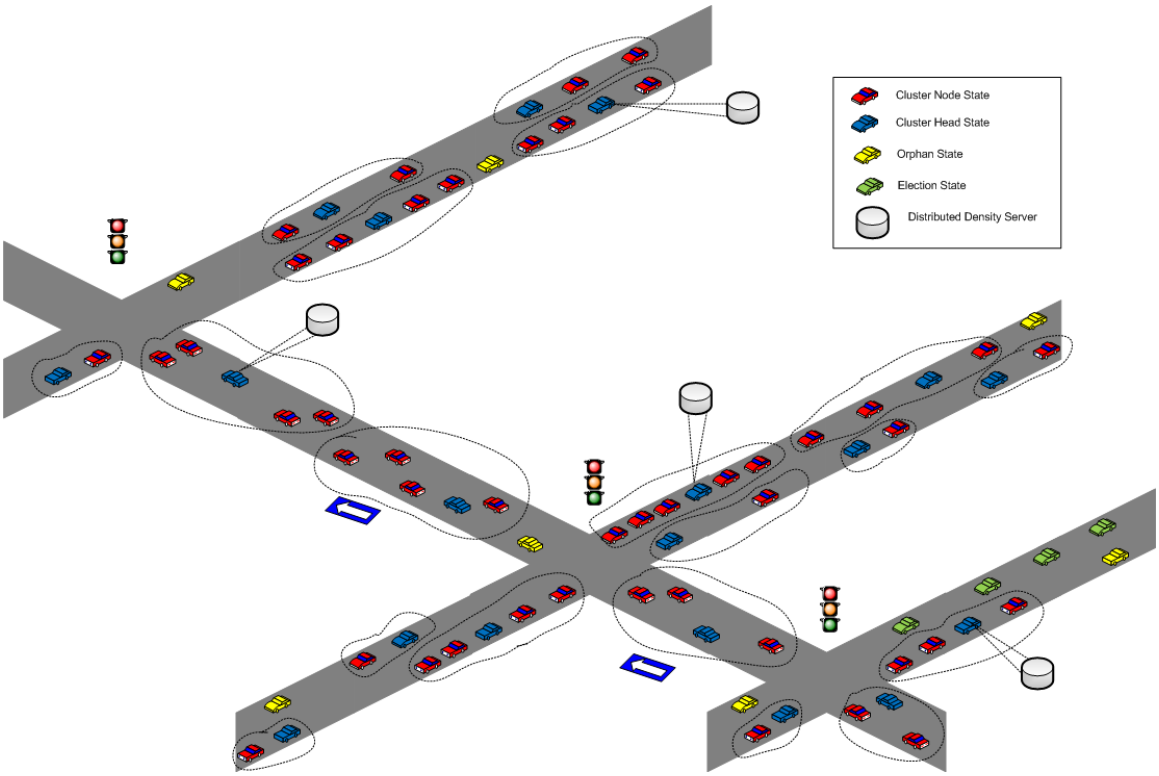


Figure 2.2: Distributed Density Servers

query packets would be sent by vehicles which has to routed to the destination and the reply has to routed back to the vehicles. Routing in vehicular ad hoc network have some unique problems compared to MANET. In a vehicular ad hoc network, vehicles travel through definitive paths (i.e streets). While routing packets, there could be scenarios where the shortest paths may be sparsely populated with vehicles as shown in Figure 2.3. In such scenarios, it would be better to route through longer paths with higher density of vehicles for increased reliability and decreased delay.

Routing protocols which address these problems have been proposed in [18] and [17]. In [18], Trajectory Based Forwarding (TBF) was proposed for routing packets. TBF specifies

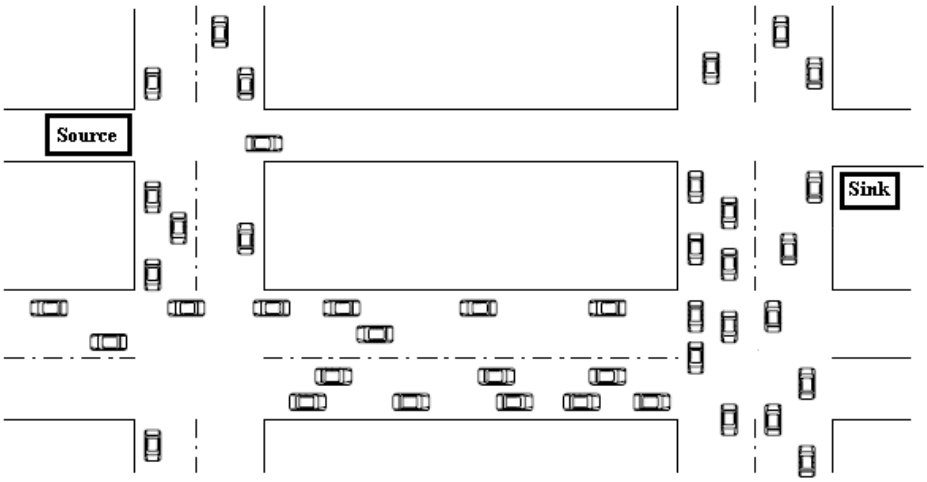


Figure 2.3: Unique scenario seen in Vehicular Ad hoc Network [17]

the routing path in the form of parametric equations. By combining the proposed architecture with TBF, a better routing algorithm could be devised. The proposed architecture would give the information about density of vehicles at various places and their average speed. Using this information, a source can choose the denser path and encode the trajectory in the packet. In [17], Vehicle-assisted Data Delivery (VADD) protocol was proposed. In [17], VADD is shown to outperform conventional routing protocols such as DSR and GPSR. The main assumption of VADD is that density information will be preloaded in the vehicle. This has certain drawbacks. For example if there is an accident, then the density and average speed of the vehicles in that road would be different from the preloaded one. Instead, using on-the-fly density information got from the proposed architecture would lead to more informed routing decision thereby achieving higher delivery ratio and lower delay.

2.2.2 Collecting Real-time Traffic Information

The density of vehicles varies with time and day of the week. In case of an accident or an emergency which blocks the roads, the traffic pattern would vary drastically from the usual pattern observed. A real-time knowledge of density and average speed of vehicles in various roads can help a driver to navigate through the shortest time route considering the present traffic conditions. Currently, websites such as Google [19], Yahoo [20], Microsoft [21] are offering services which attempts to give the real-time traffic information. Usually data is collected from road sensors and taxi fleets and sent to a centralized server [22]. In the server, data is analyzed and results are made available in a simple form through web access. Google Maps [19] can only be run on certain mobiles [23]. In scenarios where the driver has a mobile capable of running Google Maps, data could be received while driving also. There are certain drawbacks of the current system. Firstly, these services are available only in selected major cities of the United States. Secondly, these services are available on the internet only. The driver has to look up on the internet before starting the trip and by the time he or she starts to travel, the information could have changed. Instead, if a mobile is used to access the internet and get the information, it increases the load on cellular infrastructure. Then, cellular service providers have to constantly upgrade their infrastructure. Thirdly, traffic data collected is first sent to a server where it is analyzed and then made available on the internet. This information could be old as compared to on-the-fly query and reply. In the proposed architecture all these problems could be solved.

2.2.3 Adding other services

By using the proposed clustering protocol and the distributed service architecture, other services could also be added. For example using this methodology, a location service could be built. In [24] [25] [26], frameworks are proposed to build location servers. In the proposed architecture, cluster heads could collect location information from its cluster nodes and send it to distributed servers where it could be maintained. Whenever the source wants location information of the sink, then it can query these servers.

2.2.4 Building fully automated vehicles

As indicated in Figure 1.1, the ultimate goal of Intelligent Vehicles is to build fully automated vehicles. This can be achieved when vehicles co-operate with each other. The concept of cooperative driving was first presented by JSK (Japans Association of Electronic Technology for Automobile Traffic and Driving) in the early 1990s [6]. Since then, research has been carried out in feasibility studies such as California's PATH project, the European Unions Chauffeur project, and Japans Demo 2000 Cooperative Driving System [6]. In order to achieve co-operation, vehicles should exchange their status information with other vehicles. Using the framework of the distributed service architecture, status information of the vehicles could be exchanged to build a fully autonomous vehicle. Autonomous vehicles not only reduces burden of the driver but also increases the efficiency of fuel as there would be lesser delays in the traffic.

Chapter 3

Related Work

3.1 Lowest ID and Highest Connectivity based clustering

Gerla et.al [1] proposed one of the foremost work on clustering in ad hoc networks. It aimed at clustering wireless stations deployed for emergency disaster relief or battlefield communication. Two schemes were proposed by Gerla et.al. The first scheme was based on lowest node id, and the second scheme was based on higest connectivity (degree). In both the schemes, all the nodes periodically broadcast the list of nodes they can hear. In the first scheme, the node with lowest id in its neighborhood becomes the cluster head. In the second scheme, the node with highest number of neighbors becomes the cluster head and in case of a tie, the node with lowest id becomes cluster head. The proposed schemes were simple but not suitable for scenarios where nodes are mobile.

3.2 MOBIC

In [8], MOBIC, a clustering algorithm based on the mobility metric was proposed for MANETs. In this work, it is assumed that power level detected at the receiving node is indicative of the distance between transmitting and receiving node pairs. At the start of the clustering process, each node sends two "hello" messages to its neighboring nodes to determine whether they are moving towards or away from it. The relative mobility metric for a node Y with respect to node X is defined as,

$$M_Y^{rev}(X) = 10log_{10}(\frac{RxPr_{X\rightarrow Y}^{new}}{RxPr_{X\rightarrow Y}^{old}}) \tag{3.1}$$

where,

$RxPr^{new}_{X \to Y}$ refers to the power level received in the second hello message

$RxPr^{old}_{X \to Y}$ refers to the power level received in the first hello message

If $RxPr^{new}_{X \to Y}$ is less than $RxPr^{old}_{X \to Y}$, then $M^{rel}_{Y}(X)$ will be negative indicating that X is moving away from Y. On the other hand, if $RxPr^{new}_{X \to Y}$ is greater than $RxPr^{old}_{X \to Y}$, then $M^{rel}_{Y}(X)$ will be positive indicating that X is moving towards Y. After calculating the pairwise relative mobility metric for each neighboring node, the aggregate relative mobility metric is calculated as a variance of the pairwise relative mobility metric as follows,

$$M_Y = var_0(M^{rel}_{Y}(X_1), M^{rel}_{Y}(X_2), M^{rel}_{Y}(X_3), \ldots, M^{rel}_{Y}(X_n)) \tag{3.2}$$

A low aggregate mobility metric (M) is desired as it indicates that the node is relatively stationary with respect to its neighboring nodes.

The clustering procedure starts by first sending two hello messages as stated above and calculating the value of M. All the nodes then broadcast their M values to their one-hop neighbors every "Broadcast_Interval" period. If a node has the lowest value of M among all its neighbors, then it changes the status to cluster head, otherwise it declares itself as cluster member. In case more than one node is potential cluster head, then the node id is used to resolve the conflict. Movement of cluster members to different clusters does not trigger reclustering.

In MOBIC, difference in the power level received between successive hello messages is used to determine if a node was moving towards or away from each other. In the paper, it is also mentioned that power level could be affected by various external factors such as tree, buildings and landscape. Hence, in traffic scenarios where external factors change rapidly

with place and time, using MOBIC may not be viable. In our proposed protocol we do not make any decision based on power levels.

3.3 TMPO

In [9], the Topology Management by Priority Ordering (TMPO) algorithm is used to form clusters in order to build a backbone infrastructure. Since TMPO is designed for ad hoc networks, energy becomes an important criterion. In TMPO, nodes communicate information about all their one-hop neighbors to their neighbors. This information is used to calculate the priority of each node in the two-hop neighborhood. A node then makes a decision to become cluster head either if it has the highest priority in its one-hop neighborhood or if it has the highest priority in the one-hop neighborhood of one of its one-hop neigbhors. The node priorities are calculated based on three components, node id, present time, and a "Willingness" value. The Willingness value for a node i is defined as a function of energy level and speed of the node as follows,

$$W_i = 2^{log_2(E_i * c_1) log_2(s_i + c_2)} \tag{3.3}$$

where,

W_i is the Willingness value,

E_i is the remaining energy in the range $[0, 1)$,

s_i is the speed in meters per second,

c_1 and c_2 are constants with values 0.9 and 2 respectively. These constants are used to eliminate boundary conditions in the logarithmic operations.

The logarithmic operation on the speed and remaining energy ensures that the willingness value is high for high energy and low speed while it is close to zero for low energy and high speed.

In order to distribute the role of cluster head fairly among nodes, the priorities of nodes change periodically, triggering re-election of cluster head. The priority of each node is recomputed asynchronously so as to avoid sudden loss of the old network states. The time to recompute priority for a node is determined as follows,

$$t = kT + \lfloor Hash\,(i)\,.T \rfloor \tag{3.4}$$

where,

t is the current time,

i is the node id,

$k = 0, 1, 2 \ldots$,

T is the priority recomputation period,

$Hash$ is a pseudo-random number generator that produces a uniformly distributed random number over range $[0, 1)$.

Since all the nodes would have all the information about its one and two hop neighbors, each node determines locally the priority of its neighbors. The priority of a node is calculated as follows,

$$i.prio = Hash\,(k \oplus i)\,.W_i \oplus i \tag{3.5}$$

where,

$i, k, Hash$ are the same as used in Equation 3.4,

W_i is the Willingness value calculated from Equation 3.3.

$\oplus$ sign is designated to carry out the bit-concatenation operation on its operands and has lower order than other operations.

After doing the above calculation if a node determines that it is the cluster head it can take up the role because its neighbors also are running the same algorithm using the same information.

TMPO is based on the assumption that all the nodes will be time synchronized which is not possible in all scenarios. The Willingness metric in TMPO makes use of absolute speed of the node. This is a sub-optimal metric as there could be scenarios where nodes are travelling at high speeds in the same direction. Though they could form stable clusters, using absolute speed would reflect that they are unstable. In RSDCP we do not require time synchronization. RSDCP is based on relative speed which is a better metric.

3.4 (p, t, d)-clustering model

In [10], a (p, t, d)-clustering model is proposed for MANETs to support Quality of Service (QoS). The model is based on intelligent mobility prediction that enables each node to anticipate the availability of its neighbors. In order to implement the protocol, the concept of virtual clusters is introduced. In virtual clusters, a geographical area under consideration is divided into equal regions of circular shape such that given location information each node can determine the virtual cluster it belongs to. Each virtual cluster has a unique identifier which is based on its geographic location. Each node is supposed to have a complete picture of the locations of virtual clusters. Once a node has the information regarding

virtual clusters it can compute its residence time in each virtual cluster. This is achieved by making use of a Mobility Prediction Model.

Every user tends to have favorite routes and habitual movement patterns. These factors are exploited by the Mobility Prediction Model. This model, motivated by computational learning theory, attempts to derive a probabilistic prediction of a particular node by utilizing its accumulated movement history. In this model, each node is responsible for generating and constructing a multiway tree or Mobility trie [27] in real time, depending on its movement and time. This in turn will be used to predict the residence time in each virtual cluster.

The clustering algorithm proposed in [10] is as follows. The cluster head broadcasts every CH_HELLO_INTERVAL a "hello" message which has the virtual cluster id, virtual cluster center, cluster radius, and neighbor-table (the set of cluster members). When a node enters a virtual cluster and receives a hello message, then it returns a JOIN message which includes its residence time within the current virtual cluster, its location information, and its Mobility Trie corresponding to the next T_{mt} minutes. T_{mt} is a system parameter which should be set to an optimal value. When a cluster head receives the JOIN message relevant to its virtual cluster, it appends the node's information into the neighbor-table. The node should wait for two successive CH_HELLO_INTERVAL before retransmitting JOIN message. The cluster head also exchange hello message among themselves. When a cluster head realizes that it will become unavailable, it triggers "CH changeover event" exactly t_{ce} seconds before it predicts to leave the virtual cluster. All the cluster members

then calculate their Ω values as follows,

$$\begin{aligned} &if \quad d_{xk}(t) \neq 0, \quad \Omega_x = p_{xk}\left(\frac{t_{xk}-t_{th}}{d_x k(t)}\right) \\ &otherwise \quad \Omega_x = p_{xk}\left(\frac{t_{xk}-t_{th}}{d_{min}}\right) \end{aligned} \tag{3.6}$$

where,

p_{xk} is the state probability of node x stays in virtual cluster k,

t_{xk} is the residence time of node x in virtual cluster k,

t_{th} is the threshold residence time,

$d_{xk}(t)$ is the distance between node x to the center of the virtual cluster,

d_{min} is the minimum value that $d_{xk}(t)$ can take.

Each node then sends its Ω values to the cluster head. Based on the information received by the cluster head, a node with highest Ω value is chosen to be the next cluster head. The cluster head also decides on two assistant cluster heads for reliability purposes. The current cluster head then broadcasts this information using a SUCCESSOR message. In case the cluster head does not send the hello message for two consecutive CH_HELLO_INTERVALs, then the first assistant takes the role of cluster head. In case the first assistant fails, the second assistant takes the role of cluster head. In case the second assistant also fails, any node which first detects it sends a "CH changeover event" and acts as the temporary cluster head.

(p, t, d)-clustering model is based on the assumption that the nodes will have a comprehensive knowledge of the area it operates in. This is a challenging task in scenarios where there are multiple product vendors and users operating in different areas. In case of vehicles

this gets complicated as they are mass produced and sold in different parts of the globe. In RSDCP, no such pre-loaded knowledge is required.

3.5 EMAC

In [11], Efficient Management Algorithm for Clustering (EMAC) was proposed for building an architecture based on clusters. Each cluster is of fixed size. Each node is identified by a state, $N_i(id_{node}, id_{CH}, Weight, Counter, N)$ where id_{node} is node identifier, id_{CH} is cluster head identifier, $Weight$ is a parameter used to indicate the suitability of a node for playing cluster head role, $Counter$ is the number of nodes presently in the cluster and N is the maximum number of nodes a cluster can have. The $Weight$ parameter is defined as follows,

$$W_i = a * \Delta_i + b * E_i + c * P_i + d * S_i \tag{3.7}$$

where,

a, b, c and d are the weighing factors and $a + b + c + d = 1$,

$\Delta_i = |d_i - N|$ with d_i being the number of neighbors,

E_i is the remaining battery power,

P_i is the actual transmission power,

S_i is the average speed until current time.

Apart from this, each cluster head maintains a CH_table where information of other cluster heads are stored as $(id_{CH}, Weight)$. The cluster head broadcasts a hello message which contains all this information.

When a new node is activated in the network, the following protocol is followed to join a cluster. Initially, the node after scanning the channel for any activity, broadcasts

a Join_Request message to all cluster head in the communication range and waits for a Welcome_ACK or a Welcome_NACK. When the node does not receive any reply or gets only Welcome_NACK from all cluster heads, it increases its transmission power and broadcasts another Join_Request. After a certain number of such attempts, it declares itself to be an isolated node and waits for certain time period before attempting the procedure again. If the node receives a Welcome_ACK, then it waits for all cluster head responses before making a decision. The node chooses a cluster head with least weight and sends a Join_Accept message. The cluster head then adds it to its table and sends a CH_ACK message to confirm. A re-election procedure is initiated by the cluster head if it finds that the new node has a lower weight. In the re-election procedure, the cluster head calculates the weights of all the nodes. Based on the weights, the number of nodes in the cluster and the power level of the nodes, the cluster head selects which node will be the next cluster head. The cluster head then transmits all the information about the cluster to the new cluster head and notifies its member about the change using CH_Change message.

EMAC considers mobility in the range of 0.83m/s to 2.77m/s which is very low compared to vehicle speed which is in the range of 20m/s to 35m/s. Similar to TMPO, EMAC considers the average speed and not the relative speed which is a better metric. EMAC has a three way message passing (Join_Request-Welcome_ACK-Join_Accept) that consumes more time compared to two-way message passing (GET_STATS-NODE_STATS) adopted in RSDCP.

Chapter 4

Clustering Protocol Design

In this chapter we give a brief overview of the clustering protocol design and the rationale behind each design decision.

4.1 Design Overview

RSDCP is dynamic because vehicles in the cluster change frequently due to high mobility. According to RSDCP, a cluster is a collection of vehicles with a leader called "Cluster head". Clusters formed in RSDCP are non-overlapping implying that a vehicle can be part of only one cluster. The cluster head is always one hop away from all the vehicles in the cluster. But all one hop neighbors of cluster head need not be part of its cluster. That is, there could be a vehicle which is one hop neighbor of a cluster head but belongs to another cluster. A vehicle could be in any of the following states: Un-clustered, Orphan, Election, Cluster node and Cluster head. When a vehicle starts it would be in Un-clustered state. A vehicle changes to Orphan state if it does not have a cluster head in the communication range. When a group of vehicles in orphan state come within the communication range of each other they enter Election state to choose a cluster head. The vehicle with best stability factor is chosen as cluster head. Once cluster is formed, the vehicle chosen as cluster head enters the Cluster head state while all other vehicles enter Cluster node state. Movement of vehicle from one cluster to another does not trigger election. If cluster nodes are in the communication range of two or more cluster head, they chose the cluster head with highest stability factor. Un-clustered and Election states are cluster setup states since they lead

to the formation of clusters and often last for only a short time period. Orphan, Cluster Node and Cluster Head states are cluster maintainence states. Section 5.4 explains in detail various states and their respective algorithms.

4.2 Design Decisions

4.2.1 Deciding on clustering nodes

Conventionally, the cluster head chooses the nodes that are part of its cluster. In our protocol, a node decides which of the clusters it wants to join. In some clustering protocols [11], the cluster head makes the decision by a three-way message passing. Firstly, a node sends a join request to the cluster head. Secondly, the cluster head replies back accepting the join request. Thirdly, the node sends a confirmation to the cluster head to actually be added to the cluster. In our protocol, the nodes make the decision by two way-message passing. Firstly, cluster head sends hello packet indicating its presence. If a node wants to join that cluster, it replies back with its statistics. In a highly mobile scenario such as vehicular communication, the topology changes rapidly. If a three way message passing is adopted, there could be scenarios where, by the time the node sends confirmation to the cluster head, it could have gone out of communication range. This could also lead to scenarios where the time spent on joining a cluster would be more than the time spent as part of cluster. In our protocol, these problems are minimized. Also, the number of packets exchanged is reduced leading to better performance in terms of time taken to form clusters and fewer packet collisions.

4.2.2 Stability factor

Stability factor is the metric used to choose the cluster head. It is crucial for stable cluster formation. As the protocol is designed for vehicular communication where there is high mobility, stability factor should be based on speed. Taking absolute speed may not be a wise decision. For example, there could be a group of vehicle travelling at high speeds. Considering their absolute speed may reflect that they are unstable but taking relative speed would show that they are stable which is true. This approach also has certain drawbacks. Consider a scenario where there are three lanes and a group of cars are travelling with different speeds as shown in the Figure 4.1. If clustering is purely based on relative speed, then they could form clusters as indicated by the dotted line. Instead, if the number of cluster nodes is also accounted for, then it could form one large cluster as indicated by the solid line. In case of large clusters, lesser packets are required to collect information from individual vehicles as well as there would be lesser points from where the information collected has to be aggregated. RSDCP makes use of this idea and accounts for number of cluster nodes for a slight trade off with relative speed.

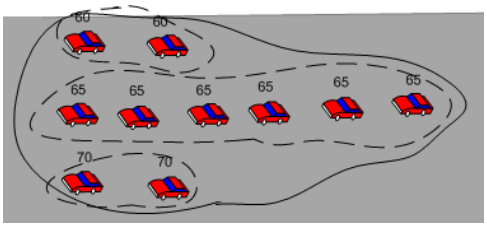


Figure 4.1: Stability factor based on relative speed and number of neighbors

The stability factor is used in two instances. The first instance is when a node decides on the cluster head to report to. When a node, x receives GET_STATS packet from cluster

head *ch*, it calculates stability factor with respect to the cluster head as follows,

$$Stability = \alpha * ch_n + \beta * (log(rel\_speed_{max}) - log(|x_{speed} - ch_{speed}| + \delta)) \tag{4.1}$$

where,

α and β are constants,

δ is a constant with very low value in order to eliminate boundary conditions in the logarithmic operation,

rel_speed_{max} is the maximum relative speed achievable,

ch_n is the number of cluster nodes reporting to the cluster head,

ch_{speed} is the speed of cluster head,

x_{speed} is the speed of cluster node.

By varying constants α and β, the contribution of the number of cluster nodes and relative speed can be changed. It is always desirable to join cluster head with more nodes as this leads to larger clusters. From the relative speed standpoint, it is always better to have low relative speed. The relative speed factor should be such that, initially with small change in relative speed, the factor should decrease by a large extent. In this way, a cluster head with lesser relative speed is chosen. If a cluster head with higher relative speed has to be chosen, then it has to make up the difference by having significantly more number of cluster nodes. Care has to be taken to ensure that a cluster head with high relative speed is not completely ignored. In order to achieve this goal, the relative speed should be a difference of log of maximum relative speed and the relative speed with the cluster head as shown in the Equation 4.1. A plot of value of relative speed factor with varying relative speed is shown in Figure 4.2.

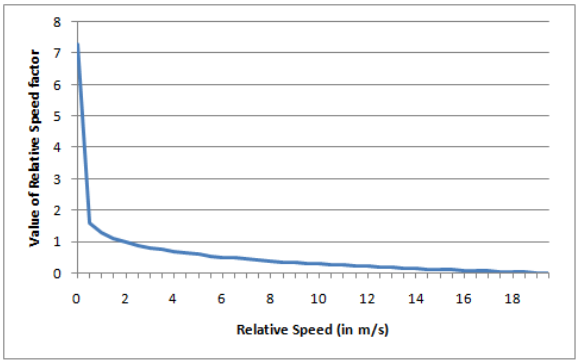


Figure 4.2: Plot of variation of Relative Speed factor with change in Relative Speed

A second instance of the stability factor usage is when vehicles distributively choose the cluster head during the election process. The stability factor should be consistent with the previous instance. One way of accomplishing this is by calculating the stability factor of each vehicle in the potential cluster head list assuming one of them is cluster head. This would lead to computation complexity of $O(n^2)$ where n is the number of vehicles in potential cluster head list. Instead, average speed can be computed and the stability of each vehicle with respect to average speed can be found. The vehicle with the highest stability can be chosen as cluster head. In this way complexity is reduced to $O(n)$. Thus a vehicle x in election state calculates the stability factor of vehicle y in its potential cluster head list as follows,

$$Stability = \alpha * y_{neigh} + \beta * (log(rel\_speed_{max}) - log(|x_{avg\_speed} - y_{speed}| + \delta)) \qquad (4.2)$$

where,

α, β, δ and rel_speed_{max} are same as in Equation 4.1,

x_{avg_speed} is the average of speed of x and speed of its neighbors,

y_{neigh} is the number of neighbors of y,

y_{speed} is speed of y.

4.2.3 Election process

Cluster formation and cluster head election can be described mathematically by the Minimum Dominating Set problem [9]. It is known that the Minimum Dominating Set problem is NP-hard [9] [11] even when the complete network topology is available. As a result we had to adopt three phases during election process. During the first phase, each vehicle discovers its neighbors. In the second phase, vehicles exchange each others information to build the potential cluster head list. Only in the third phase, vehicles decide their cluster head. Once a vehicle knows that the cluster head it has chosen has become cluster head, it is important to announce that it is not going to be cluster head to its neighbors. Failing to do so leads to a cascading effect. Consider for example the scenario in which a certain number of cars are involved in election as shown in Figure 4.3. It could

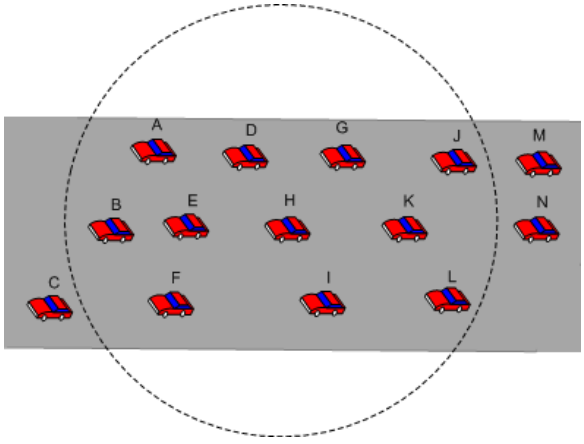


Figure 4.3: Cascading Effect example scenario

happen that C chooses E as potential cluster head and M and N chooses K as potential

cluster head. But E and K could have chosen H as potential cluster head. C, M and N are not aware of H as they are outside the communication range of H as shown in the figure. So, if E and K does not indicate they have a cluster head, C, M and N would choose them as cluster head and later realize that E and K are part of another cluster. They would have to enter election state again to choose a new cluster head.

4.2.4 Mobility

As the proposed protocol is for vehicular communication it should be able to adapt to high mobility. In this section we discuss various mobility scenarios and the methodology adopted to handle them. While a vehicle is part of cluster, it could be in cluster head state or cluster node state. In case any cluster node slows down or speeds up, the cluster head would leave the decision to the cluster node to join another cluster or continue reporting to it. In case the cluster head itself slows down or speeds up, ending up alone on the road, it would change to orphan state. In case a cluster node slows down or speeds up such that it is left alone on the road, it would switch to orphan state.

Many of the clustering algorithms consider mobility, but assume that nodes are stationary during the election process. It would be inappropriate to assume this if the nodes are vehicles. RSDCP handles this problem in an efficient way. After the first phase of election where the vehicles have exchanged hello packets, some of them may go out of communication range of each other. During second phase, the actual statistics used for election are exchanged. If a vehicle goes out of communication range during the second phase, then it would not be able to send its statistics. Besides, the first phase messages are used to find the number of neighbors which has lesser impact on the stability than the

relative speed. Vehicles could also move out of communication range of each other after phase two. This could have severe repercussions. A vehicle may choose another vehicle as cluster head which is about to move out of its communication range. In order to solve this problem, neighbors outside the tolerance range are discarded. Tolerance range is calculated as a fraction of communication range. The fraction should be such that it ensures that the vehicle stays in the communication range at least till the election ends. It can be statically set or dynamically set based on the speed of vehicles.

4.3 Alternate Design

While designing the clustering protocol, some of the alternative methods were considered. In this section we explain one of them. Given a map, the area could be divided into blocks. Vehicles from the same block forms a cluster. This is similar to the virtual cluster concept proposed in [10]. A cluster head will be elected among them and will collect density information. When the cluster head moves out of the block, it hands over the information to a newly elected vehicle. As the vehicles are highly mobile, the hand off could be more frequent. For example, if vehicles are travelling at 20 meters per second (45mph) and each block is 250 meters (i.e. communication range), then a vehicle would be part of a cluster for 13 seconds in the best case which is not desirable. If the block size is more than communication range, then communicating with cluster head could be delayed as they have to go through more than one hop. In case of discontinuity in the flow of vehicles, there could be problems passing the density information from old cluster head to the new cluster head of the block. For these reasons, the proposed protocol was chosen.

Chapter 5

Dynamic Clustering Protocol Based On Relative Speed

In this chapter we describe in detail our proposed Dynamic Clustering Protocol based on Relative Speed (RSDCP). Assumptions, the concept of "sensing direction", and packet types of RSDCP are discussed below following which the protocol is explained in detail.

5.1 Assumptions

The following assumptions were made while developing the proposed protocol,

1. Every vehicle has a unique id. This id could be generated from the electronic license plate.

2. Every vehicle is equipped with GPS or has means to find its position and speed.

3. The wireless communication is symmetrical. This means if vehicle X transmits and vehicle Y receives it, then if vehicle Y transmits, vehicle X would receive it.

5.2 Sensing Direction

The uniqueness of the proposed protocol is its ability to distinguish vehicles moving in the same direction from vehicles moving in opposite direction. In order to achieve this, each direction is given a different value as shown in Figure 5.1. The direction of vehicle is found from the change in x and y co-ordinates as shown in Table 5.1. There are certain problems which have been addressed using this scheme. Firstly, at a traffic signal, there could be more than two roads meeting. Vehicles at the traffic signal should be able to distinguish

Figure 5.1: Direction Assignment

vehicles moving in the same direction from vehicles moving in other directions. Secondly, in two or more lane roads, vehicles would change lanes. The direction of vehicle would change for few seconds during lane change. At that instance, other vehicles should not interpret it as a vehicle in a different direction. There is a tolerance of 22.5 degrees on either side to cater for this behavior (see Figure 5.1). In real world, there could be scenarios where the angle between the roads could be smaller than 45 degrees. In this case, a balance has to be struck between tolerance given for lane change and the number of directions detected.

5.3 Packet Types

Table 5.2 summarises the packet types used in the proposed clustering protocol. Direction is a field which appears in all the packets so that vehicles moving in a different direction can be distinguished from vehicles moving in the same direction. GET_STATS packet is

$\Delta x > 0$	$\Delta y \geq 0$	$0^o \leq \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 22.5^o$	$direction = 0$
$\Delta x > 0$	$\Delta y > 0$	$22.5^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 67.5^o$	$direction = 1$
$\Delta x \geq 0$	$\Delta y > 0$	$67.5^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 90^o$	$direction = 2$
$\Delta x < 0$	$\Delta y > 0$	$90^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 112.5^o$	$direction = 2$
$\Delta x < 0$	$\Delta y > 0$	$112.5^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 157.5^o$	$direction = 3$
$\Delta x < 0$	$\Delta y \geq 0$	$157.5^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 180^o$	$direction = 4$
$\Delta x < 0$	$\Delta y < 0$	$180^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 202.5^o$	$direction = 4$
$\Delta x < 0$	$\Delta y < 0$	$202.5^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 247.5^o$	$direction = 5$
$\Delta x \leq 0$	$\Delta y < 0$	$247.5^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 270^o$	$direction = 6$
$\Delta x > 0$	$\Delta y < 0$	$270^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 292.5^o$	$direction = 6$
$\Delta x > 0$	$\Delta y < 0$	$292.5^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) \leq 337.5^o$	$direction = 7$
$\Delta x > 0$	$\Delta y < 0$	$337.5^o < \arctan\left(\frac{\Delta y}{\Delta x}\right) < 360^o$	$direction = 0$

Table 5.1: Direction sensing

the hello packet broadcasted by cluster head at regular intervals. It contains the id and speed of the cluster head and the number of cluster nodes currently reporting to the cluster head. A GET_STATS packet with number of cluster nodes field set to zero is broadcasted by vehicles in the orphan state. In case the vehicles are beaconing at regular intervals as part of network routing protocol (e.g. GPSR), GET_STATS packet could be incorporated into it, to lessen protocol overhead. The NODE_STATS packet is the cluster node's reply for a GET_STATS packet. Currently it contains the cluster node's id and speed. If any further services have to be added, then the corresponding information has to be added to this packet and transmitted by the cluster node. MY_STATS is similar to GET_STATS packet except that it contains more fields, the x, and y coordinates. MY_STATS packet is broadcasted during the election state to exchange information in order to determine the cluster head in a distributed manner. MY_STATS packet can be part of beaconing packets

as well. NOT_CH packet contains only the id of the vehicle transmitting the packet. It is used to announce to the neighbors that the vehicle has chosen some other vehicle as cluster head and to remove its entry from the potential cluster head list. This prevents the cascading effect explained in Section 4.2.3.

Packet Type	State of Sender	Contents
GET_STATS (gpkt)	Cluster Head / Orphan	Node ID, Direction, Speed, Number of Cluster nodes.
NODE_STATS (npkt)	Cluster Node	Node ID, Direction, Speed.
MY_STATS (mpkt)	Election	Node ID, Direction, X, Y, Speed, Number of Neighbors.
NOT_CH (nchpkt)	Election	Node ID, Direction.

Table 5.2: Packet Types

5.4 States

In [28], clustering protocol is explained by using state interaction which is intutive and easy to implement using object oriented languages. Inspired from this work, we have adopted a similar strategy. The state interaction diagram for our proposed clustering protocol is shown in Figure 5.2. The following sections explain in detail the algorithm for each state.

5.4.1 Un-clustered State

When a vehicle starts from parking, it would first change to the un-clustered state. It would remain in this state for un-clustered time period, t_{uc}. During this time it would listen to the wireless medium for GET_STATS packet. On receiving a GET_STATS packet from

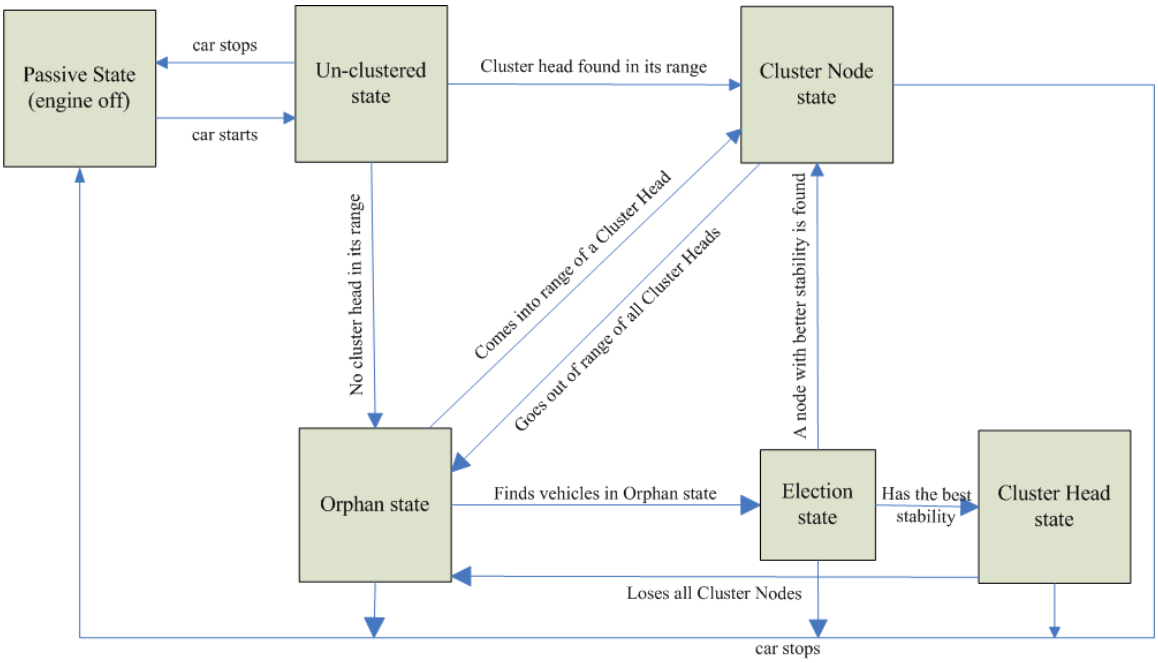


Figure 5.2: State Transition Diagram

a cluster head, it would switch to cluster node state, otherwise at the end of t_{uc} it would switch to orphan state. The algorithm for un-clustered state is shown in Algorithm 1.

Algorithm 1: Un-clustered State

```
Wait for time t_uc
if gpkt is received AND gpkt_dir = i_dir AND gpkt_n > 0 then
    i_cid = gpkt_cid
    Enter Cluster Node state
else
    Enter Orphan state
```

5.4.2 Orphan State

As the name suggests, a vehicle is in orphan state if there is no vehicle within its communication range in the same direction. There are exceptions to this rule, for example, if

the last vehicle in a road starts to lag behind the cluster head and goes out of communication range of cluster head, then it would change its state to orphan. Though in this case there would be vehicle(s) in its vicinity, they would be in cluster node state. In orphan state, the vehicle transmits GET_STATS packet at regular intervals of hello time period, t_h. The number of cluster nodes field in this GET_STATS packet will be 0. This field is used to detect whether a GET_STATS packet is sent by a cluster head or an orphan. If an orphan receives a GET_STATS packet from another orphan, then it triggers both of them to enter election state. At any point, if an orphan receives a GET_STATS packet from a cluster head, then it switches to cluster node state.

Algorithm 2: Orphan State

```
Every t_h time send gpkt with gpkt_n = 0
if gpkt is received AND gpkt_dir = i_dir then
    if gpkt_n = 0 then
        Enter Election state
    else
        i_cid = gpkt_cid
        Enter Cluster Node state
```

5.4.3 Election State

A vehicle enters this transient state to choose a cluster head and form clusters. When two or more vehicles in orphan state come within the communication range of other(s), and receive GET_STATS packet from one of them, they enter election state. At any point in election state, if a GET_STATS packet is received from a cluster head, then the vehicle withdraws from election process and changes its state to cluster node. There are three

phases in the election state. The time period of each phase is election time period, t_e, which could be same as hello time period t_h or different.

During the first phase, all the vehicles involved in election exchange GET_STATS packet. The GET_STATS packet is used to calculate the number of neighbors. Each vehicle broadcast MY_STATS packets in the beginning of second phase. On receiving MY_STATS packet from neighbors each vehicle starts building the list of neighbors (*nlist*). While building the list of neighbors, MY_STATS packet received from vehicles outside the toleration range (see Section 4.2.4) are discarded. This is done to prevent a vehicle from choosing a neighbor with high probability of moving out of the communication range anytime, as cluster head. At the end of the second phase, a potential cluster head list (*pcidlist*) is formed by sorting the neighbor list by descending stability factor value. If there are two or more neighbors with same stability value, the vehicle with lowest id wins. The stability factor is calculated using Equation 4.2. Section 4.2.2 explains in detail about stability factor and the rationale behind it.

At the start of the third phase, if a vehicle finds that it has the highest stability factor among its neighbors, then it would switch to cluster head state. Otherwise it would wait for time period t_e. If it received any NOT_CH packet (*nchpkt*) during this period, then it would delete that neighbor from the potential cluster head list. After deleting the neighbor, if a vehicle discovers that it is the most stable among its neighbors, then it switches to cluster head state. At the end of third phase, the most stable neighbor would have been chosen as cluster head and the rest would switch to cluster node state. The algorithm for a vehicle in election state is shown in Algorithm 3.

Algorithm 3: Election State

Start a timer
At any time, **if** $gpkt_n > 0$ *AND* $gpkt_{dir} = i_{dir}$ **then**
    $i_{cid} = gpkt_{cid}$
    broadcast *nchpkt*
    Enter Cluster Node state
for $timer < t_e$ **do**
    if $gpkt_n = 0$ *AND* $gpkt_{dir} = i_{dir}$ **then**
        $ncounter = ncounter + 1$
end
broadcast *mpkt* with $mpkt_{neigh} = ncounter$
for $timer < 2 * t_e$ **do**
    if $mpkt_{dir} = i_{dir}$ *AND* $distance(i, mpkt_{id}) < TOL\_RANGE$ **then**
        Add them to *nlist*
end
Add i to *nlist*
$speed_{avg} = average(gpkt_{speed}, i_{speed})$
for *1 to n in nlist* **do**
    $nlist[i]_{stability} =$
    $\alpha * nlist[i]_{neigh} + \beta * (log(rel\_speed_{max}) - log(|speed_{avg} - nlist[i]_{speed}| + \delta))$
    Add to *pcidlist* in descending order of stability
end
if $pcidlist[1]_{id} = i_{id}$ **then**
    delete *pcidlist*
    Enter Cluster Head state
for $timer < 3 * t_e$ **do**
    if *nchpkt is received with* $nchpkt_{dir} = i_{dir}$ **then**
        if $nchpkt_{id} = pcidlist[j]_{id}$ **then**
            delete $pcidlist[j]$
        if $i_{id} = pcidlist[1]_{id}$ **then**
            delete *pcidlist*
            Enter Cluster Head state
end
$i_{cid} = pcidlist[1]_{id}$
delete *pcidlist*
Enter Cluster Node state

5.4.4 Cluster Node State

Vehicles in cluster node state report their mobility statistics to the cluster head at regular intervals of hello time period, t_h. Unlike most other clustering protocol, in RSDCP the cluster node decides which of the cluster heads it wants to report to based on the stability factor. The stability factor of a cluster head with respect to a cluster node is given by Equation 4.1. Whenever a cluster node receives GET_STATS packet, it calculates stability with respect to the cluster head transmitting GET_STATS packet and stores it. It replies back with a NODE_STATS packet. A NODE_STATS packet is transmitted after a random small delay time (2 seconds) in order to prevent simultaneous transmission of NODE_STATS packets by other cluster nodes of the cluster, leading to packet collisions. If a cluster node receives GET_STATS packet from two cluster heads, it chooses the new cluster head if the stability is found to be better than that of existing cluster head by a factor of ω. The factor of ω is used to prevent frequent hand-off. To prevent replication of data, cluster node also keeps track of the time it last sent NODE_STATS packet. Suppose a stabler cluster head is found, then it would transmit a NODE_STATS packet only if it has not transmitted the packet for the last t_h time period.

Not receiving GET_STATS packet for time period t_h could imply that it has moved out of communication range of cluster head. So it readily accepts any cluster head found after that, regardless of the stability. If a cluster node does not receive GET_STATS packet for deadline time period, t_d, it assumes that there is no cluster head in its communication range and switches to orphan state. The deadline time period could be $n * t_h$ where $n > 2$. Consider a scenarios where distance between the cluster heads is slightly more than twice the communication range. If a cluster node moves from one cluster to the other, it may

miss GET_STATS packet from both the cluster heads by fraction of a second. In that case it would not receive GET_STATS packet for $2 * t_h$ time period. Consequently, it is safe to have the value of n as greater than 2. Algorithm 4 shows the algorithm for cluster node state.

Algorithm 4: Cluster Node State

```
Start a timer
if gpkt_dir = i_dir then
    stability = α * gpkt_n + β * (log(rel_speed_max) − log(|gpkt_speed − i_speed| + δ))
    if timer < t_h then
        if gpkt_cid = i_cid then
            i_stability = stability
            send npkt
        else
            if stability/i_stability > ω then
                i_cid = gpkt_cid
                i_stability = stability
                if npkt not sent for t_h then
                    send npkt
        reset timer
    else if t_h < timer < t_d then
        i_cid = gpkt_cid
        i_stability = stability
        send npkt
        reset timer
if timer > t_d then
    Enter Orphan state
```

5.4.5 Cluster Head State

The cluster head is responsible for collecting mobility statistics from its cluster nodes. In order to achieve this, it sends a GET_STATS packet at regular interval of hello timer period, t_h. Cluster nodes which finds it to be stable, reply with mobility statistics. The

cluster head collects and stores the statistics at the end of t_h. In case none of cluster nodes reply, the cluster head assumes that it has lost all of them and changes its state to orphan. The algorithm for cluster head state is shown in Algorithm 5.

Algorithm 5: Cluster Head State

for *every time period* t_h **do**
  Store previously collected statistics
  send *gpkt*
end
if $npkt_{dir} = i_{dir}$ **then**
  recalculate current statistics
if *npkt is not received for* t_h **then**
  Enter Orphan state

Chapter 6

Protocol Simulation

The proposed protocol was simulated on Network Simulator (ns-2). The following sections explain the details of the simulator environment and the method adopted to transform the proposed design into a simulation.

6.1 Network Simulator (ns-2)

Network Simulator (ns-2) [29] branched out from the REAL network simulator in 1989. ns-2 is an object oriented simulator, written in C++, with an OTcl interpreter as a frontend. ns-2 uses two languages as it has two different kinds of tasks to achieve. Firstly, it requires a system programming language to efficiently manipulate bytes, packet headers, and implement algorithms that run over large data sets. Secondly, it has to vary parameters or configurations, or quickly explore a number of scenarios. In these cases, iteration time would be important. ns-2 meets both of the requirements by using two languages, C++ and OTcl. C++ is fast to run, rendering it suitable for detailed protocol implementation. OTcl runs slower but can be altered quickly, making it ideal for simulation configuration [30].

Simulation of vehicular communication is possible in ns-2 by creating mobile nodes with wireless properties. While creating a node in the Tcl file, the physical layer and channel type can be specified as wireless. The error models can also be chosen such that they are as close to the real scenario as possible.

6.2 Application

An application was designed to verify the protocol and have an estimate of the cost to gather the density information. In the application, a QUERY packet can be sent by any vehicle which needs the density information at various places in the network. It contains the id of the querying vehicle and a timestamp. Vehicles in cluster head state which have the aggregated information of all the cluster nodes would reply. Also, vehicles in orphan state, election state, and un-clustered state reply back giving all the details to capture the exact density of the area. The response to the query is sent by transmitting CLUSTER_STATS packet. It contains the id, x and y coordinates, speed (average speed in case its cluster head), density, timestamp and timeout. The timeout option is added so that the data can be discarded after certain period of time and a new query initiated. The application packet types are shown in Table 6.1. The query and its response is implemented by using a simple controlled flooding protocol with caching.

Packet Type	State of Sender	Contents
QUERY	All	Node ID, Time Stamp.
CLUSTER_STATS	Cluster Head / Orphan / Election / Un-clustered	Node ID, Direction, X, Y, Average Speed, Density, Timestamp, Timeout.

Table 6.1: Application Packet Types

6.3 Data Structures

The Tcl file used to run simulation specifies the communication type as wireless and chooses the appropriate error model and communication range. The following is an extract from the Tcl file depicting the configuration used:

```
set opt(chan)   Channel/WirelessChannel
set opt(prop)   Propagation/TwoRayGround
set opt(netif)  Phy/WirelessPhy
set opt(mac)    Mac/802_11
set opt(ifq)    Queue/DropTail/PriQueue
set opt(ifqlen) 50
set opt(ll)     LL
set opt(ant)    Antenna/OmniAntenna

$ns_ node-config -adhocRouting gpsr \
                 -llType $opt(ll) \
                 -macType $opt(mac) \
                 -ifqType $opt(ifq) \
                 -ifqLen $opt(ifqlen) \
                 -antType $opt(ant) \
                 -propType $opt(prop) \
                 -phyType $opt(netif) \
                 -channelType $opt(chan) \
                 -topoInstance $topo \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace ON \
                 -movementTrace OFF
```

To simulate a vehicle, we created a new class `IVAgent` which was inherited from `Agent` class and also contained a `MobileNode` object. `IVAgent` inherited virtual functions `send` and `receive` from `Agent` class, used to send and receive packets from the MAC layer. The `MobileNode` object is used to retrieve the mobility information of the vehicle. The vehicle id is automatically set by the Tcl file. A set of timer objects are used for checking elapsed time and taking suitable actions. Since a vehicle could be in any of the five states, `IVAgent` encapsulates status information of all states. For example, `IVAgent` contains variables `cid_` and `cid_stability_` used in cluster node state only as well as variables `density_` and `mean_speed_` used in cluster head state only. A snapshot of `IVAgent` is shown below,

```
class IVAgent : public Agent {
private:
  ......

  MobileNode *node_;             /* the attached mobile node */
  nsaddr_t my_id_;               /* node id (address) */
  double my_x_;                  /* node location info */
  double my_y_;                  /* obtained from the attached node */

  /* Timer objects */
  IVUnclusteredTimer unclustered_timer_;
  IVHelloTimer hello_timer_;
  IVElectionTimer election_timer_;

  /* Timer periods */
  double hello_period_;
  double unclustered_period_;
  double election_period_;

  ClusterState  node_state_; /* State of the vehicle */

  /* Used in Election state only */
  /* Used to store the neighbor list during first phase of election process */
  IVNeighbors *nlist_;
  /* Used to store the list of potential cluster heads during
   * second phase of election process. */
  IVNeighbors *pcidlist_;

  /* Used in Cluster Node only */
  nsaddr_t cid_;             /* Cluster head id */
  double cid_stability_;     /* Cluster head stability */

  /* Used in Cluster Head only */
  /* Used to incrementally collect current statistics */
  IVLocalStats local_stats_;
  /* contains previously collected density information */
  double density_;
  /* contains previously consolidated average speed of cluster nodes */
  double mean_speed_;

  void turnon();              /* set to be alive */
  void turnoff();             /* set to be dead  */
```

```
  /* Timer handlers */
  void electiontout();        /* election state time-out */
  void unclusteredtout();     /* unclustered state time out */
  void hellotout();           /* hello packet time out used in cluster
                                 head, cluster node and orphan states */

  /* Packet handlers */
  void recvHello(Packet*);        /* GET_STATS packet handler */
  void recvStats(Packet*);        /* NODE_STATS packet handler */
  void recvElecStats(Packet *p);  /* MY_STATS packet handler */
  void recvNotCH(Packet *p);      /* NOT_CH packet handler */

  /* Function to send packets */
  void hellomsg();                /* sends GET_STATS packet*/
  void sendNodeStats();           /* sends NODE_STATS packet */
  void sendNotCH();               /* sends NOT_CH packet */

  .....

public:
  IVAgent();

  ......

};
```

To aid easy manipulation of neighbor data during the election process, a separate class called `IVNeighbors` was created. It has a doubly linked list containing data of each neighbor in each item of the list. It also has id, x, and y coordniates of the vehicle it is part of. This class provides member functions to add/delete neighbors and to perform operations on neighbor's information. The following is a snapshot of `IVNeighbors`,

```
class IVNeighbors {
private:
  struct gpsr_neighbor *head_;   /* Start of neighbor list */
  struct gpsr_neighbor *tail_;   /* End of neighbor list */
```

```
  int nbSize_;                    /* Number of neighbors */

  nsaddr_t my_id_;                /* my id */
  double my_x_;                   /* my geographic information */
  double my_y_;

  /* find the entry in neighbor list according to the provided id */
  struct iv_neighbor *getnb(nsaddr_t id);

  /* delete the entry in neighbors list according to the provided id */
  void delnb(nsaddr_t id);

  .....

public:
  IVNeighbors();
  ~IVNeighbors();

  .....

  /* return the number of neighbors */
  int nbsize();

  /* used to delete all neighbors */
  void delall();
  /* used to prepare pcidlist from nlist */
  nsaddr_t calculateCH();
  /* Add a neighbor during election process, first phase */
  void newNB(struct hdr_dclus_getstats *ghh);
  /* Add a neighbor during election process, second phase */
  void newNB(struct hdr_dclus_mystats *ghh);
  /* Delete a neighbor during election process, third phase */
  nsaddr_t delnb(struct hdr_dclus_notch *gnh);
};
```

As mentioned before, timers are required to keep track of time elapsed since an event occured and take suitable action when the time period exceeds certain limit. Currently, we use three timer classes, `IVUnclusteredTimer` used for un-clustered state, `IVHelloTimer` used for orphan, cluster node, and cluster head state, and `IVElectionTimer` for election

state. These classes are inherited from `TimerHandler` class and thus have member functions `sched`, `resched` and `cancel` to schedule and cancel the timers. Apart from these member functions, these classes have the `expire` member function which is called on expiration of the timer value. Given below is a snapshot of `IVElectionTimer` used in election state. Note that it has variable `round` which is used to keep track of the phase in the election state.

```
class IVElectionTimer : public TimerHandler {
public:
  .....
  IVElectionTimer(IVAgent *a) : TimerHandler() {a_=a; round = 1;}
protected:
  virtual void expire(Event *e);   /* Called when timer expires */
  IVAgent *a_;       /* linked to the vehicle object */
  uint8_t round;     /* used to keep track of the phase */
};
```

As explained in Section 5.3 and Section 6.2, there are four protocol packet types and two application packet types respectively. Data structures are constructed such that they are part of the payload of the packet transmitted. The following GET_STATS packet structure is an example of the packet structure used in the implementation.

```
struct hdr_dclus_getstats {
  u_int8_t type_;             /* packet type */
  u_int8_t no_of_neighbors_;  /* number of neighbors in orhpan state
                                 or number of cluster nodes in
                                 cluster head state */
  u_int8_t dir_;              /* direction */
  nsaddr_t cid_;              /* id of the vehicle */
  double speed_;              /* speed of the vehicle */
  .....
};
```

The primary goal of the clustering protocol is to collect density information by forming clusters. The cluster head gets the information incrementally from each cluster node. We developed a class `IVLocalStats` to collect the density information incrementally. Whenever a cluster node reports, it recalculates density and average speed values using the member function `recalculate`. At the end of hello time period t_h, the density and average speed information is stored in `density_` and `mean_speed_` member variable of `IVAgent` in the cluster head. The `IVLocalStats` member variable values are re-initialized using `reset` member function. The following gives a snapshot of `IVLocalStats` class,

```
class IVLocalStats {
public:
  .....

  IVLocalStats() { density = 0, mean_speed = 0.0;}  /* Constructor */

  /* Used to recalculate density and average speed */
  void recalculate(double speed) ;

  /* Used to retrieve current information */
  double getLocalDensity() { return density; }
  double getLocalSpeed() { return mean_speed; }

  /* Used to re-initialize values */
  void reset() { density = 0, mean_speed = 0.0; }

protected:
  double density;        /* Density value */
  double mean_speed;     /* Average speed value */
};
```

6.4 Implementation

This section describes briefly the way data structures explained in Section 6.3 were used to implement algorithms in Section 5.4. Since ns-2 is a discrete event-driven network simulator [31], our protocol was implemented in it by using two main types of events. The first event is the "timer expire event" triggered when a timer expires. The second event is the "packet receive event" triggered when a packet is received.

When a vehicle starts, `turnon` function is called where any pre-requisite settings can be made and suitable timers initiated. Once the timer expires, corresponding timer handler functions are called where relevant actions can be performed. For example, once a vehicle changes to election state, the election timer is started with timeout value t_e. Until it expires, GET_STATS packets are received. Once the timer expires, the corresponding timer handler function, `electiontout` is called, signalling the end of the first phase. The MY_STATS packet is formed and broadcasted as explained in Algorithm 3. Similarly whenever vehicle enters a state, corresponding timers are set, and on their expiration, suitable actions are performed.

When a vehicle receives a packet, it first appears in the `recv` function of `IVAgent` class. Based on the packet type, different packet handlers are called such as `recvHello`, `recvStats`, `recvElecStats` and `recvNotCH`. In the packet handler, decisions are made based on the state of the vehicle. For example, when a vehicle receives GET_STATS packet, it is first seen in `recv` function. In `recv` function, having detected it to be GET_STATS packet, `recvHello` packet handler is called. If the state of the vehicle is cluster head, it drops the packet. If the state of vehicle is cluster node, then it makes a decision whether to send a NODE_STATS packet as explained in Algorithm 4. If the state is orphan, then it could

either switch to election state or cluster node state as explained in Algorithm 2. If the state is election, then it must add the information in the packet to neighbor list or change the state to cluster node as explained in Algorithm 3. Each packet handler is developed in a similar manner. There are also functions to create packets such as `hellomsg`, `sendNodeStats` and `sendNotCH`. These in turn call `send` function to send it to lower layers.

Chapter 7

Performance Evaluations and Results

In this chapter we explain the methodology of performance evaluation and discuss the results.

7.1 Protocol Parameters

In Section 5.4 we explained the algorithm containing a number of protocol parameters. Table 7.1 summarises the protocol parameters and their repsective values used in the simulation. In real world deployment, these parameters could be changed to increase the performance.

Parameter	Description	Value
α	Weightage for number of cluster nodes	0.1
β	Weightage for relative speed	1
δ	Constant to eliminate boundary conditions in the logarithmic operation	0.000001
ω	Factor to prevent frequent hand-off	1.03
rel_speed_{max}	Maximum relative speed	20 meters/second
t_h	hello time period	10 seconds
t_{uc}	un-clustered time period	5 seconds
t_e	election time period	5 seconds
t_d	deadline time period	$4 * t_h$ / 40 seconds
TOL_RANGE	tolerance range	0.9 * communication range / 225 meters

Table 7.1: Protocol parameter values

α, β and ω values were determined empirically. The maximum difference in the speed between two vehicles moving in different lanes is usually below 45 mph. So a value of

20 meters/second was chosen for rel_speed_{max}. There are separate values for t_h, t_{uc} and t_e to optimize the performance of the protocol. Having high t_h values would reduce the number of packets but cluster head would have stale information. Having low t_{uc} value would lead to high network packets as the vehicle would quickly switch to orphan state and start broadcasting GET_STATS packet. t_e should be such that it is large enough to collect all the neighbor statistics and quickly form a cluster. If it is too large then all the vehicles would be in election state for a longer period of time with most of the time being idle. t_d should be carefully chosen. It should be always greater that twice the hello time period as explained in Section 5.4.4. Sometimes it may happen that a cluster node switches from one cluster head to another and also misses GET_STATS packet from the new cluster head due to packet collision. To optimize the performance, we have chosen t_d to be four times t_h. This can be changed based on the packet collision rate and wireless connectivity. TOL_RANGE should be chosen such that the vehicles are in the communication range at least untill the election process ends. Assuming a relative speed of 10mph, .i.e, 4.44 meters/second, the distance gained by the faster vehicle will be less than 25 meters in 5 seconds (same as t_e). The communication range was chosen as 250 meters. Therefore, TOL_RANGE was fixed as 225 meters.

7.2 Clustering protocols compared

In order to understand the performance improvement of our protocol, we designed two more protocols for comparison. The first protocol was a simple id-based protocol (IDS), similar to the one proposed in [1]. In [1], nodes periodically broadcast list of their neighboring nodes. Using this information, node with lowest id was chosen as cluster head.

In order to simulate this, clusters were formed in the same way as our protocol but dissolved after regular intervals. This would trigger election thereby leading to exchange of updated information among vehicles and choosing a new cluster head based on id. The advantage of dissolving the cluster at regular intervals is that cluster head which can cover larger number of vehicles could be chosen. The second protocol had state transitions same as the one shown in Figure 5.2. The only difference it had from RSDCP was that it used lowest id as stability factor instead of Equation 4.1 and Equation 4.2 which is based on relative speed. Henceforth we would refer it as Dynamic Clustering Protocol based on ID (IDDCP).

7.3 Experimental setup

As explained in Section 6.1, ns-2 has two languages. So the protocol was developed using C++ and the scenarios were chosen in Tcl file, used to run the simulations. The mobility traces of a node is defined as part of Tcl file. The Tcl file used to run the simulation can make use of mobility traces from a different Tcl file. VanetMobiSim [32] was used to generate the Tcl file with mobility traces. VanetMobiSim is an extension of CanuMobiSim, a flexible framework for user mobility modeling. VanetMobiSim mobility traces have been validated against TSIS-CORSIM, a well known traffic generator [32]. In VanetMobiSim, Intelligent Driver Model with Lane Change (IDM-LC) micro-mobility was chosen in the simulator since it accounts for nearby vehicles and takes advantage of multiple lanes by switching lanes and overtaking others [33].

Two scenarios, Highway and City, were chosen to evaluate the performance of the protocols. In highways, vehicles move in predictable patterns but have high speeds. In city traffic, vehicles move in unpredictable way but at low speeds. By choosing both the

scenarios, we wanted RSDCP to be evaluated for both, unpredictable movement as well as high speeds. VanetMobiSim had different parameters which can be configured. We chose three parameters, Initial stay, Safe time headway, and Traffic light time. When the simulation starts, a vehicle can start immediately or it can wait for random period before starting. This can be configured by using Initial stay parameter. In case of random initial stay, there is a scope to specify maximum period a vehicle can stay before starting. In our configurations, we specified maximum stay period as 5 seconds. The distance between vehicles can be maintained such that there is a specific time lag before which the trailing vehicle reaches the same point. This parameter is called "Safe time headway". The time period for which traffic lights will show green for a road can also be set using the Traffic light time parameter. The following sections explain in detail about each scenario and their respective VanetMobiSim parameters.

7.3.1 Highway scenario

Highway scenario consists of a 20 km (12.5 miles) long bidirectional straight road with four lanes. The minimum speed specified was 26.67 meters/second (60mph) and the maximum was 35.55 meters/second (80mph). Vehicles were randomly deployed at the ends of each road and the traffic would start to flow at the start of simulation. If a vehicle reaches the other end of the road, it would make a U-turn to continue in the opposite direction. Initial stay and Safe time mobility parameters were varied to achieve five configurations. Table 7.2 contains the details about the configurations used.

Configuration	Initial stay	Safe time headway
1	none	2s
2	random, less than 5s	2s
3	random, less than 5s	2s
4	random, less than 5s	2s
5	random, less than 5s	1.5s

Table 7.2: Highway Scenario Configurations

7.3.2 City scenario

In City scenario a customized city downtown was chosen as shown in Figure 7.1. It was a square area of 2 km (1.25 miles) with 32 bidirectional two lane roads. Each road had different speed limits, with minimum speed being 4.89 meters/second (11mph) and maximum speed being 20 meters/second (45mph). There were 14 traffic lights. Vehicles would be randomly deployed on five boundary points. Random trips were generated for each vehicle. All the three parameters of VanetMobiSim explained earlier were varied as shown in Table 7.3 to achieve five different configurations.

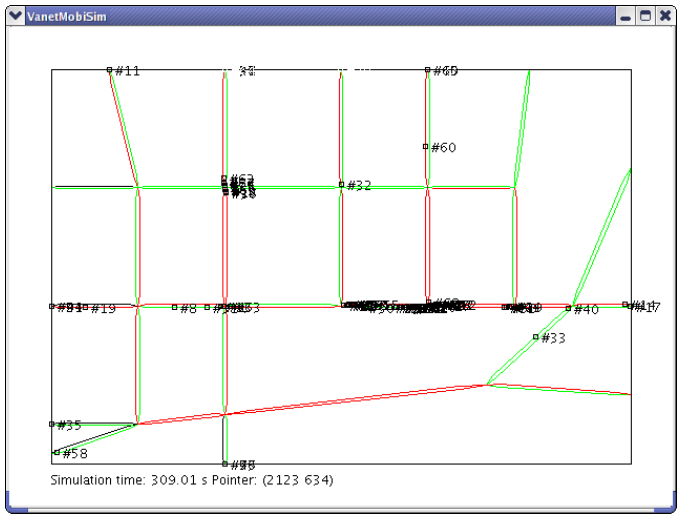


Figure 7.1: City Scenario

Configuration	Initial stay	Safe time headway	Traffic light time
1	none	2s	45s
2	random, less than 5s	2s	45s
3	random, less than 5s	2s	30s
4	random, less than 5s	2s	60s
5	random, less than 5s	2.5s	45s

Table 7.3: City Scenario Configurations

7.4 Results and Analysis

The following section explains the results comparing our proposed protocol, RSDCP with IDDCP and IDS. The number of vehicles was varied for each scenario to understand the impact of density. In the City scenario, the number of vehicles was chosen to be 50, 75, 100 and 125 while in Highway scenario, it was set to 25, 50, 75 and 100. The analysis of the results was done from three perspectives. Firstly, analysis with respect to time spent in each state. Secondly, analysis of variation of number of clusters and number of vehicles per cluster. Thirdly, analysis from the network perspective in which number of packets used in each protocol and their respective application were examined.

7.4.1 Time Distribution

The simulation time was 1000 seconds. For easy analysis, we divided the simulation time into three parts namely, Orphan time, Election time and Clustered time. Time spent by a vehicle in un-clustered state or orphan state is considered as Orphan time. We added time spent in un-clustered state to Orphan time as it is a short (5 seconds) and also during this time the vehicle is not associated with any cluster. The time period during which a vehicle is in election state is considered as Election time. Time spent by a vehicle in cluster

node state or cluster head state represents Clustered time. In each scenario, ideal Orphan time was found by measuring the time period during which the vehicle was disconnected from others. The ideal Clustered time is found by subtracting ideal Orphan time from the total simulation time.

Highway Scenario

Figure 7.2 shows the time distribution for all the three protocols and the ideal time distribution for Highway scenario. It can be seen that as the number of vehicles increases, Clustered time increases in all the three protocols. Looking closely at the figure, we can notice that the difference in Clustered time between RSDCP and other protocols increases with increase in number of vehicles. This is because at lower densities, vehicles are connected to fewer vehicles thereby having lesser choice of cluster head and so it does not make much difference if the cluster head is chosen based on id or relative speed. At higher densities, there are more choice for cluster head. Since RSDCP is based on relative speed, cluster heads are chosen such that they remain connected with cluster nodes for longer period of time. This also leads to lower Election time as the clusters are more stable. In IDS, as election is triggered every 60 seconds, the election time nearly doubles compared to IDDCP and RSDCP. The Orphan time in RSDCP is lesser than other protocols because RSDCP is able to choose a cluster head whose speed is close to average speed. This leads to even distribution of cluster heads thereby covering larger areas. If the protocol is id based as in IDS and IDDCP, the cluster heads could be distributed unevenly thereby leaving vehicles in orphan state in between the clusters. RSDCP has nearly 5% longer Clustered time than

IDDCP for all vehicle densities. RSDCP has nearly 30% increase in the Clustered time compared to IDS for all vehicle densities.

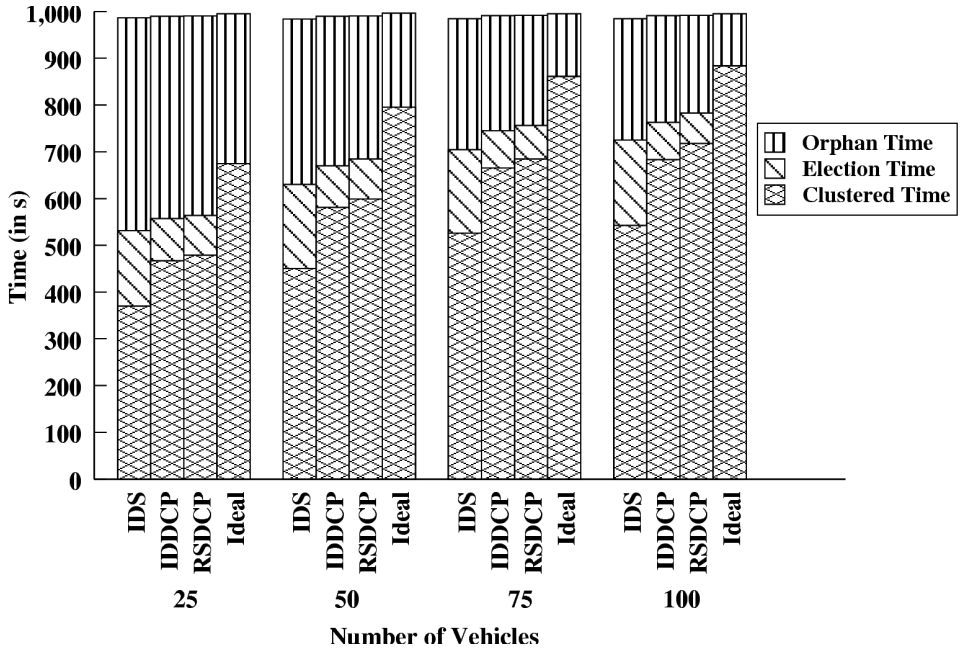


Figure 7.2: Time Distribution in Highway Scenario

City Scenario

In Figure 7.3, time distribution for the City scenario is shown. Similar to the Highway scenario, with increase in number of vehicles, clustered time increases. The difference in Clustered time between RSDCP and other protocols is steady and higher than in Highway scenario. Likewise comparing the ideal Clustered time with that of RSDCP, the difference is higher in case of City scenario. The reason is that in City scenario the traffic is more dynamic. RSDCP strives to choose a cluster head which is stable and succeeds better than IDS and IDDCP. Election time in IDS is nearly double that of IDDCP and RSDCP owing to the frequent re-election of cluster head. The Orphan time for all the protocols are

lower than that of Highway scenario with respective vehicle density because in City scenario vehicles are better connected. The Clustered time in RSDCP is nearly 4% longer than that of IDDCP. Compared to IDS, RSDCP has nearly 22% increase in Clustered time for all variations in number of vehicles.

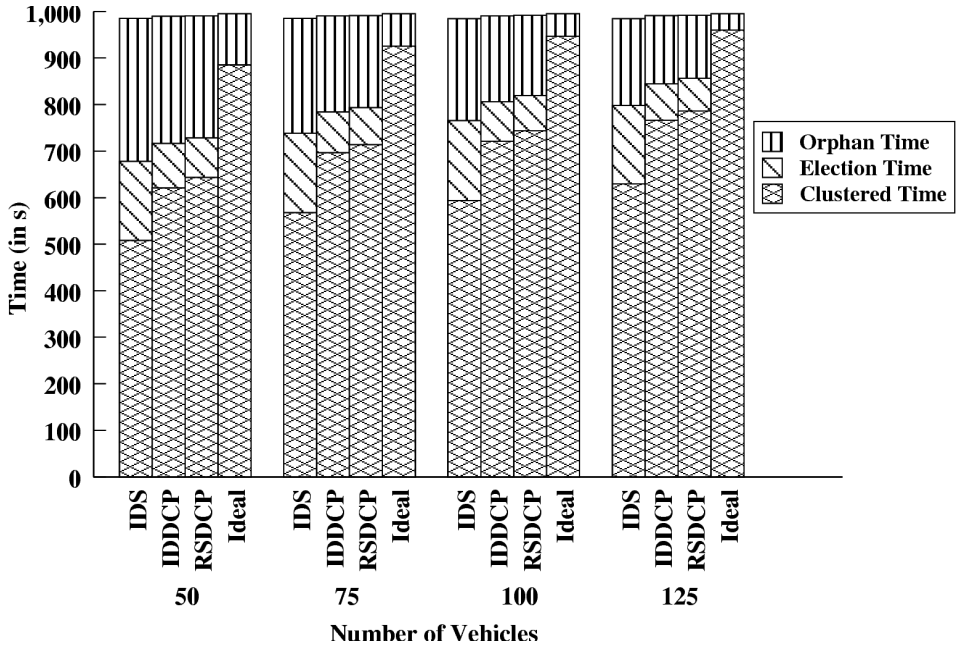


Figure 7.3: Time Distribution in City Scenario

7.4.2 Cluster statistics

Highway Scenario

Figure 7.4 shows the variation of number of clusters with respect to number of vehicles for each protocol. For fewer vehicles, all the three protocol perform almost identically, but with large number of vehicles, RSDCP starts to perform better. The reason for this is that with the increase in density of vehicles, each vehicle has more choices and has to chose the

right cluster head. RSDCP is able to achieve this by chosing a vehicle which has low relative speed with respect to its neighbors.

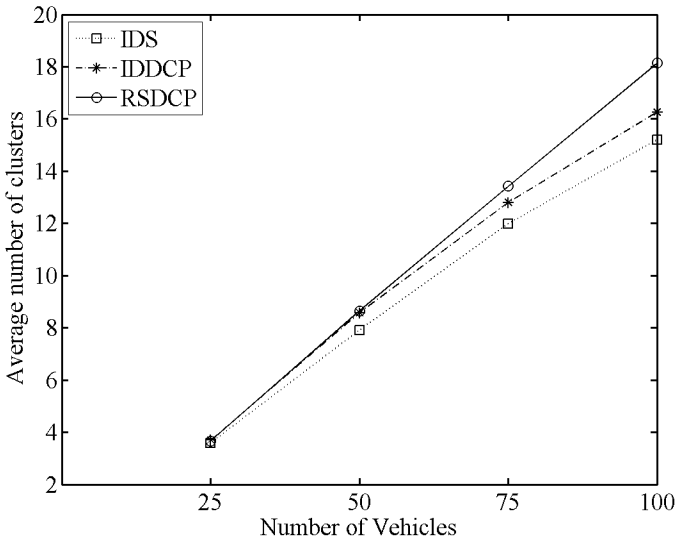


Figure 7.4: Average number of clusters in Highway Scenario

It is also important to consider the standard deviation of the number of clusters to analyze the stability of the clusters. From Figure 7.5, we can see that the standard deviation is consistently low for RSDCP compared to other protocols indicating that it forms stable clusters. There is an increase in the standard deviation of number of clusters with increase in number of vehicles as the numerical range of number of clusters becomes higher.

In Figure 7.6 the variation of average number of vehicles per cluster for Highway scenario is shown. The average number of vehicles per cluster in RSDCP is always steadily higher than IDDCP as well as IDS. Comparing Figure 7.4 and Figure 7.6 we can notice that the number of clusters as well as number of vehicles per cluster is higher in RSDCP. The reason is that the product of number of clusters and number of vehicles per cluster

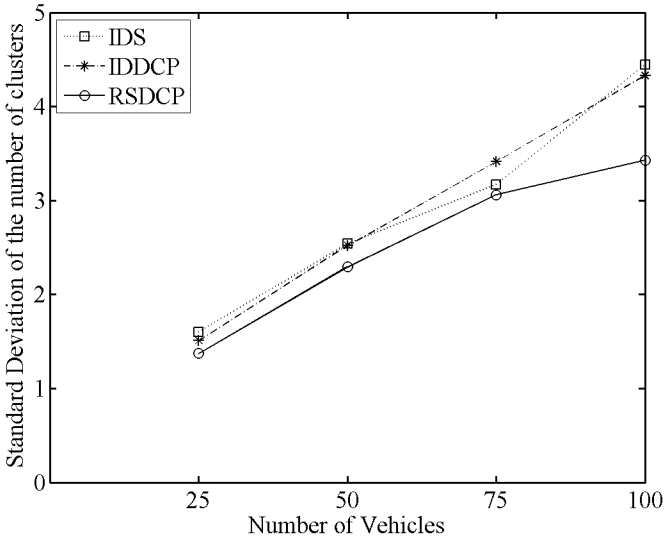


Figure 7.5: Standard Deviation of number of clusters in Highway Scenario

does not always add up to the total number of vehicles as there could be vehicles in orphan state or election state. The following illustration would give a better understanding.

Consider a scenario as shown in Figure 7.7. In Figure 7.7(a), we can see that 4 clusters are formed with average number of vehicles per cluster to be $(3 + 3 + 2 + 2)/4 = 2.5$. The problem in this case is that vehicles A, F, G and N are not part of any cluster as they are in orphan state. Consider Figure 7.7(b) in which 5 clusters are formed with average number of vehicles per cluster to be $(3 + 3 + 3 + 3 + 2)/5 = 2.8$. RSDCP is able to achieve scenario as shown in Figure 7.7(b), making it possible to get better number of clusters as well as better number of vehicles per cluster.

Figure 7.8 shows the number of times a vehicle has switched from un-clustered or orphan or election state to cluster node or cluster head state for each protocol. RSDCP performs comparable with IDDCP initially but for more number of vehicles, RSDCP starts performing significantly better by having fewer changes to clustered state. The reason

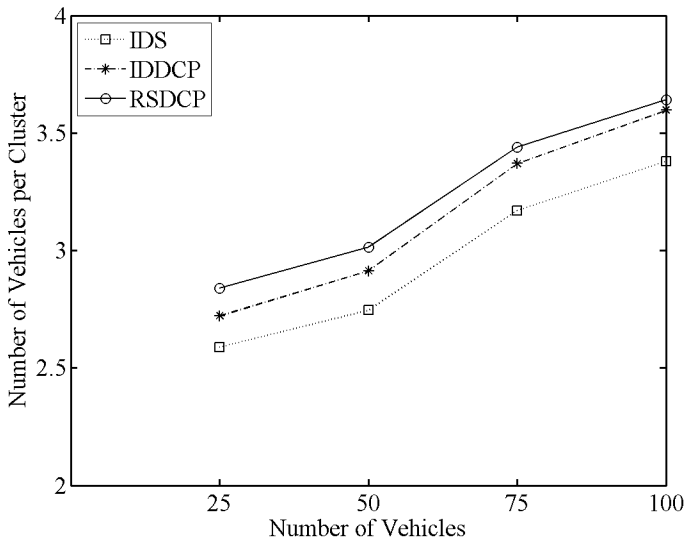
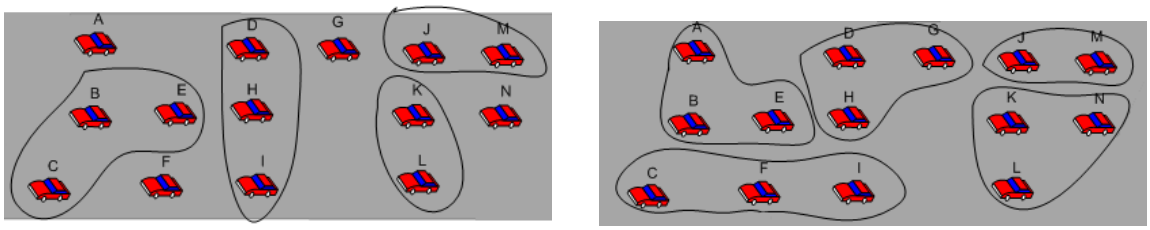


Figure 7.6: Average number of vehicles per cluster in Highway Scenario



(a) (b)

Figure 7.7: Clustering example

being that RSDCP forms stable clusters as indicated by higher average and lower standard deviation of the number of clusters. Since the election is triggered frequently in IDS, it has nearly double the number of switches as compared to RSDCP and IDDCP. Another interesting observation that can be made in the figure is that with an increase in the number of vehicles, IDS starts performing worse by switching states more number of times. This could be due to frequent election of a cluster head having high relative speed but lowest id in the neighborhood. In IDDCP, this problem does not impact too much as there are fewer elections and cluster nodes makes decisions between already elected cluster heads.

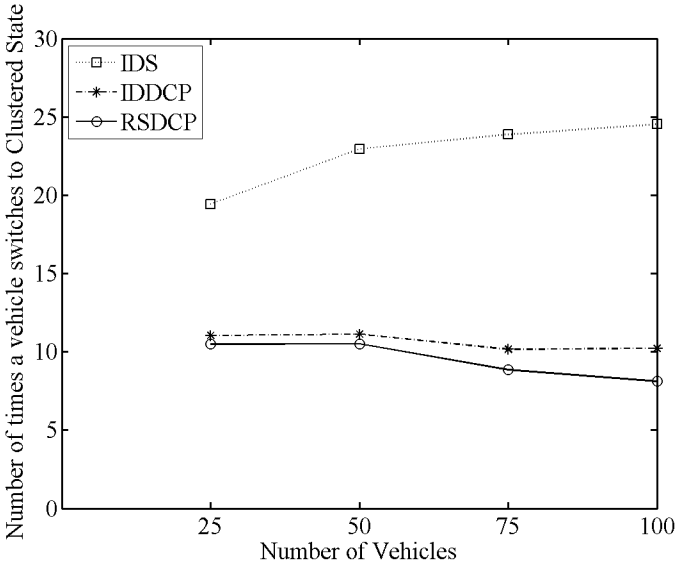


Figure 7.8: Number of times a vehicle switches to clustered state in Highway Scenario

City Scenario

In Figure 7.9, the average number of clusters is plotted against number of vehicles for different protocols for City Scenario. RSDCP and IDS have similar performance. In City scenario, more number of vehicles can be found in the communication range. Since IDS

re-organizes frequently by triggering re-election, it forms more number of clusters with less number of cluster nodes. This can be seen by the high standard deviation of the number of clusters and low vehicles per cluster in case of IDS depicted in Figure 7.10 and Figure 7.11. IDS and RSDCP have higher average number of clusters than IDDCP especially with an increase in number of vehicles.

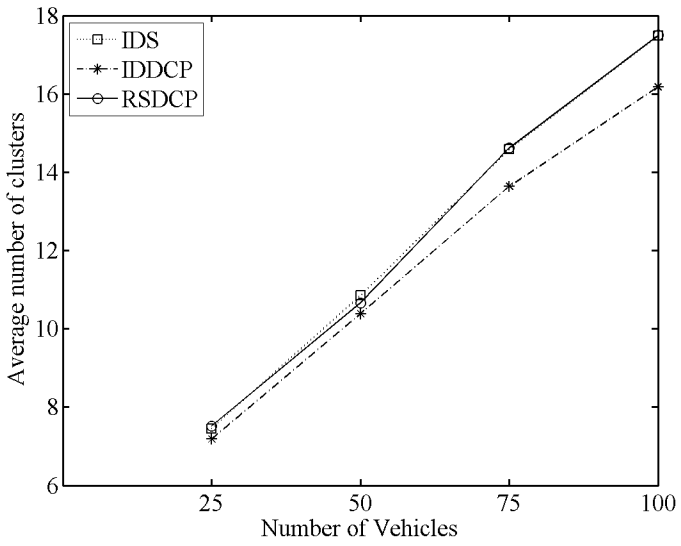


Figure 7.9: Average number of clusters in City Scenario

Figure 7.10 depicts the standard deviation of the number of clusters. It shows that RSDCP is marginally lower than IDDCP. The standard deviation randomly increases in case of IDS indicating it could lead to less stable clusters. In a city scenario, it would be difficult to keep up with the dynamics. This is evident by noticing that standard deviation in City scenario is nearly double that of Highway scenario. RSDCP though comparable in terms of standard deviation with IDDCP has higher average number of clusters. RSDCP has average number of clusters comparable to IDS but has lower standard deviation. So it suggests that RSDCP forms stable clusters as compared to IDS as well as IDDCP.

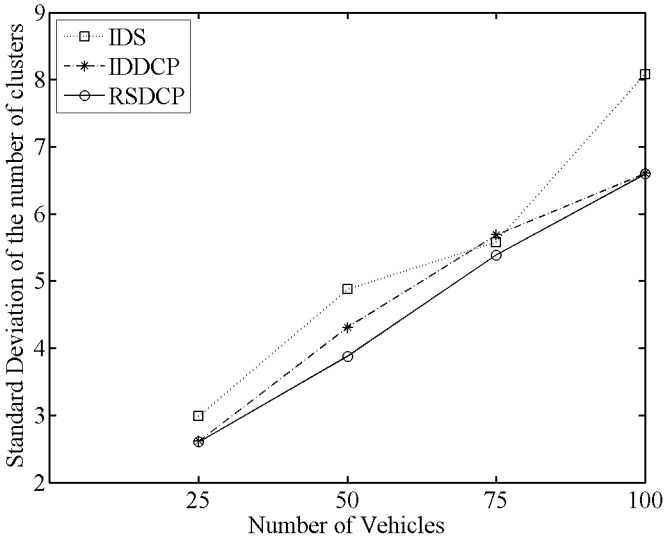


Figure 7.10: Standard Deviation of the number of clusters in City Scenario

The number of vehicles per cluster increases with an increase in number of vehicles as shown in Figure 7.11 for all the three protocols. We can notice that the number of vehicles per cluster in City scenario is higher than that of Highway scenario as the vehicles are more connected. RSDCP has consistently higher values than IDS while with IDDCP it is significantly higher except when the number of vehicles is 125.

In Figure 7.12, the number of times a vehicle switches to clustered state from unclustered or orphan or election state for all the three protocols is shown. The pattern of variation follows similar to the one in Highway scenario except for that, the difference in number of switches between IDDCP and RSDCP is much higher in case of City scenario. This is because in City scenario, the movement is not predictable for which the id based protocols are not able to adapt.

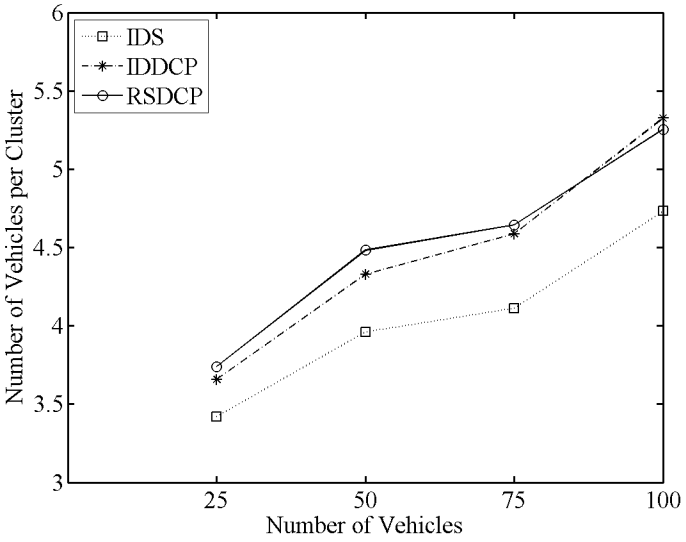


Figure 7.11: Average number of vehicles per cluster in City Scenario

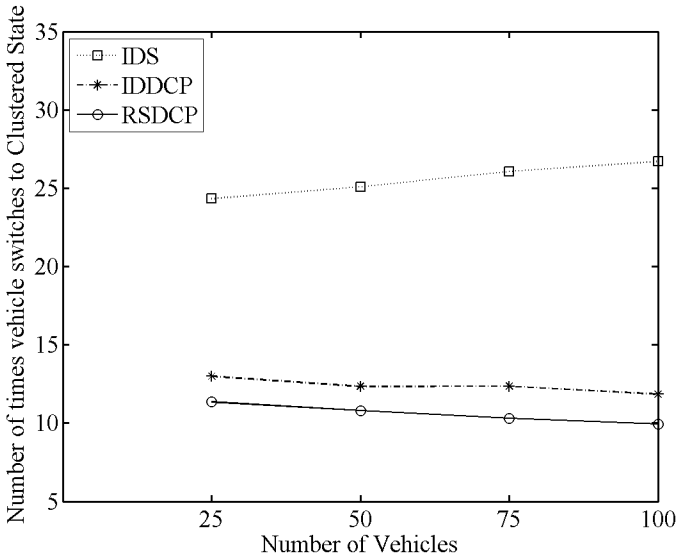


Figure 7.12: Number of times a vehicle switches to clustered state in City Scenario

7.4.3 Packet Statistics

The following section explains the statistics from the networking perspective, in terms of packets transmitted. The GET_STATS packet, NODE_STATS packet, MY_STATS packet and NOT_CH packet formed the protocol packets. The QUERY and CLUSTER_STATS packet formed the application packets. For all the configurations, ten vehicles sent query every 100 seconds making the total number of queries sent during simulation as 100. To measure the efficiency of RSDCP, percentage overhead of other protocols in terms of number of packets was computed as follows,

$$Overhead\ in\ \% = \frac{pkt_{IDDCP/IDS} - pkt_{RSDCP}}{pkt_{RSDCP}} * 100 \tag{7.1}$$

where,

$pkt_{IDDCP/IDS}$ is the number of protocol packets or application packets of IDDCP or IDS.

pkt_{RSDCP} is the number of protocol packets or application packets of RSDCP.

Highway Scenario

Figure 7.13 shows the percentage overhead in Highway scenarios comparing RSDCP with IDDCP and IDS for both, protocol packets and application packets. When the number of vehicles is 75 and 100, RSDCP has more overhead than IDDCP indicated by the negative values in the figure. In RSDCP more clusters are formed as shown in Figure 7.4. The packets used to maintain additional clusters becomes an overhead. But, if we see the overhead in the application packets, it is evident that the application running on RSDCP needs much fewer packets to collect the information. Comparing RSDCP with IDDCP, the overall overhead

of IDDCP is in the range 5% to 10%. Compared to RSDCP, IDS has overall overhead is in the range 30% to 50%.

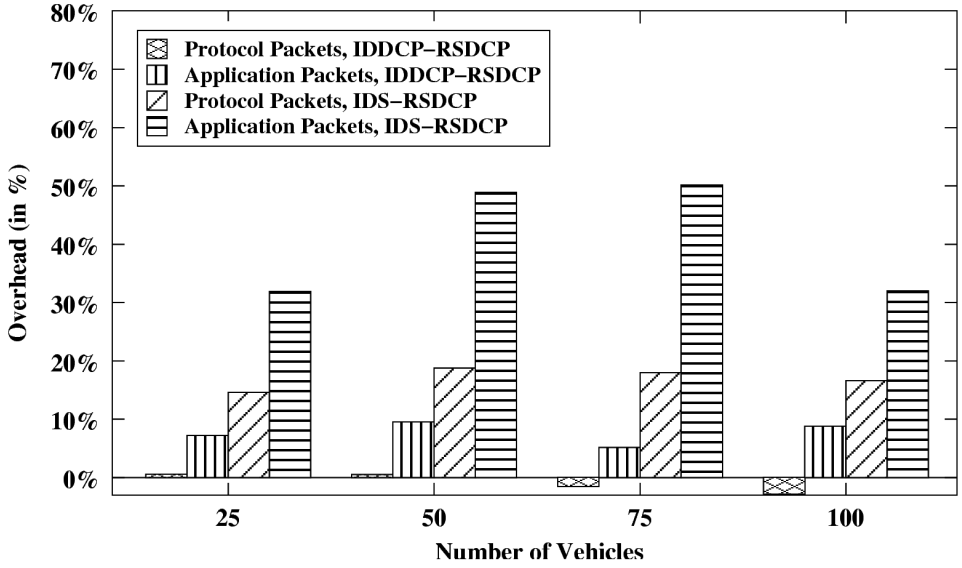


Figure 7.13: Percentage overhead of IDDCP and IDS compared to RSDCP in Highway Scenario

The application has to use more packets in case of IDS and IDDCP as there are more number of sources to collect the information. This is supported by finding the total number of sources of CLUSTER_STATS packets. Figure 7.14 shows the total number of CLUSTER_STATS sent for each protocol. As RSDCP has fewer sources, the application has to use fewer packets to collect the same information.

City Scenario

As explained in Equation 7.1, percentage overhead in the number of packets is calculated for City scenario. Figure 7.15 shows the percentage overhead in number of packets for IDDCP and IDS as compared to RSDCP. It can be seen that RSDCP has higher protocol

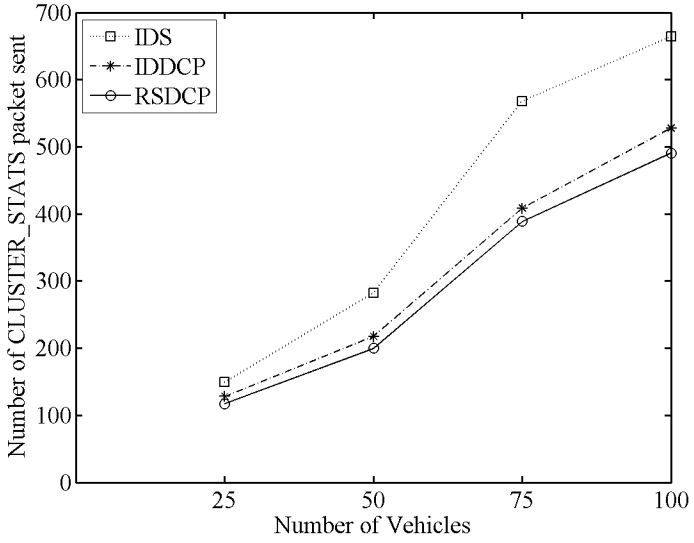


Figure 7.14: Number of CLUSTER_STATS packets sent in Highway Scenario

overhead as it has more cluster heads collecting the density information. However, application packets in case of IDDCP and IDS have more overhead, about 10% in case of IDDCP and 40%-70% in case of IDS, making the overall efficiency of RSDCP much higher.

Similar to the Highway scenario, in the City scenario the number of sources from which the application has to collect information is low in case of RSDCP. This is indicated by the number of sources of CLUSTER_STATS packet shown in Figure 7.16. The main advantage with fewer packets, especially in wireless protocols, is that there will be fewer packet collisions.

Comparing the overall results of RSDCP and IDS we notice that RSDCP performs better in all the perspectives. Comparing the overall results of RSDCP and IDDCP we notice that RSDCP performs better than IDDCP but not drastically. It may appear that IDDCP makes use of id, a simpler information to collect. Once the clustering protocol forms part of a network routing protocol or application, the vehicles have to exchange their

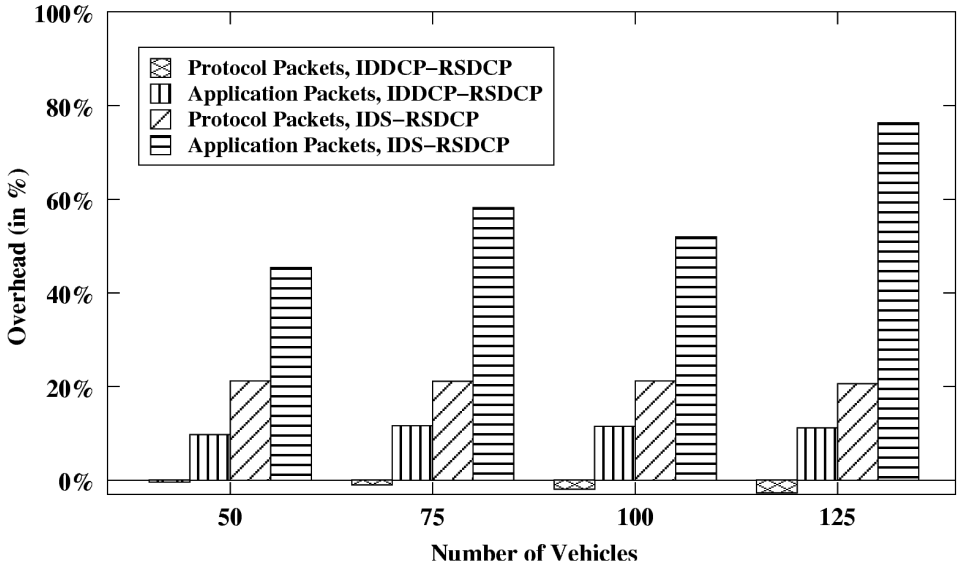


Figure 7.15: Percentage overhead of IDDCP and IDS compared to RSDCP in City Scenario

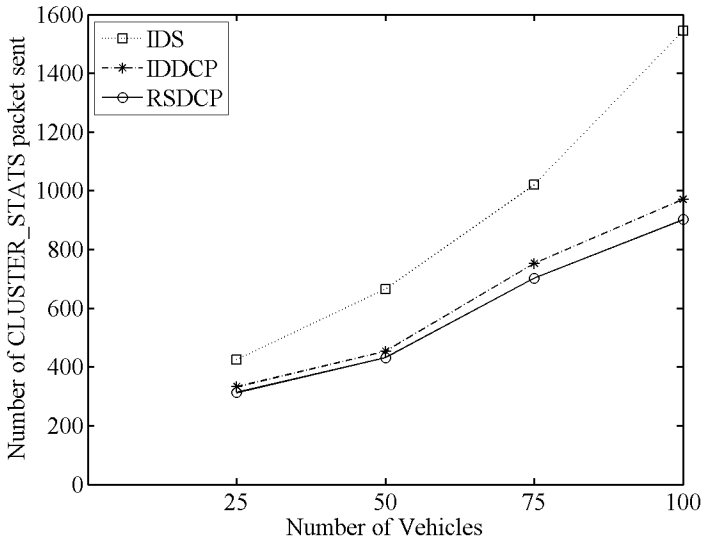


Figure 7.16: Number of CLUSTER_STATS packets sent in City Scenario

speed and number of neighbors information. In this way collection of speed and number of neighbors required for RSDCP will be easily available. Besides in wireless communication among vehicles even a slight decrease in the number of packets transmited implies lesser packet collisions resulting in more reliablity in communication.

Chapter 8

Conclustion and Future Work

In this thesis, we have proposed, simulated and evaluated a novel clustering protocol, RSDCP for Intelligent Vehicles. Summarizing the results, we notice that,

- RSDCP forms stable cluster indicated by higher average number of clusters and lower standard deviation.
- RSDCP has lower number of switches to clustered state corroborating that it forms stable cluster.
- The number of vehicles per cluster is high in RSDCP showing that it forms large clusters.
- RSDCP has higher Clustered time which is a resultant of stable and large clusters.
- In order to achieve all these RSDCP has a slightly higher protocol overhead which is insignificant as the load on the application is lessened to a large extent.

To improve the proposed clustering protocol, traffic modeling techniques could be investigated for developing better stability factor. If it is possible to get real world mobility traces, the clustering protocol could be validated against them to get more confidence in the protocol. As discussed in Section 6.2, to completely realize the distributed server architecture, the next step would be to design an algorithm which optimally chooses certain cluster heads to perform the role of distributed servers.

The main contribution of this thesis is envisioning distributed servers to collect density information explained in Section 6.2 and designing a clustering protocol well suited for Intelligent Vehicles. The architecture, once implemented, would be useful to everyday travelers as it would minimize the traffic delay time. It would also be useful in building vehicular routing protocols which can be used by many future Intelligent Vehicle applications.

Bibliography

[1] Mario Gerla and Jack Tzu-Chieh Tsai. Multicluster, mobile, multimedia radio network. *Journal of Wireless Networks*, 1(3):255–265, 1995.

[2] S. Hahn. Automation of driving functions - future development, benefits and pitfalls. In *1996 IEEE, Intelligent Vehicles Symposium*, pages 309–312, Tokyo, Japan, 19-20 September 1996.

[3] Jing Zhao and Guohong Cao. VADD: Vehicle-assisted data delivery in vehicular ad hoc networks. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications*, pages 1–12, April 2006.

[4] Research U.S. Department of Transportation and Bureau of Transportation Statistics Innovative Technology Administration. Transportation statistics annual report, December 2006.

[5] David Schrank and Tim Lomax. The 2005 urban mobility report, May 2005.

[6] Richard Bishop. A survey of intelligent vehicle applications worldwide. In *2000 IEEE, Intelligent Vehicles Symposium IV*, pages 25–30, Dearborn, MI, 3-5 October 2000.

[7] Li Li, Jingyan Song, Fei-Yue Wang, Wolfgang Niehsen, and Nan-Ning Zheng. Ivs 05 new developments and research trends for intelligent vehicles. *IEEE Intelligent Systems*, 20(4):10–14, July-August 2005.

[8] Stefano Basagni. Distributed clustering for ad hoc networks. In *ISPAN '99: Proceedings of the 1999 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '99)*, pages 310–315, Washington, DC, USA, 1999. IEEE Computer Society.

[9] Prithwish Basu, Naved Khan, and Thomas D.C. Little. A mobility based metric for clustering in mobile ad hoc networks. In *International Conference on Distributed Computing Systems Workshop*, 2001.

[10] Lichun Bao and J.J. Garcia-Luna-Aceves. Topology management in ad hoc networks. In *MobiHoc 2003: Proceedings of the 4th ACM International Symposium on Mobile Ad hoc Networking and Computing*, pages 129–140, Annapolis, Maryland, USA, 2003. ACM Press.

[11] Siva Sivavakeesar, George Pavlou, and Antonio Liotta. Stable clustering through mobility prediction for large-scale multihop intelligent ad hoc networks. In *Wireless Communications and Networking Conference, IEEE*, volume 3, pages 1488–1493, 21-25 March 2004.

[12] Zouhair El-Bazzal, Michel Kadoch, Basile L. Agba, Franois Gagnon, and Maria Bennani. An efficient management algorithm for clustering in mobile ad hoc network. In *International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems: Proceedings of the ACM International Workshop on Performance Monitoring, Measurement, and Evaluation of Heterogeneous Wireless and Wired Networks*, pages 25–31, Torremolinos, Spain, 2 October 2006. ACM Press.

[13] Ouri Wolfson, Bo Xu, Sam Chamberlain, and Liqin Jiang. Moving objects databases: Issues and solutions. In *International Conference on Statistical and Scientific Database Management*, pages 111–122, Capri, Italy, July 1998.

[14] Jussi Myllymaki and James Kaufman. High-performance spatial indexing for location-based services. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 112–117, New York, NY, USA, 2003. ACM Press.

[15] A. Prasad Sistla, Ouri Wolfson, Sam Chamberlain, and Son Dao. Modeling and querying moving objects. In *IEEE International Conference on Data Engineering*, pages 422–432, Birmingham, UK, April 1997.

[16] Zhiming Ding and Ralf Hartmut Guting. Managing moving objects on dynamic transportation networks. In *16th International Conference on Scientific and Statistical Database Management*, pages 287–296, 21-23 June 2004.

[17] Dimitris Papadias, Jun Zhang, Nikos Mamoulis, and Yufei Tao. Query processing in spatial network databases. In *29th Conference on Very Large Databases (VLDB)*, pages 790–801, Berlin, Germany, 2003.

[18] Badri Nath and Dragos Niculescu. Routing on a curve. In *HotNets-I*, Princeton, NJ, USA, 2002.

[19] Google maps. Website, http://www.google.com/mobile/index.html.

[20] Yahoo! maps. Website, http://maps.yahoo.com/.

[21] Microsoft live. Website, http://maps.live.com/.

[22] Source for google real time traffic data. Website, http://news.zdnet.com/2100-9588_22-6163203.html.

[23] Google maps FAQ. Website, http://www.google.com/support/mobile/bin/answer.py?answer=39891&topic=9120.

[24] Jinyang Li, John Jannotti, Douglas De Couto, David Karger, and Robert Morris. A scalable location service for geographic ad-hoc routing. In *6th ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 120–130, August, 2000.

[25] Zygmunt J. Haas and Ben Liang. Ad hoc mobility management with uniform quorum systems. *IEEE ACM Transactions on Networking*, 7(2):228–240, April, 1999.

[26] Ivan Stojmenovic. Home agent based location update and destination search schemes in ad hoc wireless networks, September 1999.

[27] Fei Yu and Victor C.M. Leung. Mobility-based predictive call admission control and bandwidth reservation in wireless cellular networks. In *INFOCOM 2001: Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE*, volume 1, pages 518–526, Anchorage, AK, USA, 22-26 April 2001.

[28] A. Bruce McDonald and Taieb F. Znati. Design and performance of a distributed dynamic clustering algorithm for ad-hoc networks. In *34th Annual Simulation Symposium*, pages 27–35, Seattle, WA, USA, 22-26 April 2001.

[29] ns-2 homepage. Website, http://www.isi.edu/nsnam/ns/.

[30] ns-2 manual. Website, http://www.isi.edu/nsnam/ns/ns-documentation.html.

[31] Ns by example. Website, http://nile.wpi.edu/NS/.

[32] Vanetmobisim homepage. Website, http://vanet.eurecom.fr/.

[33] Marco Fiore, Jerome Harri, Fethi Filali, and Christian Bonnet. Vehicular mobility simulation for vanets. In *40th Annual Simulation Symposium, ANSS 07*, pages 301–309, Norfolk, VA, USA, March 2007.