APROVE: A Stable and Robust VANET Clustering Scheme using Affinity Propagation

by

Christine Shea

A thesis submitted in conformity with the requirements for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Copyright © 2009 by Christine Shea
Abstract

APROVE: A Stable and Robust VANET Clustering Scheme using Affinity Propagation

Christine Shea
Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2009

The need for an effective clustering algorithm for Vehicle Ad Hoc Networks (VANETs) is motivated by the recent research in cluster-based MAC and routing schemes. VANETs are highly dynamic and have harsh channel conditions, thus a suitable clustering algorithm must be robust to channel error and must consider node mobility during cluster formation. This work presents a novel, mobility-based clustering scheme for Vehicle Ad hoc Networks, which forms clusters using the Affinity Propagation algorithm in a distributed manner. This proposed algorithm considers node mobility during cluster formation and produces clusters with high stability. Cluster performance was measured in terms of average cluster head duration, average cluster member duration, average rate of cluster head change, and average number of clusters. The proposed algorithm is also robust to channel error and exhibits reasonable overhead. Simulation results confirm the superior performance, when compared to other mobility-based clustering techniques.
Acknowledgements

This work was supported by AUTO21 Network of Centres of Excellence and in-kind contributions of Mark IV Industries.

I wish to express my gratitude to my supervisor, Prof. Shahrokh Valaee, for his invaluable guidance and encouragement in producing this work.

I would also like to thank my colleagues at Wireless and Internet Research Laboratory (WIRLab) for their constructive comments.
# Contents

1 Introduction

1.1 Motivation ........................................... 1
1.2 Advantages to Clustering in VANETs .................. 4
1.3 Cluster-based MAC and Routing Protocols for VANETs 5
1.4 Scope and Objectives ................................ 7
1.5 Contributions ........................................ 9

2 Background .............................................. 10

2.1 Clustering in MANETs ............................... 10
   2.1.1 Lowest-ID and Least Cluster Change Algorithms ........ 12
   2.1.2 MOBIC ............................................ 12
2.2 Clustering in VANETs ............................... 14
2.3 Data Clustering ...................................... 16
   2.3.1 K-Means Clustering .............................. 17
   2.3.2 Affinity Propagation ............................ 18
2.4 Factor Graphs and the Sum-Product Algorithm ........ 22
   2.4.1 Factor Graphs .................................... 22
   2.4.2 Sum-Product Algorithm ......................... 24
   2.4.3 Belief Propagation and Loopy Belief Propagation ...... 27
2.5 Derivation of Affinity Propagation .................. 29
3 APROVE Clustering Scheme for VANETs

3.1 Similarity Function
  3.1.1 Similarity Function Parameters

3.2 Neighbour List and Message Updating
  3.2.1 Neighbour List
  3.2.2 Availability and Responsibility Update Rules

3.3 APROVE Message Passing
  3.3.1 Segregated Message Passing
  3.3.2 Aggregated Message Passing
  3.3.3 Behaviour of Message Passing

3.4 APROVE Clusterhead Selection and Maintenance
  3.4.1 APROVE with Clustering Interval, CI
  3.4.2 APROVE with Clusterhead Contention
  3.4.3 Asynchronous APROVE

3.5 Chapter Summary

4 Analysis of APROVE Algorithm

4.1 Affinity Propagation as Pearl’s Belief Propagation

4.2 Parameter Selection
  4.2.1 Self-Similarities (Input Preferences)
  4.2.2 Future Prediction Period: $\tau_f$
  4.2.3 Clustering Interval: $CI$

4.3 Overhead Analysis
  4.3.1 Overhead of aggregated message passing algorithm
4.3.2 Comparison to Overhead of MOBIC ........................................ 58
4.4 Convergence Analysis ............................................................... 59
  4.4.1 Convergence of APROVE to Non-Oscillating States ................. 60
  4.4.2 Convergence of APROVE to Coherent Clusterhead states .......... 61
4.5 Robustness to Channel Error ..................................................... 63
  4.5.1 Robustness of APROVE with Clustering Interval ..................... 63
  4.5.2 Robustness of APROVE with Clusterhead Contention ............... 64
  4.5.3 Robustness of Asynchronous APROVE ................................. 64
4.6 Synchronization Analysis .......................................................... 65
  4.6.1 Synchronous APROVE ....................................................... 65
  4.6.2 Asynchronous APROVE ..................................................... 66

5 Simulation Results ........................................................................ 68
  5.1 Simulation Set-up ...................................................................... 69
    5.1.1 Network Simulator ......................................................... 69
    5.1.2 Traffic Simulator ............................................................ 69
  5.2 Clustering Algorithm Performance Metrics .............................. 71
  5.3 Parameter Optimization .......................................................... 72
    5.3.1 Tuning Self-Similarities .................................................. 73
    5.3.2 Tuning Future Prediction Period ....................................... 75
  5.4 Mobility Performance ............................................................. 77
    5.4.1 Mobility Performance of APROVE with Clustering Interval .... 78
    5.4.2 Mobility Performance of APROVE with Clusterhead Contention 81
    5.4.3 Mobility Performance of Asynchronous APROVE ................ 83
  5.5 Overhead Performance ............................................................. 86
  5.6 Robustness to Channel Error ................................................... 87
5.6.1 Robustness of APROVE with Clustering Interval . . . . . . . . . . 88
5.6.2 Robustness of APROVE with Clusterhead Contention . . . . . . 91
5.6.3 Robustness of Asynchronous APROVE . . . . . . . . . . . . . . . 93

6 Conclusions and Future Work ........................................... 97
   6.1 Future Work .......................................................... 99

Bibliography ................................................................. 101
List of Tables

3.1 A summary of the different clusterhead fields . . . . . . . . . . . . . . . . 46

5.1 The speed distributions of the mobility scenarios . . . . . . . . . . . . . . 70
# List of Figures

1.1 A VANET in a city scenario ........................................ 2  
1.2 A VANET preventing a multi-vehicle collision .................. 4  
2.1 A clustered network topology ...................................... 11  
2.2 The K-means Clustering Algorithm [1] ............................. 18  
2.3 The Affinity Propagation Algorithm [2] ............................ 21  
2.4 Factor graph example for a global function factorization ......... 23  
2.5 Calculation of the marginal function using a factor graph ....... 25  
2.6 The sum-product algorithm ......................................... 26  
2.7 Factor graph for the Affinity Propagation algorithm [3] ........... 30  
3.1 The APROVE clustering algorithm ................................. 34  
4.1 Timing diagram for APROVE with clustering interval .......... 66  
4.2 Timing diagram for asynchronous APROVE ....................... 67  
5.1 Tuning of Self-Similarities with $CI = 60s$ ....................... 73  
5.2 Tuning of Self-Similarities with $vel = 33m/s$ and $CI = 60s$ .... 74  
5.3 Tuning of Self-Similarities with $vel = 33m/s$ and $CI = 120s$ ... 74  
5.4 Tuning of $\tau_f$ parameter with Self-Similarities = $-2000$, and $vel = 11m/s$ 75  
5.5 Tuning of $\tau_f$ parameter with Self-Similarities = $-2000$, and $vel = 22m/s$ 76  
5.6 Tuning of $\tau_f$ parameter with Self-Similarities = $-2000$, and $vel = 33m/s$ 76  
5.7 Tuning of $\tau_f$ parameter with Self-Similarities = $-2000$, and $vel = 44m/s$ 77
Chapter 1

Introduction

1.1 Motivation

From 2000 through 2004, there were 14,082 deaths in Canada caused by motor vehicle accidents. This was 32% of all accidental deaths in Canada during that period [4]. The National Highway Traffic Safety Administration (NHTSA) in the US estimated that in the year 2000, the total economic impact of motor vehicle collisions was $230.6 billion [5]. This included lifetime costs, property damage, medical expenses, costs of travel delay, and lost market productivity.

The extreme economic and social impacts of motor vehicle accidents have motivated the research and development of Intelligent Transportation Systems (ITS). Intelligent transportation systems aim to improve the safety, security and efficiency of surface transportation systems, through the implementation of intelligent technologies. These intelligent systems encompass a variety of technologies, however most applications rely fundamentally on the ability for communication in the vehicular environment. Vehicular communication may constitute vehicle-to-roadside (V2R) communication or vehicle-to-vehicle (V2V) communication. Communication amongst vehicles involves the formation of a vehicular ad hoc network (VANET), whereby vehicles are able to form and maintain
a network without pre-existing infrastructure. VANETs are a vital component of ITS, and they can help alleviate traffic congestion, prevent motor vehicle accidents, and aid in navigation.

The safety-driven applications of vehicular communication have stringent reliability and delay requirements, which are not satisfied by current wireless standards. As a result, many consortia have been initiated to tackle this difficult problem [6]. To enable communication in the vehicular environment, the Dedicated Short Range Communication (DSRC) standard was created. DSRC comprises a set of communication protocols and standards specific for automotive use. In addition, the Federal Communications Commission (FCC) allocated 75MHz of spectrum for DSRC use in 1999. Since the current WIFI standards do not satisfy the DSRC requirements, a new draft amendment to the IEEE 802.11 standard was created called IEEE 802.11p or Wireless Access in Vehicular Environments (WAVE).

The vast sensor network that VANETs will create is inciting countless applications,
and making VANETs a hot topic in ad hoc networking today. VANETs will enhance driver safety and reduce traffic deaths and injuries by implementing collision avoidance and warning systems. In addition, VANETs can help navigation and relieve traffic congestion by providing a driver with live routes that avoid road hazards and bottleneck areas as illustrated in Figure 1.1. Some of the exciting VANET applications that have been suggested are summarized below:

1. **Roadway Condition Awareness and Obstacle Warning**: VANETs will enable the dissemination of traffic and road conditions [7]. Vehicles can warn one another of upcoming dangers or obstacles on the road, such as an ice-covered ramp or an oil slick. In addition, dissemination of traffic conditions will allow vehicles to avoid congested routes.

2. **Collision Avoidance**: Collision avoidance is an important application to avoid both line-of-sight collisions and hidden collisions occurring at an intersection [8]. This application can also include the avoidance of chain collisions (multi-car pile-ups) as illustrated in Figure 1.2.

3. **Cooperative Driving/Adaptive Cruise Control**: VANETS will also enable cooperative driving and adaptive cruise control [9]. This may include automatic breaking to avoid collision, or smooth acceleration and deceleration of the vehicle during heavy traffic. An adaptive cruise control application in heavy traffic would greatly aid the flow of traffic and reduce travel times.

Although VANETs offer many exciting applications to ensure the future safety and ease of the roads, many complications must first be overcome. The VANET topology and harsh VANET environment present many difficult communication challenges. As described in the next section, most of these challenges can be addressed by a clustered network topology.
1.2 Advantages to Clustering in VANETs

The dynamic and dense VANET topology and the harsh VANET environment, produce many challenges for communication and networking. In traditional Mobile Ad hoc Network (MANET) research, these difficulties were often overcome by a clustered topology. As a result, clustering has become a common topic in the VANET research community.

One of the many challenges for VANETs is the dynamic and dense network topology, resulting from the high mobility and high node-density of vehicles [10]. This dynamic topology causes routing difficulties as well as congestion from flooding, and the dense network leads to the hidden terminal problem. A clustered structure can make the network appear smaller and more stable in the view of each node [11], [12]. By clustering the vehicles into groups of similar mobility, the relative mobility between communicating neighbour nodes will be reduced, leading to intra-cluster stability. In addition, the hidden terminal problem can be diminished by clustering [13].

Another issue generated by the dynamic and dense network, is the “broadcast storm problem” [14]. The broadcast storm problem describes the congestion resulting from re-broadcasts and flooding in a MANET. The dynamic topology of VANETs demand a high frequency of broadcast messages to keep the surrounding vehicles updated on position and safety information. In addition, many routing algorithms necessitate flooding the network to find routes, which in a dynamic network needs to be done frequently to keep
routes updated. All of this flooding leads to severe congestion, which can be alleviated by a clustered topology [14], [15]. When the network is clustered, only the clusterhead participates in finding routes, which greatly reduces the number of necessary broadcasts. In addition, MAC schemes using different CDMA codes in adjacent clusters can greatly reduce interference.

An additional challenge for VANETs is Quality-of-Service (QoS) provisioning. In VANETs, many different types of data will need to be transmitted, and messages will be both delay-intolerant and delay-tolerant. For example, safety messages will demand high reliability and low delay, whereas non-vital road and weather information will be tolerant to longer delays. These different data types necessitate QoS provisioning, which can be achieved by a clustered network [16], [17].

1.3 Cluster-based MAC and Routing Protocols for VANETs

Motivated by the many advantages of clustering described above, there has been much research on cluster-based VANETs in the recent literature. Most of the research has been focused on developing cluster-based routing protocols, as in [18] and [19], and cluster-based MAC protocols, as in [13, 17, 20 – 24].

The Cluster Based Location Routing (CBLR) algorithm [19] defines a simple way to form clusters. If a node is undecided, it starts a timer and broadcasts a Hello message. If the node hears a response from a clusterhead before the timer expires, it becomes a member of that clusterhead. Otherwise, the node becomes a clusterhead itself. In CBLR, the clusterhead contains the position information of its neighbouring clusters, and packets are routed to the clusterhead nearest to the destination.

In both [13] and [17], the clusterhead takes on a managerial role and facilitates intra-cluster communication by providing a TDMA schedule to its cluster members. In [17],
adjacent clusters are also assigned different CDMA codes to avoid interference between clusters. When compared to traditional 802.11 MAC, the work in [17] has been shown to substantially reduce probability of message delivery failure. In terms of the clustering algorithm used, many of the cluster-based MAC schemes use a version of the clustering method from CBLR. In [13], CBLR clustering is modified during clusterhead contention (when two clusterheads are within range of one another). During clusterhead contention, the winning clusterhead is the one with both lower relative mobility and closer proximity to its neighbours. Alternatively, [24] uses CBLR but addresses mobility by first classifying nodes into speed groups, such that nodes will only join a clusterhead of similar velocity.

The recent research in cluster-based MAC and routing protocols for VANETs motivates the need for an effective VANET clustering scheme. However, the clustering schemes used in the above research are lacking in cluster stability when faced with the highly mobile VANET environment. These algorithms do not exhibit cluster stability because they make no attempt to select a stable clusterhead during initial clusterhead election. For highly-mobile networks, mobility must be considered during the clustering process in order to ensure cluster stability. Mobility is considered in [13] as a reactive measure, in that it is only considered after two clusters collide. Speed groups are used in [24] to cluster vehicles of similar velocity, which may result in some stability improvement; however, the predefined speed groups have very large variations in speed. Since the clusters provide the foundation for cluster-based MAC and routing schemes, cluster stability is vital for achieving reliable communication. Stable mobility-based clustering schemes do exist for general MANETs, such as MOBIC [25].

In addition to stability, an effective clustering algorithm must be robust to the harsh channel conditions present in the VANET environment. As discussed in [10], VANET’s have unreliable radio channel characteristics. The high mobility of the environment and numerous reflective obstacles lead to shadowing and multipath fading. It is thus important to have an algorithm that is robust to channel error.
1.4 Scope and Objectives

The goal of this work is to create a mobility-based clustering algorithm for VANETs, which is also robust to channel error. The proposed algorithm is a distributed clustering algorithm, which possesses excellent cluster stability, where stability is defined by long clusterhead duration, long cluster member duration, and low rate of clusterhead change. In addition, the algorithm is robust to channel error and exhibits a reasonable overhead. The algorithm is achieved by utilizing a new data clustering technique called Affinity Propagation (AP) [2]. The clustering scheme uses a vehicle’s position (provided by GPS) and velocity information to form clusters with low relative mobility between the cluster heads and their cluster members. The performance of the proposed algorithm is validated against the mobility-based ad-hoc clustering scheme, MOBIC [25]. MOBIC is well-established and stability-driven, and thus provides a good benchmark for the algorithm’s success.

In Chapter 2, the relevant background material is presented. The concept of clustering is introduced, followed by a literature review of MANET clustering as well as a further discussion of VANET clustering. Next, this chapter explores static data clustering, such as K-means clustering and Affinity Propagation. To derive the Affinity Propagation algorithm, the concepts of factor graphs, sum-product algorithm, and belief propagation are introduced. Affinity propagation is derived with loopy sum-product algorithm, thus the convergence of loopy sum-product algorithm is also discussed.

Chapter 3 introduces the proposed VANET clustering algorithm, called Affinity PROpagation for VEhicular networks (APROVE). The APROVE algorithm uses affinity propagation from a communications perspective and in a distributed manner. Messages are passed between neighbouring nodes, and each node independently makes its own clustering decision. Convergence is ensured by using a convergence flag to indicate when a specific node converges to clusterhead status. Two different message passing procedures are introduced for APROVE. In “segregated” message passing, each of the affinity
Chapter 1. Introduction

Propagation messages are sent separately, whereas “aggregated” message passing combines all clustering-related messages into one packet. The aggregated message passing procedure is an improvement, due to its lower overhead. Three different formulations for clustering decisions are also presented. These include APROVE with a clustering interval, APROVE with clusterhead contention, and asynchronous APROVE. The first two methods demand synchronization amongst the clustering intervals, whereas the last formulation is completely asynchronous.

In Chapter 4, analysis of the APROVE algorithm is presented. This chapter provides some insight into the operation of APROVE. First, APROVE’s parameter settings are discussed. Next, the overhead of APROVE and MOBIC is computed and compared. Convergence analysis is then presented, where convergence describes both non-oscillation of the algorithm and coherency of clusterhead decisions. Robustness of APROVE is discussed for all three APROVE formulations, and some channel-error induced issues are revealed. Finally, the synchronization requirements for the three different formulations are presented.

In Chapter 5, the simulation results are presented. The clustering performance of APROVE is verified against MOBIC, due to the lack of validated VANET clustering schemes in the recent literature. The clustering performance metrics used in the chapter include: clusterhead duration, cluster member duration, average rate of clusterhead change, and average number of clusters. This chapter begins by tuning the parameters of APROVE to the highway scenario. Next, the clustering performance is measured as a function of both node mobility and channel error, to measure the mobility performance and robustness respectively. All clustering performance evaluations are performed on all three APROVE formulations and compared.

Finally, Chapter 6 presents the concluding remarks and some topics of future work.


1.5 Contributions

The contributions of this work are summarized as follows, including the chapters presenting them, and publications referring to them:

1. **APROVE message passing procedures**: A segregated message passing procedure for APROVE is presented, which enables a distributive version of affinity propagation. In segregated message passing, nodes broadcast the different affinity propagation messages in separate packets. (Chapter 3 – Section 3.3.1 and [26])

   In addition, an aggregated message passing procedure is introduced. This is an improvement to segregated message passing, where all clustering related information is broadcast in one packet. This message passing procedure reduces the overhead of the algorithm. (Chapter 3 – Section 3.3.2)

2. **APROVE with clustering interval**: A formulation of APROVE is presented, where new clustering decisions are made periodically with a period equal to the Clustering Interval parameter. (Chapter 3 – Section 3.4.1 and [26])

3. **APROVE with clusterhead contention**: An improvement to APROVE with clustering interval is made, where clusterheads within range of one another contend for clusterhead status. (Chapter 3 – Section 3.4.2)

4. **Asynchronous APROVE**: A fully distributed and asynchronous version of APROVE is proposed. (Chaper 3 – Section 3.4.3)
Chapter 2

Background

2.1 Clustering in MANETs

A Mobile Ad hoc NETwork (MANET) is a self-organizing wireless network of mobile nodes, which can communicate without pre-existing infrastructure. MANETs can have either a flat topology or a hierarchical clustered topology. Research has shown that a flat topology faces scalability issues in large networks [27]. Scalability issues arise in large MANETs with a flat topology, where routing leads to congestion and the broadcast storm problem [14]. Routing in MANETs requires flooding to find routes and in large networks this flooding leads to severe congestion. This problem worsens as mobility is increased, since in a highly mobile network, broadcasts need to be frequent enough to keep surrounding nodes aware of the dynamic topology.

The work in [14] suggests a hierarchical, cluster-based approach to solve MANET scalability issues. Clustering creates a backbone network of nodes, providing scalability for large networks, and stability for dynamic networks. Some mobility-based MANET clustering schemes are presented in [25], [28], and [29]. Many other clustering algorithms have been suggested, and [12] provides a comprehensive survey of MANET clustering schemes.
Clustering is the process of separating the nodes of a network into organized partitions called clusters. The clusters form sub-networks in the overall network, thus forming the hierarchical topology. Nodes in a cluster must be one of the following types:

1. **Clusterhead (CH):** An elected node that acts as the local controller for the cluster. The clusterhead’s responsibilities may include: routing, relaying of inter-cluster traffic from cluster members, scheduling of intra-cluster traffic, and channel assignment for cluster members.

2. **Cluster Member:** A normal node belonging to a cluster. Cluster members usually do not participate in routing, and they are not involved in inter-cluster communication.

3. **Gateway Node:** This is an optional node, which is used in some clustering schemes. The gateway node belongs to more than one cluster, acting as the bridge between clusterheads. When present, the gateway nodes participate in both forwarding of inter-cluster traffic and the routing process. The clusterheads and gateway nodes form the backbone network.

A clustered network (without gateway nodes) is shown in Figure 2.1.

![Figure 2.1: A clustered network topology](image)

Clustering in MANETs reduces congestion and increases the stability of dynamic networks. The cluster members don’t participate in route establishment, thus the overall
packet flooding is greatly reduced. In mobile networks with highly dynamic topologies, clustering can increase network stability. If clusters are designed and maintained with mobility in mind, then intra-cluster relative mobility can be minimized. This increased stability leads to more constant routes. In addition, changes to the network topology (a node joining or leaving) have a smaller impact on routing in a clustered network. New additions have a local impact only, and can join a cluster without changing the global routing paths.

2.1.1 Lowest-ID and Least Cluster Change Algorithms

A common distributive clustering scheme with low overhead is the Lowest-ID algorithm [30]. In Lowest-ID, each node is assigned a unique ID, and the node with the lowest ID in its two-hop neighbourhood is elected to be the clusterhead. The algorithm works as follows: (1) Each node periodically broadcasts its unique ID, along with the ID of its neighbours. (2) If a node has the lowest ID of all ID’s it hears, it becomes a clusterhead. (3) The lowest-ID a node hears is its clusterhead, unless that node gives up clusterhead status to another lower ID node. In this case, the node will re-evaluate lowest ID status amongst undetermined nodes. (4) A node that hears from more than one clusterhead is a gateway node.

In an effort to reduce the frequent re-clustering involved in maintaining the lowest-ID status of all clusterheads, the Least Cluster Change (LCC) algorithm was suggested [31]. In LCC, re-clustering is only performed when two clusterheads come within range of one another. At this point, the clusterhead with the lower ID remains the clusterhead.

2.1.2 MOBIC

An extension to the lowest-ID algorithm, which considers mobility, is MOBIC [25]. The motivation for MOBIC was to improve cluster stability by considering mobility during initial cluster formation. MOBIC proposes an aggregate local mobility metric, which is
the basis for cluster formation instead of node ID. The goal is for the node with the lowest relative mobility to its neighbours to be elected as clusterhead.

MOBIC assumes that the power received from node $X$ at node $Y$, $RxPr_{X\rightarrow Y}$ is proportional to the distance between the nodes. The relative mobility between node $X$ and node $Y$ is then approximated by taking the ratio of $RxPr$ at node $Y$ for two successive “hello” messages from node $X$. The relative mobility metric, $M^\text{rel}_Y(X)$, at node $Y$ with respect to $X$, is as follows:

$$M^\text{rel}_Y(X) = 10 \log_{10} \frac{RxPr_{X\rightarrow Y}^{\text{new}}}{RxPr_{X\rightarrow Y}^{\text{old}}}$$  \hspace{1cm} (2.1)$$

In the above metric, if $RxPr_{X\rightarrow Y}^{\text{new}} < RxPr_{X\rightarrow Y}^{\text{old}}$ then $M^\text{rel}_Y(X) < 0$, which implies the nodes are moving away from one another. On the other hand, if $RxPr_{X\rightarrow Y}^{\text{new}} > RxPr_{X\rightarrow Y}^{\text{old}}$ then $M^\text{rel}_Y(X) > 0$, which indicates that the nodes are moving towards one another. Therefore, the closer $M^\text{rel}_Y(X)$ is to zero, the lower the relative mobility.

Node $Y$ calculates an aggregate mobility metric by considering the $M^\text{rel}_Y(X_i)$ for each neighbour, $X_i$. The aggregate mobility metric is found by finding the variance, with respect to zero, for the set of relative mobility values, $M^\text{rel}_Y(X_i)$. This aggregate mobility metric, $M_Y$, is computed in (2.2).

$$M_Y = \text{var}_0\{M^\text{rel}_Y(X_j)\}_{j=1}^m = E\left[\left(M^\text{rel}_Y\right)^2\right]$$  \hspace{1cm} (2.2)$$

MOBIC begins with every node in UNDECIDED state. Every node then periodically broadcasts its aggregate mobility metric to its neighbours. If a set of nodes is in the UNDECIDED state, the node with the lowest $M$ amongst its neighbours assumes CLUSTERHEAD status and broadcasts this decision. If a node hears from a neighbour in CLUSTERHEAD state, then it joins the node and declares CLUSTER_MEMBER status. Similar to LCC, clusterhead re-election only occurs when two clusterheads move within range of one another. However, MOBIC allows incidental, short-lived contact of clusterheads, and only re-clusters if the clusterheads are within range for longer than the Cluster Contention Interval (CCI) time period.
One disadvantage of MOBIC is its failure to consider mobility during cluster maintenance. If a clusterhead is elected, it will remain a clusterhead until it meets another clusterhead and triggers re-clustering. During this time, cluster members are allowed to pass freely amongst clusterheads, thus low relative mobility between a clusterhead and its members cannot be guaranteed throughout the lifetime of the cluster. As discussed in [12], MOBIC is most effective on networks with group mobility behaviour, such as a highway of vehicles. When groups of nodes move with similar and constant speeds, the relative mobility amongst clusters will remain relatively stable. However, if node mobility is random and nodes change their speeds often, then the elected clusterhead will not maintain low relative mobility to its members, and performance will degrade.

2.2 Clustering in VANETs

Vehicle ad hoc networks are a special instance of MANETs, where the mobile nodes are vehicles with mobility that has both deterministic and stochastic qualities. The deterministic aspects of vehicular mobility include the assumption that vehicles will drive around the speed limit, and the assumption that vehicles will follow the speed of the vehicles in front of it. Some stochastic aspects of vehicular mobility include: speeding (or excessively slow driving), passing, lane-changing, and drastic speed changes caused by an accident or onset of heavy traffic.

The many challenges associated with VANETs provide ample motivation for a clustered network. VANETs suffer from the hidden terminal problem and have congestion issues caused by routing in the dense and highly mobile environment. Buildings and larger vehicles lead to shadowing, which results in a rapidly changing topology. Congestion, a dynamic topology, and the hidden terminal problem can all be alleviated by clustering the network. In addition, VANETs require Quality-of-Service (QoS) to deal with the delay-intolerant safety messages and the delay-tolerant data. Clustering has
been shown to aid in fulfilling QoS requirements, as described in [16].

As a result of the advantages described above, there is ample recent research surrounding cluster-based VANETs. Most of this research has been focused on the development of cluster-based MAC protocols and cluster-based routing protocols instead of the clustering scheme itself. Cluster-based MAC protocols are presented in [20, 13, 21, 22, 23, 24, 17] and cluster-based routing protocols are presented in [18] and [19]. In [13] and [17], the clusterhead takes on a managerial role and facilitates intra-cluster communication by providing a TDMA schedule to its cluster members. In [17], adjacent clusters are assigned different CDMA codes to avoid interference between clusters. This work shows a substantial reduction in probability of message delivery failure, when compared to the traditional 802.11 MAC.

Much of the recent VANET research discussing cluster-based MACs and routing schemes, also present a low-maintenance clustering algorithm. Each of these algorithms works essentially the same way, whereby nodes periodically transmit HELLO beacons to indicate their present state. States can be one of the following: Undecided, Clusterhead, Cluster Member, and sometimes Gateway. An undecided node will join the first clusterhead that it hears a HELLO beacon from (or joins all clusterheads if Gateway nodes are allowed). If the node does not hear from a clusterhead within a given time period, it will become a clusterhead itself. In addition, protocols are introduced to deal with colliding clusters, which occurs when two clusterheads come within range of one another. During a cluster collision, one clusterhead decides to give up its status to the other. This technique is used by [17] and [19] without regard for mobility. In [13], mobility is addressed during cluster collision, whereby the winning clusterhead is the one with both lower relative mobility and closer proximity to its members. Alternatively, [24] addresses mobility by first classifying nodes into speed groups, such that nodes will only join a CH of similar velocity.

The above clustering techniques offer low complexity, but in the highly mobile VANET
environment, they are lacking in cluster stability. The algorithms do not have a proactive approach to cluster stability, in that they make no attempt to select a stable clusterhead during initial clusterhead election. Node mobility must be taken into consideration in order to achieve stability, however many of the proposed techniques ignore it. Mobility is considered in [13] as a reactive measure, in that it is only considered after two clusters collide. Speed groups are used in [24] to cluster vehicles of similar velocity, which may result in some stability improvement; however, the predefined speed groups have very large variations in speed. For example, one speed group contains the range of velocities between 60kmph and 110kmph. This range will include most if not all vehicles on a road with a speed limit of 80kmph, and thus each cluster will still have high intra-cluster relative mobilities.

The VANET clustering algorithms proposed thus far lack in cluster stability, and do not consider mobility effectively. The recent research has been focused on cluster-based MAC and routing protocols as opposed to the underlying clustering schemes. The clustering schemes are simply proposed and not validated in terms of performance against any benchmark MANET clustering schemes.

2.3 Data Clustering

The clustering discussed thus far is from an ad hoc networking perspective. Clustering is also used in scientific data analysis, where it is designed to process and detect patterns in data. Data clustering is a static, one-shot process that searches data for a set of centers, or exemplars, which best describe the data. In this context, clustering aims to minimize the distance between each data point and its assigned exemplar, where distance could be Euclidean distance, or any other application-specific function.
2.3.1 K-Means Clustering

An early data clustering scheme still used today, is called K-Means Clustering [32]. K-means clustering is a static, location-based scheme, which aims at partitioning a set of data into k clusters by reducing the mean-squared error. This algorithm aims to minimize (2.3).

\[ J = \sum_{j=1}^{k} \sum_{i=1}^{n} \| x_i^{(j)} - c_j \|^2 \]  

(2.3)

for k clusters and n nodes, where \( \| x_i^{(j)} - c_j \|^2 \) is the application specific distance measure between each data point \( x_i^{(j)} \) and its cluster centre, \( c_j \).

The key to the K-means algorithm, is that the number of desired clusters needs to be set a priori to the clustering process. The algorithm works as follows:

1. Place K centers amongst the data points being clustered
2. Assign each data point to the nearest centre, thus partitioning into clusters
3. Find the centroid of each cluster, and make the centroids the new centers
4. Re-assign the data points to the nearest of the new centers
5. Repeat 1–4 until the centers remain stable

The K-means clustering algorithm is quite simple to implement, however it does not always find the global optimum to the objective function. The algorithm is very sensitive to the initial selection of clusterheads. Therefore, to find a satisfactory solution, the algorithm needs to be run many times with different initializations, and the best overall result is selected. For large networks, this algorithm can incur unreasonably long processing times, due to the necessary repetitions. The other obvious disadvantage is the requirement of setting the number of clusters prior to clustering. If the exact network topology is unknown prior to clustering, then it would be difficult to pre-set the number of clusters. In addition, this requirement makes K-means clustering ineffective for mobile networks, since in a mobile network, the number of clusters will likely be changing.
2.3.2 Affinity Propagation

A revolutionary new technique for data clustering is the Affinity Propagation (AP) algorithm [2], [3]. Affinity Propagation solves the issues presented by K-means clustering, and is a much more efficient and effective algorithm. It has been shown to produce clusters in much less time, and with much less error than K-means clustering, where error refers to the application-specific distance between each data point and its assigned exemplar. In addition, Affinity Propagation does not require that the number of clusters be known prior to clustering, instead, the clusters emerge naturally. Clustering is done by passing messages along the edges of the network, which relate the current affinity that one node has for choosing another node as its exemplar.
Similarity Function

This algorithm takes an input function of similarities, \( s(i, j) \), where \( s(i, j) \) reflects how well suited data point \( j \) is to be the exemplar of data point \( i \). Affinity Propagation aims to maximize the similarity \( s(i, j) \) for every data point \( i \) and its chosen exemplar \( j \). For example, the similarity function could be the negative Euclidean distance between data points. Negative Euclidean distance is used so that a maximum similarity corresponds to the closest data points.

Each node \( i \) also has a self-similarity, \( s(i, i) \), which influences the number of exemplars that are identified. The self-similarities are the input preferences for the algorithm. Initializing a data point with a larger or smaller self-similarity, respectively increases or decreases the likelihood of the data point becoming an exemplar. If all the data points are initialized with the same constant self-similarity, then all data points are equally likely to become exemplars. The self-similarity also controls how many clusters are produced. By increasing and decreasing this common self-similarity input, the number of clusters produced is increased and decreased respectively. If all data points are assigned the median of the input similarities, a moderate number of clusters is produced, and if they are assigned the minimum of the input similarities, the smallest number of clusters is produced (one large cluster).

Message Passing: Responsibility and Availability

There are two types of messages passed in this technique. The responsibility, \( r(i, j) \), is sent from \( i \) to candidate exemplar \( j \) and indicates how well suited \( j \) is to be \( i \)'s exemplar, taking into account competing potential exemplars. The availability, \( a(i, j) \), is sent from candidate exemplar \( j \) back to \( i \), and indicates how appropriate it would be for \( i \) to choose \( j \) as its exemplar. The availability calculation considers the positive responsibility feedback from surrounding data points. For example, if \( j \) is receiving strong responsibility messages from surrounding data points, then \( j \) will send a stronger availability to indicate
Chapter 2. Background

its exemplar suitability.

The self-responsibility, \( r(i, i) \) and self-availability, \( a(i, i) \) are two additional messages calculated for each data point, \( i \). Both of these messages reflect accumulated evidence that \( i \) is an exemplar, and they are used to find the clusters. The self-responsibility bases exemplar suitability on input preference and the maximum availability received from surrounding data points. Whereas the self-availability bases exemplar suitability on the number and strength of positive received responsibilities.

The update formulas for the responsibility (2.4), self-responsibility (2.5), availability (2.6), and self-availability (2.7) are stated below. The algorithm is begun by calculating the responsibilities with the availabilities set to 0.

\[
\begin{align*}
    r(i, j) &\leftarrow s(i, j) - \max_{j' \text{ s.t. } j' \neq j} \left\{ a(i, j') + s(i, j') \right\} \quad (2.4) \\
    r(i, i) &\leftarrow s(i, i) - \max_{j' \text{ s.t. } j' \neq i} \left\{ a(i, j') + s(i, j') \right\} \quad (2.5) \\
    a(i, j) &\leftarrow \min \left\{ 0, r(j, j) + \sum_{i' \notin \{i, j\}} \max \left\{ 0, r(i', j) \right\} \right\} \quad (2.6) \\
    a(j, j) &\leftarrow \sum_{i' \text{ s.t. } i' \neq j} \max \left\{ 0, r(i', j) \right\} \quad (2.7)
\end{align*}
\]

The responsibility and availability message updates must be damped to avoid numerical oscillations that will prevent the algorithm from converging. This is done by updating new messages as follows: \( m_{new} = \lambda m_{old} + (1 - \lambda)m_{new} \), where \( \lambda \) is a weighting factor between 0 and 1.

Cluster Decisions

In Affinity Propagation, the exemplar of each data point \( i \) is found with (2.8).

\[
\text{exemplar}_i = \arg \max_j \{ a(i, j) + r(i, j) \} \quad (2.8)
\]
This clustering procedure may be performed at any iteration of the algorithm, but final clustering decisions should be made once the algorithm stabilizes. The algorithm can be terminated once exemplar decisions become constant for some number of iterations, indicating that the algorithm has converged.

It should be noted that the algorithm possesses another useful feature; it is possible to determine when a specific data point has converged to exemplar status for a specific iteration. When a data point’s self-responsibility plus self-availability becomes positive, that data point has become the exemplar for its cluster.

**Affinity Propagation Summary**

The affinity propagation procedure is illustrated below in Figure 2.3. At the end of each iteration, (2.8) is used to determine the current clusters. It can be seen that as the algorithm progresses, the exemplars and surrounding cluster members emerge naturally.

![Figure 2.3: The Affinity Propagation Algorithm [2].](image)

Affinity Propagation has many advantages over traditional data clustering techniques. It is much faster, more optimal (the distance between cluster members and exemplars is
less), and the number of clusters is not required as an input. The technique, however, can not be applied directly to mobile ad hoc networks, since it is meant for static data clustering. This algorithm is traditionally performed on sets of data, thus no actual communication is taking place. The technique assumes a centralized controller that knows the condition of every data point, and this controller performs the *message passing* by iteratively updating matrices containing the responsibility and availability messages.

Affinity Propagation was derived using a factor graph and the sum-product algorithm (also known as Loopy Belief Propagation). This relevant material is explained in more detail in the next section, along with a brief derivation of Affinity Propagation.

### 2.4 Factor Graphs and the Sum-Product Algorithm

Factor graphs and the sum-product algorithm provide a means to understand and derive many complicated algorithms in computer science and engineering. A comprehensive tutorial on factor graphs and the sum-product algorithm is presented in [33]. If a system contains a global function of many interacting variables, and this global function can be approximated by factors of local functions, each of which depends on a subset of the variables, then the sum-product algorithm may be used. For example, in (2.9), the global function \( F(x_1, x_2, \ldots, x_n) \) can be factored into the product of several local functions:

\[
F(x_1, x_2, \ldots, x_n) = \prod_k f_k(x_k) \tag{2.9}
\]

where \( x_k \) is a subset of \( \{x_1, x_2, \ldots, x_n\} \), and \( f_k(x_k) \) is a local function with \( x_k \) as arguments.

#### 2.4.1 Factor Graphs

Factor graphs provide a simple way to model the factorization of a global function into local functions. A factor graph contains a variable node for every variable \( x_i \), and a factor
node for every local function $f_k$. The factor node $f_k$ is connected to the variable node $x_i$, if and only if $x_i$ is an argument of the local function $f_k$.

This concept is best understood with a simple example. Let the global function $F(x_1, x_2, x_3, x_4)$ be factored as in (2.10). This factorization can be expressed by the factor graph in Figure 2.4.

$$F(x_1, x_2, x_3, x_4) = f_1(x_1, x_2, x_3)f_2(x_2)f_3(x_3, x_4)f_4(x_4)$$ (2.10)

When a system has a global function of many variables, the marginal functions $F(x_i)$ are often desired. The marginal $F(x_i)$ can be computed by summing over all of the variables except $x_i$, as in (2.11):

$$F(x_i) = \sum_{\{x_N\} \setminus i} F(x_N)$$ (2.11)

where $x_N$ is the full set of variables, $\{x_1, x_2, \ldots, x_n\}$, and $\{x_N\} \setminus i$ is the set of variables minus $x_i$.

The marginal functions, $F(x_1)$ and $F(x_2)$, for the example function in (2.10) are
computed in (2.12) and (2.13) respectively.

\[
F(x_1) = \sum_{x_2} \sum_{x_3} \sum_{x_4} F(x_1, x_2, x_3, x_4) \\
= \sum_{x_2} \sum_{x_3} \left( f_1(x_1, x_2, x_3) f_2(x_2) \times \left( \sum_{x_4} f_3(x_3, x_4) f_4(x_4) \right) \right) \quad (2.12)
\]

\[
F(x_2) = \sum_{x_1} \sum_{x_3} \sum_{x_4} F(x_1, x_2, x_3, x_4) \\
= f_2(x_2) \times \sum_{x_1} \sum_{x_3} \left( f_1(x_1, x_2, x_3) \times \left( \sum_{x_4} f_3(x_3, x_4) f_4(x_4) \right) \right) \quad (2.13)
\]

It is observed from these examples that the computation of the marginal functions can be greatly simplified by exploiting the distributive law. Note how the factors are organized in an efficient sum-product-sum-product recursion. This organization can be found directly from the factor graph using a method known as the sum-product algorithm.

### 2.4.2 Sum-Product Algorithm

For the special case of cycle-free factor graphs, the sum-product algorithm is an exact method for calculating the marginals. A simplified version of the algorithm is introduced by exploring the connections between the factor graph in Figure 2.4 and the simplified marginal expression of \( F(x_2) \) in (2.13). To calculate the \( F(x_2) \) marginal function, redraw the factor graph as a tree with the variable node \( x_2 \) as the root as shown in Figure 2.5.

The algorithm begins at the leaves of the factor graph, and the nodes pass messages up the tree to the root. The message passing rules are as follows:

1. Each variable node, \( x_i \), sends the product of all the messages received from its children.

2. Each factor node, \( f_i \), with parent node, \( x_i \), takes the product of itself, \( f_i(x_i) \), with the messages received from its children and then finds the sum over all variables.
Figure 2.5: Calculation of the marginal function, $F(x_2)$ using a factor graph

except $x_i$. Where $x_1$ is the set of arguments for the local function at factor node, $f_i$.

This message passing is illustrated in Figure 2.5, where the $\sum_{\sim x_i}$ operator means to take the sum over all variables except $x_i$.

The above example finds the marginal function for one specific variable, but it is also possible to calculate all of the marginals simultaneously. This is done by performing the above algorithm on all nodes, where no particular node acts as the root. When a node sends a message to a neighbour, that neighbour becomes the temporary parent, and all other neighbours become the children. This generalized message passing is called the sum-product algorithm, and it calculates the marginal functions at all nodes. There are two types of messages passed in the sum-product algorithm. The message passed from a variable node, $x$, to a function node, $f$, is denoted $\mu_{x\rightarrow f}(x)$, and the message passed from a function node, $f$, to a variable node, $x$, is denoted $\mu_{f\rightarrow x}(x)$. The update rules for the sum-product algorithm’s two messages are as follows:
Chapter 2. Background

Figure 2.6: The sum-product algorithm

Message from variable node to function node:

\[ \mu_{x\to f}(x) = \prod_{g \in n(x) \setminus f} \mu_{g\to x}(x) \]  

(2.14)

Message from function node to variable node:

\[ \mu_{f\to x}(x) = \sum_{X \setminus x} \left( f(X) \prod_{y \in n(f) \setminus x} \mu_{y\to f}(y) \right) \]  

(2.15)

where the set of arguments of function \( f \) is \( X = n(f) \). The neighbouring nodes of \( x \) and \( f \) are \( n(x) \) and \( n(f) \) respectively, where \( x \)'s neighbours are all function nodes, and \( f \)'s neighbours are all variable nodes.

This message passing procedure is illustrated in Figure 2.6. The message sent from a variable node \( x \) to a factor node \( f \), is the product of all messages received at \( x \), except \( f \)'s message. The message sent from a factor node \( f \) to a variable node \( x \), is the product of the local function \( f \) with all messages received at \( f \), except \( x \)'s message, summed over all variables but \( x \).

When performed on a cycle-free graph, the sum-product algorithm will find the exact
marginal functions in a number of iterations equal to the depth of the tree. The marginal function $F(x)$ for node $x$, is found by taking the product of all incoming messages at $x$ as in (2.16).

$$F(x) = \prod_{g \in n(x)} \mu_{g \rightarrow x}(x)$$

(2.16)

An alternate method of computing the marginal $F(x)$, is to take the product of any two messages being passed in opposite directions along an edge incident to $x$. This is true because a message passed from $x$ to $f$ is equal to the product of all incoming messages except the incoming message from $f$.

### 2.4.3 Belief Propagation and Loopy Belief Propagation

Belief Propagation is an instance of the sum-product algorithm introduced by Pearl in [34]. Belief Propagation aims to find the marginal or posterior probabilities from a joint probability distribution. As described in [35], the joint probability distribution function can be expressed as:

$$P(x) = \frac{1}{Z} \prod_{ij} \psi_{ij}(x_i, x_j) \prod_i \phi_i(x_i)$$

(2.17)

where $Z$ is a normalizing constant, and $\psi_{ij}(x_i, x_j)$ is a joint function, also called the potential function, of two connected nodes $x_i$ and $x_j$. This $\psi$ function is equivalent to the local functions, $f_k(x_k)$, described in the sum-product algorithm. The $\phi_i(x_i)$ function is the local, previously known, distribution for $x_i$, known as the prior probabilities.

In belief propagation, factor nodes are not present, and there is only one type of message. The message update for the message sent from node $x_i$ to node $x_j$ is as follows:

$$m_{ij}(x_j) \leftarrow \sum_{x_i} \phi_i(x_i) \psi_{ij}(x_i, x_j) \prod_{k \in n(x_i) \setminus j} m_{ki}(x_i)$$

(2.18)

This equation can be obtained from the sum-product algorithm by substituting (2.14) into (2.15). The only difference is the presence of the prior probabilities, however these can be added to the sum-product algorithm by connecting a prior probability function node to each variable node.
In belief propagation, the marginal functions computed are known as “beliefs”. The belief of a variable \( x_i \), is the current posterior probability distribution of \( x_i \), given the joint probability distribution and evidence from surrounding neighbours, \( n(x_i) \). The belief of node \( x_i \) is computed as the product of the local evidence at \( x_i \) (prior probabilities) and all incoming messages into node \( x_i \):

\[
b_i(x_i) = \phi_i(x_i) \prod_{j \in n(x_i)} m_{ji}(x_i)
\]  

(2.19)

which is identical to the computation of the marginals in the sum-product algorithm.

As noted by Pearl in [34], if the graph is a cycle-free tree, then the beliefs converge to the exact marginals. The use of sum-product algorithm and belief propagation on graphs with cycles is known as loopy belief propagation, which is not guaranteed to converge [34]. Turbo decoding has demonstrated excellent empirical performance in coding theory [36], and has been found to be an instance of loopy belief propagation [37]. Further empirical studies found that loopy belief propagation often converges to a good approximation of the marginals, however in some instances fails to converge to anything useful [38]. Theoretically, loopy belief propagation was shown to be the minimization of the Bethe free energy approximation [39]. It was found that the minima of the Bethe free energy expression are equivalent to the stable fixed points in loopy belief propagation [40]. Necessary conditions for the convergence of loopy belief propagation are not known, however several sufficient conditions have been suggested in [41], [42], and [43]. Heske noted in [40] that loopy belief propagation can be interpreted as gradient-descent minimization, and non-convergence may be caused by a too-large step-size. Thus non-convergence can often be remedied by damping the updates.

In many applications, the exact marginals are not necessary, rather the optimal configuration of the random variables is required. For these scenarios, loopy belief propagation will likely converge to close enough approximations such that \( \arg \max_{x_n} b(x_n) = \arg \max_{x_n} P(x_n) \). Thus an estimate for the optimal configuration, \( x_n^* \), of each variable
can be approximated as follows:

\[ x_i^* = \arg \max_{x_i} \{ \phi_i(x_i) \prod_{j \in n(x_i)} m_{ji}(x_i) \} \quad (2.20) \]

### 2.5 Derivation of Affinity Propagation

This section presents a brief overview of the derivation of Dueck and Frey’s Affinity Propagation. For the detailed derivation, see [3]. Affinity propagation was derived using a factor graph and a loopy sum-product algorithm. Loopy sum-product algorithm is equivalent to loopy belief propagation, the only difference being in the way message passing is defined (sum-product algorithm has factor nodes and belief propagation does not). First, the clustering problem was described by the factor graph given in Figure 2.7.

Each data point involved in the clustering has a variable node, \( c_i \), where the value \( c_i = k \) means that data point \( i \) has chosen data point \( k \) as its exemplar. If \( c_k = k \), then data point \( k \) is an exemplar. The global function associated with this factor graph is the following constrained net similarity function:

\[
F(c) = e^{\sum_{i=1}^{N} s(i,c_i)+\sum_{k=1}^{N} \log f_k(c)} = \prod_{i=1}^{N} e^{s(i,c_i)} \cdot \prod_{k=1}^{N} f_k(c_1, c_2, \ldots, c_N) \quad (2.21)
\]

where \( c = \{c_1, c_2, \ldots, c_N\} \), and \( s(i, c_i) \) is the similarity between node \( i \) and its exemplar, \( c_i \). The function is exponentiated to ensure a positive function.

The first term is the net similarity function, and the second term is the coherence constraint, which promotes correct cluster configuration. The coherence constraint, which ensures that clusters cannot be formed without an exemplar, is as follows:

\[
f_k(c) = \begin{cases} 
0, & \text{if } \exists i : c_i = k \text{ and } c_k \neq k \\
1, & \text{otherwise}
\end{cases} \quad (2.22)
\]

This function will evaluate to zero if data point \( i \) chooses data point \( k \) to be its exemplar, but data point \( k \) is not an exemplar.
Figure 2.7: Factor graph for the Affinity Propagation algorithm [3].

The coherence constraint factors, $f_k(c)$ in (2.21), are functions of the set of variable nodes, $\{c\}$. These coherence constraint factors are the local functions of the sum-product algorithm, described in (2.9), and are represented by factor nodes in Figure 2.7. The net similarity factors, $e_{s(i,c_i)}$, are functions of only one variable node, $c_i$. These factors can be thought of as the prior probabilities from (2.17) and are represented in Figure 2.7 by an input factor node, $s(i,c_i)$, for each variable node, $c_i$.

Using the sum-product algorithm on the factor graph in Figure 2.7, the following message update rules were derived:

$$
\rho_{i \rightarrow k}(c_i) = e_{s(i,c_i)} \cdot \prod_{k' = 1 \atop k' \neq k}^N \alpha_{i \rightarrow k'}(c_i) \quad (2.23)
$$

$$
\alpha_{i \rightarrow k}(c_i) = \sum_{\sim\{c_i\}} f_k(c_1, c_2, \ldots, c_i, \ldots, c_N) \cdot \prod_{i' \neq i} \rho_{i' \rightarrow k}(c_{i'}) \quad (2.24)
$$

where $\rho_{i \rightarrow k}(c_i)$ is passed from variable node $i$ to function node $k$, and $\alpha_{i \rightarrow k}(c_i)$ is passed from function node $k$ to variable node $i$. Simplification was then performed on the messages, and the details of this simplification can be found in [3].

The actual message passing takes place in the log domain, where $e^{r(i,k)} = \rho_{i \rightarrow k}$ and $e^{a(i,k)} = \alpha_{i \rightarrow k}(c_i)$. The responsibility, $r(i,k)$, and availability, $a(i,k)$, are the familiar...
messages from Section 2.3.2. The belief at node $i$ is obtained by taking the product of all incoming messages at variable node $c_i$. The desired configuration is then obtained by finding the arg max of this expression for each node $c_i$, as in (2.20). After some simplification, the following formula is obtained:

$$\hat{c}_i = \arg \max_j [a(i, k) + r(i, k)]$$  \hspace{1cm} (2.25)

Upon convergence of the algorithm, the above expression is used to find the clustering configuration. The exemplars are the nodes where $\hat{c}_k = k$.

### 2.5.1 Max-Product Affinity Propagation

A *semiring* is an abstract algebraic structure, similar to a *ring*, but without the additive inverse requirement. A *commutative semiring* contains two binary operators, $+$ and $\cdot$, which are both commutative and which satisfy the distributive property. As presented in [44], message passing algorithms, such as belief propagation, may be performed on any commutative semiring. An alternate to the sum-product semiring is the max-product semiring, which was used to simplify the affinity propagation algorithm. In the max-product semiring, the sum-product message updates become the following:

$$\mu_{x \to f}(x) = \prod_{g \in n(x) \setminus f} \mu_{g \to x}(x) \quad \text{and} \quad \mu_{f \to x}(x) = \max_{\{X\} \setminus x} \left( f(X) \prod_{y \in n(f) \setminus x} \mu_{y \to f}(y) \right)$$  \hspace{1cm} (2.26)

The simple affinity propagation update equations, (2.4 – 2.7) were obtained by simplifying the max-product algorithm on the factor graph in Figure 2.7.

### 2.6 Chapter Summary

This chapter gives an introduction to clustering in MANETs and VANETs, and uses performance issues common to MANETs and VANETs as motivation for clustering. The MOBIC clustering algorithm was described in detail as a mobility-based clustering scheme.
for MANETs. MOBIC is a very popular clustering scheme, and is commonly used as a benchmark for clustering performance. The ample research in cluster-based VANETs has been focused on the development of cluster-based MAC and routing protocols, as opposed to the underlying clustering algorithms. A convincing and validated VANET clustering scheme, which considers vehicular mobility, has yet to be suggested.

This chapter also explores static data clustering algorithms, with a focus on the Affinity Propagation algorithm. The Affinity Propagation algorithm is explained in detail. To derive the Affinity Propagation, the concepts of factor graphs, sum-product algorithm, and belief propagation are introduced. Affinity Propagation is derived from loopy belief propagation, due to its cyclic factor graph, therefore the convergence of loopy belief propagation is discussed.

In the remaining chapters, affinity propagation is introduced from a communication’s perspective, where it is applied to vehicular ad hoc networks. Each node in the network acts as a data point, and the nodes perform the algorithm distributively by communicating the messages to one another. This introduces the problem where nodes will not have a complete picture of the network, thus convergence cannot be assured unless additional measures are taken. In addition, Affinity Propagation assumes that once clusters are found, the process is complete, however in a mobile network, it is necessary to provide continuous maintenance.

In the next chapter, the APROVE clustering scheme for VANET’s is introduced. In Chapter 5, the clustering performance of APROVE will be compared against MOBIC, due to the lack of validated VANET clustering schemes in the recent literature. This comparison is fair since MOBIC is well-established and works well in scenarios with group mobility, such as vehicles on a highway.
Chapter 3

APROVE Clustering Scheme for VANETs

This chapter proposes a novel clustering algorithm for vehicle ad hoc networks, which is named APROVE (Affinity PROpagation for VEhicular networks). The APROVE clustering technique uses concepts from Affinity Propagation from a communications perspective and in a distributed manner. Affinity Propagation, as presented in the Background, is a centralized, static data clustering technique designed to find the centers (exemplars) in a set of data. In APROVE, affinity propagation is adapted for clustering in VANETs. Due to the dynamic topology of VANETs, the APROVE clustering algorithm must be continuous. The mobile nodes use affinity propagation co-operatively to elect the most advantageous clusterheads. The goal of the clustering scheme is to create a small number of clusters with high stability, where stability is measured in terms of clusterhead duration, cluster member duration, and average rate of clusterhead change.

In APROVE, clusterheads use affinity propagation distributively to form clusters and elect suitable clusterheads. The high cluster stability is ensured by forming clusters where the nodes have close proximity and low relative mobility to one another. This is done by designing a similarity function that considers both vehicle position and mobility. In
this algorithm, each node in the network computes and transmits the responsibility and availability messages to each of its neighbours. The result is that each node is performing distributive affinity propagation with only the nodes in its one-hop neighbourhood. In normal affinity propagation, a central computer has access to all of the messages for every data point, and makes global cluster decisions for the entire set. In APROVE, each node possesses only the messages addressed to it from its one-hop neighbours. As a result, each node is forced to make an independent clustering decision with only the local information.

The independent and local clustering decisions made in APROVE raise convergence issues. It is possible for a node to choose another node as its clusterhead, but that node is not choosing itself as a clusterhead. This issue is addressed in APROVE by introducing
additional flags in the messages, which communicate the surrounding node’s clusterhead status and clustering intentions.

A simplified illustration of the APROVE algorithm is presented in Figure 3.1, where the strength of the message being passed is proportional to the intensity of the arrow. A node communicates its availability and responsibility messages to each of the neighbours in its range. As the algorithm progresses, received responsibilities and transmitted availabilities will increase for certain vehicles, which then gradually emerge as the clusterheads.

3.1 Similarity Function

To ensure cluster stability, APROVE takes the dynamic VANET topology into consideration during cluster formation. This is done by designing a similarity function, which is a function of both the vehicular position and mobility. The similarity function used in APROVE is a combination of the negative Euclidean distance between node positions now and the negative Euclidean distance between node positions in the future. As the similarity between nodes $i$ and $j$ is maximized, the proximity and the relative mobility between the nodes are both minimized. The similarity function between nodes $i$ and $j$ at time $k$ is stated below:

$$s_{i,j}(k) = - (\|x_i(k) - x_j(k)\| + \|x'_i(k) - x'_j(k)\|)$$  \hfill (3.1)

$$x_i(k) = \begin{bmatrix} x_i(k) \\ y_i(k) \end{bmatrix} \quad x'_i(k) = \begin{bmatrix} x_i(k) + v_{x,i}(k) \cdot \tau_f \\ y_i(k) + v_{y,i}(k) \cdot \tau_f \end{bmatrix}$$

where $x_i(k)$ is a vector of node $i$’s current position at time $k$, and $x'_i(k)$ is a vector of node $i$’s predicted future position. The function predicts each node $i$’s future position in $\tau_f$ seconds from now, based on node $i$’s current velocity $v_{x,i}(k)$ in the x direction and velocity $v_{y,i}(k)$ in the y direction. It should be noted that unlike the constant similarities
in static affinity propagation, the similarity function in APROVE changes with time, $k$. The varying similarities in APROVE are caused by the dynamic VANET topology.

### 3.1.1 Similarity Function Parameters

There are two main parameters associated with the similarities: the *Future Prediction Period*, and the input preferences. The *Future Prediction Period*, $\tau_f$, can be tuned for different types of mobility, which is discussed further in Chapter 4. The similarity function needs the set of input preferences (self-similarities), $s(i, i)$. The input preferences can be set to increase or decrease the probability of certain nodes becoming the clusterhead. For example, a large truck or bus is likely to cause shadow fading and the hidden terminal problem. If this truck or bus becomes the clusterhead, the shadowing and hidden terminal issues are solved. This is done by giving higher input preferences to trucks and buses. In this work, all vehicles were assumed to be the same size, and were thus given equal input preference.

In addition to giving preference to specific nodes during clusterhead election, the self-similarities control how many clusters are produced. In MANET clustering, it is desirable to have a minimum number of clusters, whereby there are no clusterheads within range of one another. The cluster size is limited by the broadcast range, thus it is preferred to maximize the cluster size within the broadcast range of the clusterhead. To maximize the cluster size, the self-similarities should be minimized. The self-similarity preference that gives both a minimum number of clusters and maximum cluster stability is found through simulations.

### 3.2 Neighbour List and Message Updating

In this work, the APROVE messages are broadcast periodically with a period of $T_H = 1s$. The time periods are denoted by $k = \{0, 1, 2, \ldots\}$. The messages associated with the
current time period are defined as $r_{i,j}(k)$ and $a_{j,i}(k)$, whereas the messages associated with the previous time period are defined as $r_{i,j}(k-1)$ and $a_{j,i}(k-1)$.

Although node $i$ will have its current transmitted messages at time $k$, it will not always have the current received messages (due to delay, message error, collision, or asynchronous nodes). Node $i$ stores each of the neighbour’s last received messages, along with other pertinent information, in a neighbour list. The neighbour list contains all of the last received position, velocity, and clustering related messages regarding each neighbour.

For notation purposes, a message transmitted from $i$ to $j$ in current time slot $k$ is denoted, $r_{i,j}(k)$. This is opposed to the last received message at $i$ from $j$, which is stored in the neighbour list. This message is denoted $r(i,j)$, and it does not contain information about time period $k$. The timing information is omitted since the exact time this last message was received varies based on delay, collision, and synchronization.

### 3.2.1 Neighbour List

Every node $i$ will maintain a neighbour list, $N_i$, which has a neighbour list entry, $N_i^j$, for every neighbour $j$. Each neighbour list entry, $N_i^j$ contains the following fields:

- $(x, y)_j$: last received position vector of node $j$
- $(v_x, v_y)_j$: last received velocity vector of node $j$
- $s(i, j)$: last computed similarity for $i$ and $j$
- $a(i, j)$: last availability received from $j$
- $a(j, i)$: last availability transmitted to $j$, (equivalent to $a_{j,i}(k-1)$)
- $r(j, i)$: last responsibility received from $j$
- $r(i, j)$: last responsibility transmitted to $j$, (equivalent to $r_{i,j}(k-1)$)
- $\text{CH}_{\text{convg},j}$: clusterhead converge flag for node $j$
- $\text{CH}_j$: clusterhead status flag for node $j$, (1: Clusterhead, 0: Cluster member)
- $t_{\text{expire}}$: time that node $j$ expires
The \( a(j, i) \) and \( r(i, j) \) entries are equivalent to \( a_{j,i}(k-1) \) and \( r_{i,j}(k-1) \) since these neighbour list entries are transmitted by \( i \), and are thus always known by node \( i \). As a result, node \( i \) can update these neighbour list entries at the end of every time period \( k \), for use in the succeeding time period, \( k + 1 \). Note that each node \( i \) should also keep a self-entry, \( N_i^i \), which keeps track of self-responsibility, self-availability, and node \( i \)'s cluster information.

### 3.2.2 Availability and Responsibility Update Rules

The availability and responsibility messages for APROVE are based on Affinity Propagation. The message update rules are as follows:

\[
\begin{align*}
    r_{i,j}(k) &= \alpha \left[ s(i, j) - \max_{j' \neq j} \{ a(i, j') + s(i, j') \} \right] + \beta \cdot r_{i,j}(k-1) \\
    a_{j,i}(k) &= \alpha \cdot \min \left\{ 0, r_{i,i}(k-1) + \sum_{j' \neq j,i} \max \{ 0, r(j', i) \} \right\} + \beta \cdot a_{j,i}(k-1) \\
    a_{j,j}(k) &= \alpha \cdot \sum_{j' \neq i} \max \{ 0, r(j, i) \} + \beta \cdot a_{i,i}(k-1)
\end{align*}
\]

where \( \alpha + \beta = 1 \). The \( \alpha \) and \( \beta \) factors provide damping on the message updates, which prevents oscillation in affinity propagation.

### 3.3 APROVE Message Passing

Two different types of message passing are presented for the APROVE algorithm. The first, naive approach, uses three different messages: HELLO, AVAIL, and RESP. The HELLO beacon contains position and velocity information, and the AVAIL and RESP beacons contain the availability and responsibility messages respectively. This version of the algorithm is published in [26], and is referred to as “segregated” message passing for the remainder of this work.

In an effort to reduce overhead, a second message passing procedure was developed.
The second version is referred to as “aggregated” message passing, and in this procedure, all of the clustering information is combined into one HELLO beacon.

### 3.3.1 Segregated Message Passing

Each node $i$ will periodically broadcast a HELLO beacon containing its ID, position, velocity and current clusterhead. The hello broadcast period is defined as $T_H$. Upon reception of a HELLO beacon from node $j$, node $i$ will calculate its current similarity with $j$, $s_{i,j}(k)$, using (3.1) and update its neighbour list, $s(i,j)$ with the new information.

In the highway scenario, a node should only cluster with vehicles that are driving in the same direction. Thus only nodes that are driving in the same direction as node $i$ are considered for clustering and added to the neighbour list. This also helps to reduce the overhead of the clustering, since a smaller number of neighbours are involved in the clustering process. This procedure is outlined in Procedure 1.

**Procedure 1** Broadcast and Reception of Hello Beacons

1. For every $k = T_H \cdot \{0, 1, 2, \ldots\}$ seconds, node $i$:
   
   Broadcasts HELLO beacon: $\langle j, (x, y)_j, (v_x, v_y)_j, \text{CH}_j(k) \rangle$

2. Upon reception of a HELLO beacon from neighbour, $j$, node $i$ checks if $j$ is travelling in the same direction, by comparing the velocity vector of node $j$, $(v_x, v_y)_j$ with the velocity vector of node $i$, $(v_x, v_y)_i$

3. If true, $i$ calculates similarity with $j$, $s_{i,j}(k)$ and stores in $s(i,j)$

4. Node $i$ adds/updates $j$’s neighbour list entry, $N_i^j$:
   
   $\langle j, (x, y)_j, (v_x, v_y)_j, s(i,j), t_{\text{expire}}, \text{CH}_j \rangle$

The broadcast period for availability and responsibility messages is defined as $T_M$. During the current time period, $k$, each node $i$ will calculate its responsibility with each
neighbour $j$, $r_{i,j}(k)$ using (3.3). The current responsibility, $r_{i,j}(k)$ is then stored in the neighbour list entry, $r(i, j)$ for use in the next time period, $k + 1$. The damping factors used in (3.3) and throughout this work are set as $\alpha = \beta = 0.5$. Node $i$ then accumulates $r_{i,j}(k)$ for each neighbour $j$ in the responsibility array, $\mathbf{R}_i$, and broadcasts the array in the RESP packet.

Immediately following the responsibility broadcast, each node $i$ will calculate its current availability with each neighbour $j$, $a_{j,i}(k)$, using the update equation (3.4). Node $i$ will store $j$’s current availability in $a(j, i)$ of the neighbour list, for use in the next time period. The availabilities for each neighbour $j$ are accumulated in the availability array, $\mathbf{A}_i$. This array is broadcast in the AVAIL packet. For both responsibility and availability arrays, each individual message from neighbour $j$ is stored in the next available array index, $n$. The node ID associated with the $n^{th}$ element of the responsibility and availability arrays is stored in the index array, $\mathbf{I}_i$, at this same index $n$. This index array maps the array indices to their associated node IDs. For example if node 1 has neighbours $[3, 5, 2, 8, 10]$, its transmitted arrays are as follows:

$$\mathbf{R}_1 = \begin{bmatrix} r_{1,3}(k) & r_{1,5}(k) & r_{1,2}(k) & r_{1,8}(k) & r_{1,10}(k) \end{bmatrix}$$

$$\mathbf{A}_1 = \begin{bmatrix} a_{3,1}(k) & a_{5,1}(k) & a_{2,1}(k) & a_{8,1}(k) & a_{10,1}(k) \end{bmatrix}$$

$$\mathbf{I}_1 = \begin{bmatrix} 3 & 5 & 2 & 8 & 10 \end{bmatrix}$$ (3.5)

Node $i$ must also calculate its self-responsibility, $r_{i,i}(k)$ using (3.3), and self-availability, $a_{i,i}(k)$ using (3.4). These values should be stored in the self-entry, $N_i^1$, in the neighbour list. The AVAIL packet also includes a convergence flag, $\text{CH}\text{cnvg}(k)$. Due to the nature of the affinity propagation algorithm, a node’s self-responsibility plus self-availability will become positive when it has converged to clusterhead status. Convergence of APROVE and justification for the $\text{CH}\text{cnvg}$ flag are discussed further in Chapter 4. For every iteration of the algorithm, each node $i$ checks if $r_{i,i}(k) + a_{i,i}(k) > 0$, and then sets the $\text{CH}\text{cnvg}(k)$ flag accordingly. This flag indicates to $i$’s neighbour nodes whether or not
Chapter 3. APROVE Clustering Scheme for VANETs

**Procedure 2** Broadcast of RESP and AVAIL messages
For every $k = T_M \cdot \{0, 1, 2, \ldots \}$ seconds, each node $i$ will:

1. Calculate responsibility, $r_{i,j}(k)$ for each neighbour $j$:
   
   $$r_{i,j}(k) = 0.5 \cdot [s(i, j) - \max_{j' \neq j} \{ a(i, j') + s(i, j') \}] + 0.5 \cdot r_{i,j}(k - 1)$$

2. Store responsibilities in array at index, $n$: $R_i[n] = r_{i,j}(k)$

3. Calculate availability, $a_{j,i}$ for each neighbour $j$:
   
   $$a_{j,i}(k) = 0.5 \cdot \min \left\{ 0, r_{i,i}(k - 1) + \sum_{j' \neq j,i} \max \{ 0, r(j', i) \} \right\} + 0.5 \cdot a_{j,i}(k - 1)$$

4. Store availabilities in array at index, $n$: $A_i[n] = a_{j,i}(k)$

5. Store the ID of $j$ in the index array at index, $n$: $I_i[n] = j$

6. Determine if converged to CH status:
   
   if $r_{i,i}(k) + a_{i,i}(k) > 0$, then set $CH_{cnvg}(k)$ accordingly

7. Broadcast the RESP packet: $\langle R_i, I_i \rangle$

8. Broadcast the AVAIL packet: $\langle A_i, I_i, CH_{cnvg}(k) \rangle$

they can consider node $i$ as a potential clusterhead when they make their clusterhead decisions. The responsibility and availability broadcast procedure is outlined in Procedure 2.

When node $i$ receives a RESP or AVAIL packet from $j$, it searches for its ID in the index array, $I_j$. If found, it will read its specific responsibility and availability messages from the $R_j$ and $A_j$ arrays. These message are stored in the received message fields, $r(j, i)$ or $a(i, j)$ of $j$’s neighbour list entry, $N_j^i$. If the received packet is of AVAIL type, node $i$ will also update the $CH_{cnvg,j}$ field for $j$ according to the $CH_{cnvg}$ flag received. This routine is summarized in Procedure 3.
Procedure 3 Reception of \textit{RESP} and \textit{AVAIL} messages

Upon reception of a \textit{RESP} or \textit{AVAIL} packet from node $j$, node $i$ will:

1. Search for its ID in the $I_j$ index array.

2. If found, $I_j[n] = i$, read off the appropriate responsibility or availability messages at $R_j[n]$ or $A_j[n]$

3. Update the $r(j,i)$ or $a(i,j)$ field in the neighbour list entry, $N_{i}^{j}$

4. If \textit{AVAIL} packet, update CH$_{cnvg,j}$ field in $j$’s neighbour list entry, $N_{i}^{j}$

3.3.2 Aggregated Message Passing

In the above segregated message passing, the hello beacons are broadcast periodically with period $T_H$, and the responsibility and availability beacons are broadcast with period $T_M$. A logical improvement is to make $T_H = T_M$ and aggregate the messages into one broadcast beacon. This aggregated message passing reduces collisions, overhead, and simplifies the clustering procedure.

In aggregated message passing, each node $i$ will periodically broadcast a HELLO beacon containing all of the necessary information for the APROVE algorithm, with a period of $T_H$. The HELLO packet for node $i$ contains: ID, position, velocity, current clusterhead index, responsibility and availability for each of $i$’s neighbouring nodes, and clusterhead converge flag. Node $i$ will calculate its responsibility for each neighbour $j$, $r_{i,j}(k)$, using (3.3) and its availability for each neighbour $j$, $a_{j,i}(k)$, using (3.4). Node $i$ then stores $r_{i,j}(k)$ and $a_{j,i}(k)$ in the responsibility array, $R_i$, and availability array, $A_i$, respectively. These values are stored in the arrays at the next available index $n$, and the node ID is stored in the index array, $I_i$, at the same index $n$ as shown in (3.5). Next the self-responsibility and self-availability are calculated with (3.3) and (3.4) respectively, and the CH$_{cnvg}(k)$ flag can be set as in the previous section. The complete HELLO broadcast
Chapter 3. APROVE Clustering Scheme for VANETs

**Procedure 4** Broadcast of HELLO Beacons
For every $k = T_H \cdot \{0, 1, 2, \ldots\}$ seconds, each node $i$ will:

1. Calculate responsibility, $r_{i,j}(k)$ for each neighbour $j$ with (3.3), $(\alpha = \beta = 0.5)$.

2. Calculate availability, $a_{j,i}(k)$ for each neighbour $j$ with (3.4), $(\alpha = \beta = 0.5)$.

3. Store responsibility for neighbour $j$, in responsibility array: $R_i[n] = r_{i,j}(k)$

4. Store availability for neighbour $j$, in availability array: $A_i[n] = a_{j,i}(k)$

5. Store the ID of $j$ in the index array: $I_i[n] = j$

6. Determine if converged to CH status:
   
   if $r_{i,i}(k) + a_{i,i}(k) > 0$, then set $CH_{cnv}(k)$

7. Broadcast HELLO beacon:
   
   $\langle j, (x, y)_j, (v_x, v_y)_j, CH_j(k), R_i, A_i, I_i, CH_{cnv}(k) \rangle$

procedure is outlined in Procedure 4.

Upon reception of a HELLO beacon from node $j$, node $i$ will calculate its current similarity with $j$, $s_{i,j}(k)$, using (3.1) and update the position, velocity and similarity fields in $j$’s neighbour list entry. A node only considers neighbours moving in the same direction, and ignores broadcasts from traffic in the opposite direction. Node $i$ then searches for its ID in the index array, $I_j$. If found, it will read its specific responsibility and availability messages from the $R_j$ and $A_j$ arrays. These message are then stored in the received message fields, $r(j,i)$ or $a(i,j)$ of $j$’s neighbour list entry, $N^j_i$. Node $i$ will also update the CH$_j$ and CH$_{cnv,j}$ flags in $j$’s neighbour list entry. This routine is summarized in Procedure 5.

The above aggregated message passing is an improvement over the segregated message passing in terms of overhead, collisions, and simplicity. The reduced number of messages passed each $T_H$ leads to a reduced number of collisions, which in turn improves overall
Procedure 5 Reception of HELLO Beacons

Upon reception of a HELLO packet from node $j$, node $i$ will:

1. Check if $j$ is travelling in the same direction. If false, do nothing.

2. Else, calculate its similarity with $j$, $s_{i,j}(k)$.

3. Search for its ID in the $I_j$ index array. If found, $I_j[n] = i$, read off the appropriate responsibility and availability messages at $R_j[n]$ and $A_j[n]$.

4. Check if $CH_{cnvg}$ flag is set

5. Add/update $j$’s neighbour list entry, $N_j^i$:
   
   $\langle j, (x,y)_j, (v_x,v_y)_j, s(i,j), CH_j, a(i,j), r(j,i), t_{expire}, CH_{cnvg} \rangle$

clustering performance. As a result of these improvements, the remaining analysis and simulations in this work will be performed on the aggregated message passing algorithm.

3.3.3 Behaviour of Message Passing

There are some important notes regarding the passing of availability and responsibility messages. First, the messages are not reset once cluster decisions are made, which gives memory to the algorithm and provides preference to previous clusterheads. This feedback results in less frequent cluster changes. Second, the message passing does not have to be fully synchronous. Assuming no channel error or collisions, if every node broadcasts messages with a period of $T_H$, no matter when the messages are transmitted, the received messages and thus neighbour list entries, will be at most one $T_H$ old. A vehicle’s change in mobility and position over one $T_H$ is small, thus the algorithm’s performance will not be effected. If channel error is introduced, the neighbour list entries will become outdated, leading to performance degradation. The effects of channel error are discussed in Chapter 4. Although synchronization is not required for the message passing in APROVE, it may
be required when making clusterhead decisions, as discussed in the next section.

3.4 APROVE Clusterhead Selection and Maintenance

This section discusses procedures regarding cluster formation and cluster maintenance. The first method introduces a Clustering Interval time, $CI$, whereby all nodes will make their clusterhead decisions every $CI$ seconds. This method requires some synchronization amongst nodes, so that the cluster decisions are all made in the same time period. This Clustering Interval method of clusterhead selection is the method published in [26]. A slight variation of this first method is to add clusterhead contention in between the main clustering decisions, which reduces the average number of clusters. The final method introduced, solves the problem of synchronization by removing the Clustering Interval, allowing it to operate completely asynchronously. Synchronization is discussed further in Chapter 4.

The clusterhead decision and maintenance procedures discussed in this section, involve several different clusterhead flags and fields. To avoid confusion amongst them, these various fields are summarized in Table 3.1.

3.4.1 APROVE with Clustering Interval, $CI$

In this first method of clusterhead selection, clustering decisions are made periodically with a period of $CI$ called the Clustering Interval. This clustering interval must be large enough to allow the algorithm to converge, however affinity propagation converges very quickly with a small number of data points. Simulation trials established that affinity propagation will converge in less than 10 iterations with an average neighbourhood of 40 nodes. Therefore a $T_H$ of 1s, requires a minimum $CI$ of 10s.

Every $CI$ seconds, if node $i$’s $CH_{cnvg,i} = 1$ then node $i$ becomes a clusterhead,
myCH

\(_i = i\). Otherwise, node \(i\) chooses its clusterhead as follows:

\[
\forall N_j \in N_i: \text{CH}_{cnvg,j} = 1, \quad \text{myCH}_i = \operatorname{arg\ max}_{j \neq i} \{a(i,j) + r(i,j)\}
\] (3.6)

Node \(i\) finds its neighbour with the maximum received availability plus transmitted responsibility, but it only considers its neighbours with the CH\(_{cnvg,j}\) flag set. If a node sets its CH\(_{cnvg,j}\) flag, it is definitely going to choose itself as a clusterhead, thus this flag ensures that node \(i\) will pick a valid clusterhead. In the event that none of the neighbours have set their CH\(_{cnvg,j}\) flag, node \(i\) becomes its own clusterhead. Each node \(i\) keeps track of its current clusterhead, \(j\) with myCH\(_i = j\). If a node \(i\) becomes a clusterhead itself, then it sets its current clusterhead status flag, CH\(_i = 1\).

In between Clustering Intervals, cluster maintenance is performed with a period of \(T_{CM}\), to ensure the validity of each node’s current clusterhead. Every \(T_{CM}\), node \(i\) purges its neighbour list of old entries by checking the \(t_{expire}\) fields. Next, node \(i\) checks the status of its clusterhead. If \(\text{myCH}_i = j\), and node \(j\) was purged or is not currently a clusterhead (CH\(_i = 0\)), then node \(i\)’s clusterhead has been lost. If the clusterhead has been lost, node

<table>
<thead>
<tr>
<th>Field</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CH(_{cnvg,i})</td>
<td>Clusterhead Convergence Flag</td>
<td>Indicates when node (i) has converged to clusterhead status. If CH(_{cnvg,i}) is set, node (i) may not currently be a clusterhead, but it will become a clusterhead when the next clustering decisions are made.</td>
</tr>
<tr>
<td>CH(_i)</td>
<td>Current Clusterhead Status Flag</td>
<td>Indicates if node (i) is currently a clusterhead. 1: node (i) is currently a clusterhead 0: node (i) is not currently a clusterhead</td>
</tr>
<tr>
<td>myCH(_i)</td>
<td>My Current Clusterhead</td>
<td>Indicates the index of node (i)’s current clusterhead. If myCH(_i = j), node (j) is the clusterhead of (i)</td>
</tr>
</tbody>
</table>

Table 3.1: A summary of the different clusterhead fields
Procedure 6 Neighbour List and CH Purge
For each node $i$, and for $k = T_{CM} \cdot \{0, 1, 2, 3 \ldots \}$ seconds:

\[
\text{lost}_{CH} \leftarrow 0
\]

\[
\text{for all } N^j_i \in N_i \text{ do}
\]

\[
\text{if } \text{myCH}_i(k) = j \text{ then}
\]

\[
\text{if } CH_j = 0 \text{ or } N^j_i \text{ expired then}
\]

\[
\text{lost}_{CH} \leftarrow 1
\]

\[
\text{end if}
\]

\[
\text{end if}
\]

\[
\text{if } N^j_i \text{ expired then}
\]

\[
\text{Purge } N^j_i \text{ from } N_i
\]

\[
\text{end if}
\]

\[
\text{end for}
\]

$i$ chooses a new one from the current clusterheads as follows:

\[
\forall N^j_i \in N_i : CH_j = 1, \quad \text{myCH}_i = \arg \max_j \{a(i, j) + r(i, j)\} \quad (3.7)
\]

The $CH_{cnvg,j}$ flag is not used here because it indicates the potential clusterheads for the next round of clustering decision, not the current clusterheads. If node $i$ can not find another neighbour that is currently a clusterhead, it becomes its own clusterhead.

The clusterhead election and maintenance process is divided into two procedures. Procedure 6 purges expired nodes from the neighbour list, and checks the validity of the current clusterhead. This procedure is followed immediately by Procedure 7, where clusterhead selections are made. In this procedure, if it is a $CI^{th}$ iteration, nodes choose a clusterhead using (3.6). In between $CI$ iterations, if a node’s current CH is lost, it chooses a CH with (3.7), or else becomes its own clusterhead. These procedures are presented in pseudo code to provide a more thorough description of clustering decisions.
Procedure 7 Clusterhead Selection

For each node $i$, and for $k = T_{CM} \cdot \{0, 1, 2, 3 \ldots \}$ seconds:

if $k = m \cdot CI$ for $m = \{0, 1, 2, \ldots \}$ then

    if $CH_{cnvg,i} = 1$ then
        $myCH_i(k) \leftarrow i$
    else
        for all $N_j^i \in N_i : CH_{cnvg,j} = 1$ do
            $myCH_i(k) \leftarrow \text{arg max}_j \{a_{ij}(k) + r_{ij}(k)\}$
        end for
    end if

else

    if $\text{lost}_{CH} = 1$ then
        for all $N_j^i \in N_i : CH_j = 1$ do
            $myCH_i(k) \leftarrow \text{arg max}_j \{a_{ij}(k) + r_{ij}(k)\}$
        end for
        if $\forall N_j^i \in N_i, CH_j = 0$ then
            $myCH_i(k) = i$
        end if
    end if

end if

3.4.2 APROVE with Clusterhead Contention

The main issue concerning the above clustering decision method, is with the number of clusters that are formed. As the $CI$ is increased, nodes have a high probability of losing their current clusterhead, and are forced to pick a new one. During the cluster maintenance phase, if a node cannot find another clusterhead within range, it will become its own clusterhead. Once a node becomes its own clusterhead, it cannot join another cluster until the next clustering interval. Thus as $CI$ is increased, there is an increased
opportunity for new clusterheads to form, and there is no method to remove the un-
wanted clusterheads. This next procedure borrows a common mechanism from MANET
clustering schemes called clusterhead contention. Here, clusterheads that come within
reach of one another for more than a given contention period, will contend for clusterhead
status.

In the message passing procedures discussed above, a convergence flag, $CH_{cnvg,i}$ was
included in node $i$’s broadcast beacons, where $CH_{cnvg,i}$ indicated if $r_{ii}(k) + a_{ii}(k) > 0$.
If clusterhead contention is desired, it is necessary to instead send the actual value,
$CH_{cnvg,i} = r_{ii}(k) + a_{ii}(k)$, rather than just a flag. The full $CH_{cnvg,i}$ value is necessary
because this value will be used to determine the winning clusterhead during clusterhead
contention. For example, if clusterhead $i$ sees that clusterhead $j$ has been in range of
it for more than the Cluster Contention Time (CCT), $i$ makes a decision on whether or
not to remain a CH. If $CH_{cnvg,i} < CH_{cnvg,j}$, then $i$ relinquishes its clusterhead status,

Procedure 8 Neighbour List and CH Purge with Clusterhead Contention

For each node $i$, and for $k = T_{CM} \cdot \{0, 1, 2, 3 \ldots \}$ seconds:

\[
\text{\cdots In addition to Procedure 6}
\]

\[
\text{for all } N_i^j \in N_i \text{ do}
\]

\[
\text{if } CH_i(k) = 1 \text{ and } CH_j(k) = 1 \text{ then}
\]

\[
CCT_j \leftarrow CCT_j + 1
\]

\[
\text{end if}
\]

\[
\text{if } CCT_j \geq 5 \text{ s then}
\]

\[
CCT_j \leftarrow 0
\]

\[
\text{if } CH_{cnvg,i} < CH_{cnvg,j} \text{ then}
\]

\[
\text{lost}_{CH} \leftarrow 1
\]

\[
\text{end if}
\]

\[
\text{end if}
\]

\[
\text{end for}
\]
otherwise it remains unchanged. If $i$ has to relinquish its clusterhead status, then it will choose a new clusterhead using the clusterhead selection rules described in Procedure 7. The cluster members that used to belong to $i$ will see that $i$ has reset its CH status flag, $\text{CH}_i = 0$, which will cause them to make new clusterhead decisions as well. This process is described in Procedure 8, which is meant to be an extension to Procedure 6. Here, the clusterhead contention time is assumed to be 5 seconds, and $\text{CCT}_j$ is the contention timer for each neighbour node $j$. The clusterhead selection process is unchanged from Procedure 7.

### 3.4.3 Asynchronous APROVE

Since synchronization of nodes in a MANET is often a difficult requirement to maintain, a third clustering method is introduced, which is completely asynchronous. This is achieved by removing the clustering interval requirement and using the clusterhead contention method introduced above. Instead of $\text{CI}$ controlling when clusterhead decisions are made, node $i$’s $\text{CH}_{\text{cnvg},i}$ value indicates when $i$ should assume or relinquish clusterhead status. If node $i$ is without a clusterhead, and its $\text{CH}_{\text{cnvg},i} > 0$, then node $i$ will become a clusterhead. If node $i$ has no clusterhead, and its $\text{CH}_{\text{cnvg},i} < 0$, then it will choose the best current clusterhead in its neighbour list as in (3.7). In the event that no current clusterheads are found, node $i$ becomes its own clusterhead. When two clusterheads come within range of one another for more than $\text{CCT}$, then the clusterhead with the lesser $\text{CH}_{\text{cnvg}}$ value relinquishes its CH status. The final difference between this method and previous methods, involves the hand-over of clusterhead status. If a node $i$ belongs to clusterhead $j$, but over time $\text{CH}_{\text{cnvg},i}$ becomes greater than $\text{CH}_{\text{cnvg},j}$, then it can be inferred that node $i$ is taking over the clusterhead role. In this case, node $i$ drops node $j$ as its CH and becomes its own clusterhead. Shortly after, cluster contention will occur, and node $j$ will relinquish its CH role. The neighbour list and CH purge procedure is presented in Procedure 9, and the cluster selection process is in Procedure 10.
Procedure 9 Neighbour List and CH Purge for Asynchronous APROVE

For each node \(i\), and for \(k = T_{CM} \cdot \{0, 1, 2, 3\ldots\}\) seconds:

\[
\begin{align*}
\cdots & \text{ In addition to Procedure 6} \\
& \text{ for all } N_j^i \in N_i \text{ do} \\
& \quad \text{ if } CH_i(k) = 1 \text{ and } CH_j(k) = 1 \text{ then} \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \qu
Chapter 3. APROVE Clustering Scheme for VANETs

Procedure 10 Clusterhead Selection for Asynchronous APROVE
For $k = T_{CM} \cdot \{0, 1, 2, 3 \ldots \}$ seconds:

if $\text{lost}_{CH} = 1$ then
  if $\text{CH}_{\text{cnvg},i} > 0$ then
    myCH$_i(k) = i$
  else
    for all $N_j^i \in N_i : \text{CH}_j(k) = 1$ do
      myCH$_i(k) \leftarrow \arg \max_j \{a_{ij}(k) + r_{ij}(k)\}$
    end for
    if $\forall N_j^i \in N_i, \text{CH}_j(k) = 0$ then
      myCH$_i(k) = i$
    end if
  end if
end if

of clusterhead selection and maintenance. The first method uses Clustering Intervals, which requires some synchronization amongst nodes. The second method is a simple extension to this first method, where clusterheads that are within range of one another can contend for clusterhead status. This has the effect of reducing the average number of clusters, but also reducing clusterhead duration. The final method uses the cluster contention idea to create a completely asynchronous version of APROVE.

In the following chapter, APROVE will be studied in terms of overhead, convergence, channel error, and synchronization.
Chapter 4

Analysis of APROVE Algorithm

This chapter provides some insight into the operation of the APROVE algorithm. First, analogies are drawn between Pearl’s Belief Propagation and affinity propagation, where the concepts of likelihood, prior probability, and posterior probability are introduced. Next, APROVE’s parameter settings are discussed including: Self-similarity, $s(i, i)$, Clustering Interval, $CI$, and Future Prediction Period, $\tau_f$. The overhead of APROVE is shown to be reasonable by comparing APROVE’s overhead to the overhead of MOBIC. Convergence of APROVE is discussed in terms of both coherency of clusterhead decisions, and oscillation during message updates. The behaviour or APROVE in the presence of channel error is discussed, and APROVE’s robustness is argued. Finally, the synchronization requirements of the different formulations of APROVE are presented in more detail.

4.1 Affinity Propagation as Pearl’s Belief Propagation

Several of the justifications presented in this chapter benefit from viewing the affinity propagation algorithm from the perspective of Pearl’s belief propagation [34]. In [34], Pearl states that the local evidence at node $x$ is described by the likelihood function,
\( \lambda(x) = P(e \mid x) \), and the prior probability of \( x \) is \( \pi(x) = P(x) \). When the node \( x \) belongs to a tree, \( \lambda(x) \) and \( \pi(x) \) become functions of messages passed from \( x \)'s children and parents respectively. In belief propagation, the likelihoods are propagated up a tree, and the prior probabilities are propagated down a tree. As the algorithm progresses, the outgoing likelihood, \( \lambda(x) \) is a function of the incoming likelihoods \( \lambda(y_i) \) from each of \( x \)'s children, \( y_i \). The outgoing \( \pi(x) \) is a function of incoming \( \pi(u_i) \)'s from each of \( x \)'s parents, \( u_i \). At any iteration of belief propagation, the current beliefs are found by fusing \( \lambda(x) \) and \( \pi(x) \) as follows:

\[
BEL(x) = \alpha \lambda(x) \pi(x) \quad (4.1)
\]

where \( \alpha \) is a normalizing constant, \( \lambda(x) \) is the propagated likelihood received at \( x \), and \( \pi(x) \) is the current prior probability transmitted from \( x \). The computed beliefs are the posterior probabilities, where \( BEL(x) = P(x \mid e) \).

As discussed in the background, the local evidence that node \( j \) is the clusterhead of node \( i \), \( c_i = j \), is equal to the similarity function, \( s(i, j) \). Thus the similarity function can be thought of as the input log-likelihood function that node \( j \) is the clusterhead of node \( i \). In addition, the self-similarity, \( s(i, i) \), can be thought of as the prior log-probabilities for node \( i \). Notice that affinity propagation operates in the log-domain, thus log-probabilities are used. In affinity propagation, these log-likelihoods and log-prior probabilities are propagated through the network with the availability and responsibility messages. To find the belief that node \( j \) is the clusterhead of node \( i \), the incoming responsibility \( r(i, j) \) is combined with the outgoing availability \( a(i, j) \).

### 4.2 Parameter Selection

This section provides some insight into APROVE’s parameter selection. The parameters that must be set before APROVE can operate are the self-similarities, Future Prediction Period, and the Clustering Interval (if not in asynchronous mode). The self-similarities
control the number of clusters produced, as well as the input preference for a node to achieve clusterhead status. The Clustering Interval is used in the first two formulations of APROVE, and it determines the period of clusterhead election. The Future Prediction Period is present in the similarity function, and it should be tuned to the network’s mobility pattern.

4.2.1 Self-Similarities (Input Preferences)

As discussed above in the comparison to belief propagation, the similarity function $s(i, j)$ represents the log-likelihood that node $j$ is the clusterhead of node $i$, and the self-similarity $s(i, i)$ represents the prior probability that node $i$ is a clusterhead. Given these relationships, the effects of the similarity function and input preferences are apparent. In APROVE, the similarity function is a negative function with a maximum value of 0. A similarity close to 0 corresponds to a likelihood close to 1, which is achieved when two vehicles have a small distance and low relative velocity between them. On the other hand, a highly negative similarity corresponds to a likelihood close to 0, which is achieved when two vehicles have a large distance and a high relative velocity between them. Clearly the closer vehicle $j$ is to vehicle $i$ in terms of proximity and velocity, the higher the likelihood that node $j$ is a clusterhead of node $i$.

The self-similarity input preferences should be set on the same scale as the similarity function. Assigning a certain node a higher input preference, gives this node a higher prior probability of being a clusterhead. The simulations performed in this work were done with equal input preference values, thus no vehicle has an initial advantage for clusterhead status. The self-similarities also affect the number of clusters that are produced by the algorithm. This behaviour can be explained by the prior probability relationship. If all nodes are given high input preferences (close to 0), then all nodes have a high probability of becoming a clusterhead (close to 1), thus all nodes will become their own clusterhead. In APROVE, low input preferences are assigned to all the vehicles to give
the smallest possible number of clusters. With low input preferences, all vehicles have a low probability of becoming a clusterhead, and the likelihoods will propagate through the algorithm, resulting in only one vehicle achieving a high posterior probability of clusterhead status. This results in the maximization of the cluster size within the 250m broadcast range of each clusterhead.

4.2.2 Future Prediction Period: $\tau_f$

APROVE’s second parameter is the Future Prediction Period, $\tau_f$, used in the similarity function (3.1). Future Prediction Period is used by the similarity function to predict the future position of a vehicle given the current velocity. To select a specific $\tau_f$ is to assume that individual vehicles will remain at a relatively constant speed for $\tau_f$ seconds. Thus, this parameter should be tuned to the network’s mobility pattern. The $\tau_f$ parameter is dependent on both the variance in speed per individual vehicle and the variance in speed from one vehicle to another. As the variance in speed per individual vehicle increases, the $\tau_f$ should be decreased, since the vehicle’s speed is becoming less constant. On the other hand, as the variance in speed from one vehicle to another is increased, the velocity of individual cars should play a larger role in clustering, thus the $\tau_f$ should be increased.

The $\tau_f$ parameter was tuned to the VANET highway scenario in the Chapter 5 simulations.

4.2.3 Clustering Interval: $CI$

The final parameter used by APROVE is the Clustering Interval, $CI$, which determines the period of cluster formation. This parameter is used in the first two formulations of APROVE, but is not used in the Asynchronous formulation.

The first formulation described in Chapter 3, uses the $CI$ parameter to determine how often the nodes elect a new clusterhead based on the affinity propagation messages being passed in the background. Every $CI$ seconds, each node $i$ examines its neighbours
for nodes that have converged to clusterhead status ($CH_{cnvg} > 0$), and then chooses the node $j$ that maximizes $r(i, j) + a(i, j)$. If $CI$ is increased, the cluster head duration will naturally increase, but on the other hand, the number of clusters will increase as well. With a long $CI$, clusterheads elected at the beginning of the interval may no longer be desirable clusterheads at the end. This results in cluster members drifting away from their current clusterheads, forcing them to select new cluster heads during the cluster maintenance phase. If a node is unable to find a replacement cluster head amongst its neighbour list, it will become its own cluster head, and remain as such until the next cluster interval arrives. Thus the longer the $CI$, the greater the opportunity for new clusters to form. This trend is observed in the simulations, and is reported in Chapter 5.

The increased number of clusters, which results from an increased $CI$, can be solved by introducing clusterhead contention. The second affinity propagation formulation presented in Chapter 3, uses clustering intervals with clusterhead contention. Clusterhead contention occurs during the cluster maintenance phase, whereby two clusterheads that come within range of one another contend for clusterhead status. The clusterhead with the higher $CH_{cnvg}$ value remains a clusterhead. This has the effect of maintaining the most desirable clusterheads, while dropping the undesirable, excess clusterheads.

### 4.3 Overhead Analysis

In Chapter 3, both segregated and aggregated message passing procedures were introduced. The main motivation for creating the aggregated message passing algorithm was to reduce overhead. In the segregated message passing algorithm, each node sends out three different messages each second, each of which requires an IP and MAC header. By making the broadcast period of the hello messages equal to the broadcast period of the responsibility and availability messages, the aggregated message passing algorithm was possible.
In this section, the overhead for APROVE with aggregated message passing and the overhead of MOBIC are presented. The resulting analysis shows that APROVE has a reasonable overhead, which is comparable to MOBIC’s overhead.

### 4.3.1 Overhead of aggregated message passing algorithm

Each APROVE HELLO beacon transmitted by node $i$ includes the IP and MAC headers, the position and velocity information, the cluster head status and convergence flags, and the responsibility, availability, and index arrays. Each position, velocity, responsibility, and availability value is assumed to occupy 4 bytes. In computing, a single precision float occupies 4 bytes, which gives a suitable precision for this application. APROVE’s overhead includes 20 and 58 bytes for the IP and MAC headers respectively, 4 bytes for each of the $x_i$ and $y_i$ positions, 4 bytes for each of the $v_{x,i}$ and $v_{y,i}$ velocities, 1 byte for the CH$_i$ and CH$_{cnvg,i}$ flags, and 4 bytes for each member of the availability, responsibility, and index arrays. Since these arrays are exactly as long as the neighbour list, we need 12 bytes for each member of the neighbour list. The size of an APROVE beacon is summarized below in (4.2).

$$\text{size}_{\text{APR}} = \text{IP_HDR_LEN} + \text{MAC_HDR_LEN} + \text{POS} + \text{VEL} + \text{CHINFO} + 12N_{\text{size}}$$

$$= 20 + 58 + 8 + 8 + 1 + 12N_{\text{size}}$$

$$= 95 + 12N_{\text{size}}$$

where $N_{\text{size}}$ is the length of the neighbour list.

### 4.3.2 Comparison to Overhead of MOBIC

The compared clustering algorithm, MOBIC, has a broadcast beacon similar to APROVE’s HELLO beacon. MOBIC’s broadcast beacon includes the IP and MAC headers, the node’s current status (CH, CM, or Undecided), the node’s aggregate mobility metric, and the index and status of each of the neighbours. The current status uses 1 byte, the mobility
metric uses 8 bytes, and 5 bytes are used for each member of the neighbour list (4 bytes for the neighbour index, and 1 byte for the neighbour status). The size of a MOBIC beacon is summarized below in (4.3).

\[
\text{size}_\text{MOB} = \text{IP_HDR_LEN} + \text{MAC_HDR_LEN} + \text{CH_STAT} + \text{MOBILITY} + 5N_{size} \\
= 20 + 58 + 1 + 8 + 5N_{size} \\
= 88 + 5N_{size} \tag{4.3}
\]

The size of the HELLO beacons, for both APROVE and MOBIC, increases as a function of neighbour list size. Although it is apparent from (4.2) and (4.3) that APROVE’s HELLO beacon is both larger and increasing at a greater rate than MOBIC’s, it is not concluded that APROVE’s overhead is greater. In the APROVE algorithm, each node periodically broadcasts a HELLO beacon with a period of 1s. In MOBIC, however, nodes make both periodic broadcasts every 1s, and event-based broadcasts. The event-based broadcasts occur every time a node decides to change its status to either cluster head or cluster member. This occurs in both cluster formation and cluster contention (when two clusters come within range and contend for the role of CH). These additional event-based broadcasts increase MOBIC’s overhead. In low to moderate density networks where the neighbour list size is reasonable, APROVE will have a lower overhead than MOBIC. In high-density networks, if the neighbour list size becomes large enough, APROVE will have a higher overhead than MOBIC. A plot comparing simulation results for the overhead of the two algorithms is presented in Chapter 5.

### 4.4 Convergence Analysis

In this section, two definitions of convergence for APROVE are presented and discussed. The first refers to the convergence of the underlying affinity propagation algorithm to non-oscillating states. Non-oscillation of affinity propagation depends on the convergence
of the sum-product algorithm on a loopy factor graph. The second definition is the convergence of APROVE to coherent clusterhead states. Clusterhead decisions are coherent if all nodes selected as clusterheads, are actually clusterheads. Since decisions are local and distributed, it is necessary to take some steps to ensure coherency.

4.4.1 Convergence of APROVE to Non-Oscillating States

Since affinity propagation is derived on a loopy factor graph, convergence to the marginals cannot be guaranteed. Several sufficient conditions for the convergence of loopy sum-product algorithm were presented in [43] and [42]. Unfortunately, affinity propagation does not satisfy these conditions, and convergence cannot be guaranteed. When affinity propagation fails to converge, oscillations occur in the beliefs. In [40], Heske notes that non-convergence of loopy belief propagation can be interpreted as a too-large step-size in gradient-descent minimization, which can be solved by message damping. This approach is also suggested in [2], where responsibility and availability messages are damped with previous iterations. In APROVE, oscillations are avoided by using damping factors of $\lambda = \beta = 0.5$.

Another cause of oscillation in affinity propagation, as stated in [2], is the presence of degenerate cases. Degeneracies lead to multiple minima, which prevent convergence of the algorithm. For example, if clustering is performed on two nodes with the exact same similarities and input preferences, then the algorithm will oscillate, with the clusterhead role alternating between the nodes. Frey suggests adding a small jitter to the similarities to prevent degenerate scenarios [2]. APROVE does not suffer from degeneracies, because of its time-varying similarity function. The variations in the similarities caused by the dynamic topology, adds a built-in noise, which prevents degeneracies and oscillations.
4.4.2 Convergence of APROVE to Coherent Clusterhead states

The second type of convergence applicable to APROVE, is convergence to coherent clusterhead states. In APROVE, each node makes a local clustering decision based on only the messages received from its one-hop neighbours. Since nodes do not have the full-picture, steps must be taken to ensure coherency of local decisions. For example, if node \( i \) chooses node \( j \) to be its clusterhead at time \( k \), node \( j \) must become a clusterhead at time \( k \).

In APROVE, coherency is ensured by introducing the \( \text{CH}_{\text{cnvg},i} \) message, where a \( \text{CH}_{\text{cnvg},i} > 0 \) indicates to other nodes that node \( i \) will become a clusterhead on \( i \)'s next clustering decision (if it is not a clusterhead already). In the synchronous APROVE formulations, which use the \( CI \) clustering interval, this flag indicates that on the next \( CI \) interval, node \( i \) will become a clusterhead. If clustering contention is used, the \( \text{CH}_{\text{cnvg}} \) flag is used to resolve the contention, such that the node with the higher \( \text{CH}_{\text{cnvg}} \) flag wins. In asynchronous APROVE, \( \text{CH}_{\text{cnvg}} \) is an indicator for clusterhead election, in addition to clusterhead contention. If node \( i \)'s \( \text{CH}_{\text{cnvg},i} > 0 \), node \( i \) will immediately become a clusterhead.

At every iteration, node \( i \) updates its convergence message as follows: \( \text{CH}_{\text{cnvg},i} = r(i,i) + a(i,i) \). A justification for why \( r(i,i) + a(i,i) > 0 \) indicates convergence to clusterhead status is now presented. In [45], a simpler derivation of affinity propagation is presented, which shows that the availability and responsibility messages correspond to log-likelihood ratios. This derivation shows that the messages can be expressed as the following difference:

\[
r(i,j) = \rho_{ij}(1) - \rho_{ij}(0) \tag{4.4}
\]

where \( \rho_{ij}(1) \) is the log-likelihood that \( j \) is the clusterhead of \( i \), and \( \rho_{ij}(0) \) is the log-likelihood that \( j \) is not the clusterhead of \( i \). It follows that \( r(i,j) \) is the log-likelihood ratio that \( j \) is the clusterhead of \( i \). This concept can be observed from the responsibility formula: \( r(i,j) \leftarrow s(i,j) - \max_{j' \neq j} \{ a(i,j') + s(i,j') \} \). The first term, \( s(i,j) \), corresponds
to the log-likelihood that \( j \) is the clusterhead of \( i \) as described in Section 4.1, and the second term is proportional to the log-likelihood that \( j \) is not \( i \)'s clusterhead.

In affinity propagation, since messages correspond to log-likelihood ratios (as opposed to log-likelihoods), the beliefs correspond to posterior log-odds (as opposed to posterior log-probabilities). As described in Section 2.4.2, the belief at node \( i \) in the Sum-Product Algorithm can be found by taking the product of all incoming message at node \( i \), or taking the product of two messages being passed in opposite directions on an edge incident to node \( i \). The second approach is true since an outgoing message from node \( i \) on edge \( e \), is the product of all \( i \)'s incoming messages, except the incoming message on edge \( e \). Thus for affinity propagation, which operates in the log-domain, the posterior log-odds that node \( j \) is the clusterhead of node \( i \) is \( r(i, j) + a(i, j) \).

Following the above logic, the posterior log-odds that node \( i \) is its own clusterhead is \( a(i, i) + r(i, i) \). A node \( i \) can determine if it is a clusterhead by using a threshold of 0 as follows:

\[
\text{if, } a(i, i) + r(i, i) > 0 \text{ then, } i \text{ is a CH} \quad (4.5)
\]

Setting the threshold to 0 is equivalent to making the Maximum A Posteriori (MAP) estimation for the clusterheads:

\[
\log \left( \frac{P(c_i = i)}{P(c_i \neq i)} \right) > 0
\Rightarrow P(c_i = i) > P(c_i \neq i)
\Rightarrow P(c_i = i) > P(c_i = j, \forall j \neq i)
\]

where the above probabilities are posterior probabilities. It is apparent from the above discussion that a higher \( CH_{cnvg} \) value gives a higher posterior probability of \( P(c_i = i) \), which implies there is greater evidence supporting node \( i \) as a clusterhead. The posterior probability of node \( i \) is dependent on the number of positive likelihoods being propagated from the neighbouring nodes. The more nodes that would like to choose node \( i \) as their clusterhead, the higher the posterior probability for node \( i \). As a result, \( CH_{cnvg} \) naturally
Chapter 4. Analysis of APROVE Algorithm

63

Chapter 4. Analysis of APROVE Algorithm

4.5 Robustness to Channel Error

This section discusses APROVE’s robustness when channel error is present. Overall, APROVE’s message passing is robust to channel error, however performance degrades as error is increased. The first APROVE formulation, which uses a clustering interval, will encounter temporary incoherent cluster states as well as a large number of clusters when channel error is large. The second and third formulations, which use clusterhead contention, solve this increase in clusterheads.

A VANET clustering algorithm must be able to withstand channel error due to the harsh channel characteristics of the VANET environment. The APROVE algorithm’s robust nature makes it suitable for error prone communication. APROVE’s messages are updated often and contain memory from the previous iterations. As a result, the message update process is gradual and the loss of one iteration will not have an adverse effect on clustering performance. When a message is lost, the algorithm is able to use the last received iteration, which dampens the negative impact of error on clustering performance. Of course as the channel error increases and messages become more outdated, the algorithm’s performance will begin to degrade. To further improve performance in severe channel error, a more reliable MAC (such as [23]) can be used to increase the message reception probability.

4.5.1 Robustness of APROVE with Clustering Interval

For high channel error, APROVE with clustering interval suffers from a large number of clusters and temporary incoherent cluster states. As channel error is increased, the

takes into consideration the number of nodes a clusterhead has. The use of $\text{CH}_{cnvg}$ in clusterhead contention is justified, since the winning clusterhead will have a higher posterior probability of clusterhead status, and will also have more cluster members.
algorithm may be unable to converge within the given clustering interval. If the algorithm is unable to converge (the posterior log-odds remain negative for all nodes) then all of the nodes will have to become their own clusterheads. These temporary clusterheads will remain until the next clustering interval arrives.

In moderate channel error, where convergence is still achieved, the number of clusters can still be increased. If the expiry timer is set too low, then there will be a reasonable probability of a node’s clusterhead expiring due to channel error. This issue can be solved by increasing the expiry time when error is increased.

In addition to the increase in clusters, high channel error will also cause temporary incoherent cluster states to occur. For example, if node $j$ transitions from clusterhead state to cluster member state near the clustering interval, but node $i$’s last received message from $j$ indicates $\text{CH}_{\text{cvn},i,j} > 0$, then node $i$ will elect node $j$ as clusterhead. Temporarily, node $i$ will have an incorrect clusterhead choice. This issue will be remedied upon the first successful transmission from $j$ to $i$, which will indicate that node $j$ is not a clusterhead, and cause node $i$ to select a new clusterhead.

### 4.5.2 Robustness of APROVE with Clusterhead Contention

When channel error is present, clusterhead contention is a valuable protocol, since it reduces the number of clusters that are formed. If convergence is not achieved, and nodes are forced to become their own temporary clusterheads, clusterhead contention will cluster them automatically. All temporary clusterheads within range of one another contend for clusterhead status, and the winning clusterhead will have the current highest posterior log-odds (even if those odds are not positive).

### 4.5.3 Robustness of Asynchronous APROVE

Asynchronous APROVE implements clusterhead contention, thus it will also have a reasonable number of clusters in high channel error. In addition, asynchronous APROVE
reduces the prevalence of incoherent clusterhead states. In the synchronous APROVE formulations, nodes are required to re-elect clusterheads every $CI$, regardless of necessity, which increases the likelihood of incoherent choices being made. In asynchronous APROVE, nodes only elect a new clusterhead, when their current clusterhead is lost.

4.6 Synchronization Analysis

In this section, the synchronization requirements of APROVE’s different formulations are discussed. The first two formulations of APROVE, which use Clustering Interval and Cluster Contention, require some synchronization, whereas the final formulation is completely asynchronous.

4.6.1 Synchronous APROVE

The first two formulations of APROVE, which use clustering interval and clusterhead contention, demand synchronization amongst the clustering intervals. Vehicles must be aware of which iteration of the algorithm they are in, so that clustering intervals fall within the same iteration. On the other hand, the message passing procedure can be performed asynchronously. If the broadcast period for all nodes is $T_H = 1\text{s}$, and asynchronous message passing is used, a received message will be at most $1\text{s}$ old. The changes in responsibility and availability messages from one iteration to another are gradual enough that performance is unaffected.

Although message passing can operate asynchronously, the clustering intervals must occur within the same iteration. This concept of synchronization is illustrated in Figure 4.1. In Figure 4.1, nodes 1 and 3 are synchronized by clustering interval, and node 2 is operating asynchronously. In this example, $CI = 10\text{s}$ for all nodes, and node 1 is becoming the clusterhead of the group, as indicated by the $\text{CH}_{\text{cnvg}}$ becoming positive at iteration 6. This positive convergence flag indicates to other nodes that node 1 will
Figure 4.1: A timing diagram for clustering with synchronous APROVE. Nodes 1 and 3 are synchronized by their clustering intervals, and node 2 is operating asynchronously. The $CI = 10\text{s}$ for all nodes.

become a clusterhead when the next clustering interval arrives. Since node 2 is not synchronized with node 1, its clustering interval arrives early, and it naively chooses node 1 to be the clusterhead, even though node 1 is not yet a clusterhead. In the following iteration, node 2 sees that node 1 has not become a clusterhead as expected, and becomes its own clusterhead. In the synchronized case, node 3 elects node 1 to be its clusterhead in its 10th iteration, and node 1 becomes a clusterhead shortly after. The clusterhead status of node 1 is confirmed by node 3 in the next iteration when node 3 receives the HELLO message from node 1.

4.6.2 Asynchronous APROVE

The Asynchronous APROVE formulation removes the clustering interval requirement and can thus operate completely asynchronously. The asynchronous APROVE operation
Figure 4.2: A timing diagram for clustering with asynchronous APROVE. In this figure, nodes 1 and 2 are clustered with node 1 as the clusterhead. Several seconds later, node 3 comes within range, which is the clusterhead of a larger cluster.

is illustrated in Figure 4.2. In this figure, both the current clusterhead of each node $i$, $CH_i$, and the current convergence value, $CH_{cnvg,i}$ are given for each node at each iteration.

In this example, node 1 and node 2 are clustered with node 1 as the clusterhead. Arriving several seconds later is node 3, which is the clusterhead of a larger cluster, as indicated by its greater $CH_{cnvg}$ value. This causes cluster contention to occur between nodes 1 and 3, and node 3 wins. Node 2 then hears that its current clusterhead has joined node 3, and follows suit. The $CH_{cnvg}$ values are adjusted automatically by the affinity propagation algorithm.
Chapter 5

Simulation Results

In this chapter, simulation results are presented for the APROVE clustering algorithm. The NS2 simulator was used to validate APROVE against MOBIC in terms of clustering performance, average overhead of the clustering algorithm, and robustness to channel error. The metrics used to measure clustering performance include: average clusterhead duration, average cluster member duration, average rate of clusterhead change, and average number of clusters. The clustering performance is simulated for each of the three formulations of APROVE presented in Chapter 3, which include APROVE with clustering interval, APROVE with clusterhead contention, and asynchronous APROVE.

This chapter first presents the simulation setup, including both the network and traffic simulators. Following this, a description of the clustering performance metrics is presented. Next, the self-similarities and Future Prediction Period are tuned using simulations. With the parameter tuning complete, simulations are performed to observe the clustering performance as a function of average velocity for both APROVE and MOBIC. The average overhead for APROVE with aggregated message passing is also plotted against MOBIC. Finally, the algorithm’s robustness is compared by plotting the clustering performance metrics as a function of channel error for both APROVE and MOBIC. The channel error performance is compared for all three APROVE formulations.
5.1 Simulation Set-up

In this section, the simulation set-up is described. Simulations were performed using the NS2 network simulator, and traffic scenarios were provided by the MOVE traffic simulator [46]. The traffic scenarios supply the node mobility patterns to NS2 with tcl trace files.

5.1.1 Network Simulator

The APROVE algorithm was implemented in NS2 [47], which has been highly validated by the networking research community. The NS2 simulations used the 802.11 MAC and the 914MHz Lucent WaveLAN DSSS network card with a radio range of 250m. The MOBIC code was taken from a legacy version of NS2 provided by [25]. For all of the APROVE simulations, the $T_H$ parameter was set to 1s, and the self-similarities, $\tau_f$ and $CI$ parameters were varied. The simulations were performed on a highway scenario with 100 vehicles. Each simulation ran for 500s, however only the last 200s were used for performance metric calculations. This was to ensure that the duration metrics (cluster head and cluster member duration) had reached a steady state before measurements were made. The simulations were run on 8 unique traces and the performance results were averaged.

5.1.2 Traffic Simulator

Realistic traffic models for the VANET scenario were generated using the MOVE (MObility model generator for VEhicular networks) tool [46]. MOVE is built on top of the open source micro-traffic simulator, SUMO [48]. The MOVE tool outputs realistic NS2 traffic traces, which were then used in the NS2 simulations.

A rectangular looped 3-lane highway was chosen for the simulations’ traffic scenario. The rectangular loop is 3km long and 300m wide, and the three lanes travel around the
### Speed Distribution for Avg. Speed = 11.1 m/s

<table>
<thead>
<tr>
<th>Max Speed (m/s)</th>
<th>5.6</th>
<th>8.3</th>
<th>11.1</th>
<th>13.9</th>
<th>16.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Speed (Km/h)</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>Probability</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>

### Speed Distribution for Avg. Speed = 22.2 m/s

<table>
<thead>
<tr>
<th>Max Speed (m/s)</th>
<th>16.7</th>
<th>19.4</th>
<th>22.2</th>
<th>25</th>
<th>27.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Speed (Km/h)</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
</tr>
<tr>
<td>Probability</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>

### Speed Distribution for Avg. Speed = 33.3 m/s

<table>
<thead>
<tr>
<th>Max Speed (m/s)</th>
<th>27.8</th>
<th>30.6</th>
<th>33.3</th>
<th>36.1</th>
<th>38.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Speed (Km/h)</td>
<td>100</td>
<td>110</td>
<td>120</td>
<td>130</td>
<td>140</td>
</tr>
<tr>
<td>Probability</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>

### Speed Distribution for Avg. Speed = 44.4 m/s

<table>
<thead>
<tr>
<th>Max Speed (m/s)</th>
<th>38.9</th>
<th>41.7</th>
<th>44.4</th>
<th>47.2</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Speed (Km/h)</td>
<td>140</td>
<td>150</td>
<td>160</td>
<td>170</td>
<td>180</td>
</tr>
<tr>
<td>Probability</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 5.1: The speed distributions of the mobility scenarios. Traffic traces were generated for average speeds of 11.1, 22.2, 33.3, and 44.4 m/s. The speed distribution for each average speed is given.

The vehicles were given different maximum speeds to provide a realistic highway scenario. Random maximum speeds were assigned to the different vehicles by providing...
SUMO with a probability distribution input. Eight unique traces were generated for each of the average maximum speed groups of 11.1, 22.2, 33.3, and 44.4 m/s (40, 80, 120, and 160 km/h). For each speed group, speed distributions were assigned to enable 40% of the vehicles to travel at the average speed, 20% of the vehicles to travel at ± 10 km/h and 10% of the vehicles to travel at ± 20 km/h. The speed distributions for the traffic scenarios are summarized in Table 5.1.

5.2 Clustering Algorithm Performance Metrics

The clustering performance is evaluated using metrics which measure clusterhead stability, cluster member stability and the number of clusters produced. To aid in parameter optimization, a Clustering Performance Grade is defined, which considers both cluster stability and number of clusters. These clustering performance metrics are described in more detail below.

Average Clusterhead Duration

The average clusterhead duration is the average length of time that a node remains a clusterhead, once it has been elected. Long clusterhead duration is important for MAC schemes where the clusterhead is the central controller and scheduler. Frequent changes to the clusterhead will degrade the performance of these cluster-based MAC schemes.

Average Cluster Member Duration

This metric is found by measuring the average length of time that a node remains a cluster member of a specific clusterhead. The average cluster member duration is a good metric for judging the overall stability of the initial clustering. If initial clustering is effective, then cluster members will remain with their clusterhead for a long time.
Average Rate of Clusterhead Change

The average rate of clusterhead change is the overall average number of clusterhead changes per second. The more clusters that are present, the greater the number of clusterhead changes; therefore this metric conveniently considers both clusterhead duration and the number of clusters formed.

Average Number of Clusters

This metric describes the average number of clusters that are present at any given time in the algorithm. To effectively decrease network contention, fewer clusters is desirable. Typically, clustering algorithms strive to have only one clusterhead within a given broadcast range.

Clustering Performance Grade (CPG)

As the number of clusters is increased, the clusterhead and cluster member durations are also increased. For example, if every node becomes its own clusterhead, the clusterhead and cluster member duration will be infinite. To help measure this trade-off, a clustering performance grade is defined, which considers both cluster stability and number of clusters formed. The Clustering Performance Grade (CPG) is defined as follows:

\[
CPG = \frac{CH_{DUR} + CM_{DUR}}{CC}
\]

(5.1)

where \(CH_{DUR}\) is the clusterhead duration, \(CM_{DUR}\) is the cluster member duration, and \(CC\) is the cluster count (number of clusters).

5.3 Parameter Optimization

In this section, the self-similarities and \(\tau_f\) parameters are optimized to the VANET highway scenario. Parameter tuning is only performed on the APROVE formulation
Figure 5.1: Tuning of self-similarities for a CI of 60s. The clustering performance grade is plotted against self-similarity, and sorted by velocity.

that uses a Clustering Interval. The self-similarity and $\tau_f$ parameters are dependent on the network topology, in terms of broadcast range and node mobility, thus the optimized parameters will be equivalent for all APROVE formulations. As a result, the parameters selected in this section will be used in all subsequent simulations.

5.3.1 Tuning Self-Similarities

To tune the self-similarities, all possible parameters were swept and plotted in a variety of scatter plots. The self-similarities were swept from -10000 to -900, the velocities were swept over: 11, 22, 33, and 44 m/s, and the CI was swept over 30, 60, 120, and 150 s. The clustering performance grade is used to measure clustering performance and determine the optimal self-similarity setting. Scatter plots of CPG vs. self-similarity, sorted by velocity, were generated for each CI setting. An example of this plot is shown in Figure 5.1 for CI = 60.
Figure 5.2: Tuning of self-similarities for an average velocity of 33\(m/s\) and a \(CI\) of 60s. The clustering performance grade is plotted against self-similarity, and sorted by \(\tau_f\).

Figure 5.3: Tuning of self-similarities for an average velocity of 33\(m/s\) and a \(CI\) of 120s. The clustering performance grade is plotted against self-similarity, and sorted by \(\tau_f\).
In addition, scatter plots for each velocity and CI combination, sorted by $\tau_f$, were generated. For example, Figure 5.2 and Figure 5.3 show the effects of self-similarity and $\tau_f$ for an average velocity of 33 m/s, and a CI equal to 60 and 120s respectively. Following the analysis of all the generated plots, it was determined that a self-similarity of approximately -2000 gave the optimal clustering performance grade. This parameter gave optimal performance regardless of CI setting or average node velocity. For all future simulations, the self-similarities of all nodes were set to -2000, which was a scenario dependent setting.

5.3.2 Tuning Future Prediction Period

Next the Future Prediction Period, $\tau_f$, is tuned. Using a self-similarity of -2000, the clustering performance grade was plotted against $\tau_f$ for each velocity of 11, 22, 33, and 44 m/s and sorted by CI. It is observed from Figure 5.4 — Figure 5.7 that as the

![Figure 5.4: Tuning of $\tau_f$ parameter with self-similarities equal to -2000, and an average velocity of 11m/s. The clustering performance grade is plotted against $\tau_f$, and sorted by CI.](image-url)
Figure 5.5: Tuning of $\tau_f$ parameter with self-similarities equal to -2000, and an average velocity of 22 m/s. The clustering performance grade is plotted against $\tau_f$, and sorted by CI.

Figure 5.6: Tuning of $\tau_f$ parameter with self-similarities equal to -2000, and an average velocity of 33 m/s. The clustering performance grade is plotted against $\tau_f$, and sorted by CI.
average velocity is increased, the optimal $\tau_f$ setting is decreased. However, in a real application of APROVE, the $\tau_f$ will need to remain constant, even when vehicles enter different speed zones. From Figure 5.4 – Figure 5.7, it is determined that the average optimal value over all velocities is $\tau_f = 30s$. It is apparent that this parameter setting gives a reasonable clustering performance grade for all velocities, therefore the remainder of the simulations will use $\tau_f = 30s$.

### 5.4 Mobility Performance

In this section the clustering performance of APROVE and MOBIC are evaluated by sweeping over node mobility. Both algorithms were run on mobility traces with the average maximum speeds of: 11, 22, 33 and 44 m/s, as given in Table 5.1. Mobility performance was measured for MOBIC and all three formulations of APROVE. Both cluster stability and the number of clusters formed are studied using the performance
Figure 5.8: The average clusterhead duration as a function of velocity for both APROVE with Clustering Interval and MOBIC. Average maximum velocity is swept over 11, 22, 33, and 44 m/s, and $CI$ is swept over 30, 60, 120, and 150s.

5.4.1 Mobility Performance of APROVE with Clustering Interval

The mobility performance for APROVE with clustering interval is presented. The clustering performance is found for $CI = 30, 60, 120, \text{ and } 150\text{s}$. The average clusterhead duration, average cluster member duration, and average rate of clusterhead change are displayed in Figure 5.8, Figure 5.9, and Figure 5.10 respectively. As the clustering interval is increased, clustering decisions are made less often, which results in the observed increase in average durations and average rate of clusterhead change. The average number of clusters is plotted in Figure 5.11. It is evident that an increase in $CI$ results in an increase in the number of clusters formed, which is not desirable. As the $CI$ is increased, there are greater opportunities for new clusterheads to form, but no method to eliminate
Figure 5.9: The average cluster member duration as a function of velocity for both APROVE with Clustering Interval and MOBIC. Average maximum velocity is swept over 11, 22, 33, and 44 m/s, and $CI$ is swept over 30, 60, 120, and 150s.

Figure 5.10: The average rate of clusterhead change as a function of velocity for both APROVE with Clustering Interval and MOBIC. Average maximum velocity is swept over 11, 22, 33, and 44 m/s, and $CI$ is swept over 30, 60, 120, and 150s.
Chapter 5. Simulation Results

Figure 5.11: The average number of clusters as a function of velocity for both APROVE with Clustering Interval and MOBIC. Average maximum velocity is swept over 11, 22, 33, and 44 m/s, and CI is swept over 30, 60, 120, and 150s.

unwanted clusterheads, thus leading to an increase in clusters. This trade-off between increasing stability and increasing number of clusters is discussed in Chapter 4.

It is clear from Figures 5.8 – 5.10, that APROVE’s cluster stability far outperforms MOBIC’s cluster stability. In Figure 5.11, it is observed that the average number of clusters produced by APROVE is similar to the number of clusters produced by MOBIC, when CI is 30 or 60s. As APROVE’s CI is increased, the average number of clusters produced is more than MOBIC. This increase in clusters with an increased CI can be addressed by the clusterhead contention procedure, which is studied in Section 5.4.2.

The APROVE algorithm obtains clusters with higher stability than MOBIC because it considers cluster member suitability during both cluster formation and maintenance. MOBIC does not consider cluster member suitability when performing cluster maintenance, specifically when new cluster members enter an existent cluster. Although the elected clusterhead will initially have the lowest relative mobility in its neighbourhood,
any new nodes entering the range of this clusterhead, are forced to join the cluster. If a new node happens to have a high relative mobility to the clusterhead, it will exit the cluster quickly leading to poor cluster stability. MOBIC’s lesser stability performance could also be caused by error in the mobility metric. The use of received power in the mobility metric can lead to inaccuracies in the relative mobility calculation due to channel fluctuations.

5.4.2 Mobility Performance of APROVE with Clusterhead Contention

In this section, the mobility performance results for APROVE with clusterhead contention are presented. The average clusterhead duration, average cluster member duration, and average rate of clusterhead change are displayed in Figure 5.12, Figure 5.13, and Figure 5.14 respectively. These figures show that when the cluster contention proce-

Figure 5.12: The average clusterhead duration as a function of velocity for both APROVE with clusterhead contention and MOBIC. Average maximum velocity is swept over 11, 22, 33, and 44 m/s, and $CI$ is swept over 30, 60, 120, and 150s.
Figure 5.13: The average cluster member duration as a function of velocity for both APROVE with clusterhead contention and MOBIC. Average maximum velocity is swept over 11, 22, 33, and 44 m/s, and $CI$ is swept over 30, 60, 120, and 150s.

Figure 5.14: The average rate of clusterhead change as a function of velocity for both APROVE with clusterhead contention and MOBIC. Average maximum velocity is swept over 11, 22, 33, and 44 m/s, and $CI$ is swept over 30, 60, 120, and 150s.
Figure 5.15: The average number of clusters as a function of velocity for both APROVE with clusterhead contention and MOBIC. Average maximum velocity is swept over 11, 22, 33, and 44 m/s, and CI is swept over 30, 60, 120, and 150s.

dure is used, the cluster stability of APROVE is still greater than MOBIC. The cluster stability of APROVE is less when clusterhead contention is used, which is due to the more frequent clusterhead changes that are caused by clusterhead contention. The average number of clusters in Figure 5.15, is less than MOBIC for all CI settings. As expected, clusterhead contention results in a substantial improvement in average number of clusters, regardless of the CI parameter.

5.4.3 Mobility Performance of Asynchronous APROVE

Next, the mobility performance of asynchronous APROVE is presented. The performance of asynchronous APROVE is compared against the previous APROVE formulations and MOBIC. The previous APROVE formulations, which include APROVE with clustering interval (referred to here as just APROVE), and APROVE with clusterhead contention were simulated with $CI = 120s$. 
Figure 5.16: A comparison of the average clusterhead duration as a function of velocity for: APROVE, APROVE with clusterhead contention, asynchronous APROVE and MOBIC. APROVE and APROVE with clusterhead contention have $CI = 120s$.

Figure 5.17: A comparison of the average cluster member duration as a function of velocity for: APROVE, APROVE with clusterhead contention, asynchronous APROVE and MOBIC. APROVE and APROVE with clusterhead contention have $CI = 120s$. 
Figure 5.18: A comparison of the average rate of clusterhead change as a function of velocity for: APROVE, APROVE with clusterhead contention, asynchronous APROVE and MOBIC. APROVE and APROVE with clusterhead contention have $CI = 120\text{s}$.

Figure 5.19: A comparison of the average number of clusters as a function of velocity for: APROVE, APROVE with clusterhead contention, asynchronous APROVE and MOBIC. APROVE and APROVE with clusterhead contention have $CI = 120\text{s}$. 
The average clusterhead duration, average cluster member duration, and average rate of clusterhead change are presented in Figure 5.16, Figure 5.17, and Figure 5.18 respectively. These plots show that the cluster stability of asynchronous APROVE is similar to the cluster stability of APROVE with clusterhead contention. Asynchronous APROVE shows a significant stability improvement over MOBIC, but to a lesser degree than APROVE without clusterhead contention. The average number of clusters for asynchronous APROVE is presented in Figure 5.19. Asynchronous APROVE and APROVE with clusterhead contention both produce a small average number of clusters. The average number of clusters obtained is superior to both MOBIC and regular APROVE. This behaviour is expected since asynchronous APROVE also uses the clusterhead contention procedure during cluster maintenance.

5.5 Overhead Performance

The overhead performance of APROVE and MOBIC were also studied and compared. In this work, overhead is computed by counting the total number of bytes from all HELLO clustering beacons for the entire network. The overhead is measured in KBytes/sec. Overhead is measured for APROVE using the Clustering Interval formulation and aggregated message passing. The overhead of the other two APROVE formulations: APROVE with clusterhead contention and asynchronous APROVE, will be slightly greater than what is described here, since the full CH\textsubscript{cng} value needs to be transmitted instead of a boolean flag.

The overhead performance of both APROVE and MOBIC is compared in Figure 5.20, and it is observed that APROVE has a slightly lower overhead than MOBIC. In this low to moderate density network, MOBIC’s excess event-based broadcasts cause a greater overhead than APROVE, which has only periodic broadcasts. Since APROVE’s overhead is more dependent on neighbour list size than MOBIC, APROVE will have a greater
Figure 5.20: A comparison of average overhead as a function of velocity for both APROVE with Clustering Interval and MOBIC. Average maximum velocity is swept over 11, 22, 33, and 44 m/s.

overhead in higher density networks. The overhead of APROVE and MOBIC is discussed in detail in Section 4.3.

APROVE has a very reasonable overhead, which was achieved by combining all important data (position, velocity, responsibility, and availability) into one HELLO message. In addition, APROVE requires only periodic broadcasts, and does not require any event-based messages, which helps to keep the overhead down.

5.6 Robustness to Channel Error

In this section, the clustering performance of APROVE and MOBIC are compared when channel error is present. The channel error is produced using a uniform error model, where received packets are randomly dropped with a given probability. Simulations were run with channel error varied over the following probabilities: 0, 0.1, 0.2, 0.4, and
Figure 5.21: The average clusterhead duration as a function of channel error for both APROVE with Clustering Interval and MOBIC. Probability of channel error is swept over 0, 0.1, 0.2, 0.4, and 0.6, and $CI$ is swept over 30, 60, 120, and 150s.

0.6. The uniform error model used in the simulations overestimates channel error when compared to the more realistic Nakagami model. The robustness to error was measured for MOBIC and all three formulations of APROVE.

### 5.6.1 Robustness of APROVE with Clustering Interval

Channel error is introduced, and the robustness of MOBIC and APROVE with clustering interval are compared. The average clusterhead duration, average cluster member duration, average rate of clusterhead change, and average number of clusters are shown in Figure 5.21, Figure 5.22, Figure 5.23, and Figure 5.24 respectively. It is observed from these plots that the cluster stability decreases and the number of clusters increases as the probability of channel error is increased. The odd behaviour of the clusterhead duration at high channel error is explained by the large number of clusters that are produced in high error. At high channel error, convergence is not achieved and almost every node
Chapter 5. Simulation Results

Figure 5.22: The average cluster member duration as a function of channel error for both APROVE with Clustering Interval and MOBIC. Probability of channel error is swept over 0, 0.1, 0.2, 0.4, and 0.6, and CI is swept over 30, 60, 120, and 150s.

Figure 5.23: The average rate of clusterhead change as a function of channel error for both APROVE with Clustering Interval and MOBIC. Probability of channel error is swept over 0, 0.1, 0.2, 0.4, and 0.6, and CI is swept over 30, 60, 120, and 150s.
becomes its own clusterhead, thus the clusterhead duration is long.

As expected, the performance of both algorithms deteriorates as channel error is increased, however MOBIC’s performance drops off at a faster rate. From Figure 5.21, Figure 5.22, and Figure 5.23, it is observed that APROVE maintains reasonable cluster stability when faced with increasing channel error probability. MOBIC’s duration, however, become negligible with very little error introduced. The number of clusters increases rapidly for both APROVE and MOBIC. In high channel error, the underlying affinity propagation algorithm may be unable to converge, since many of the message updates are lost. Any nodes that were unable to converge will become their own clusterheads until the next clustering interval arrives. This causes a large number of clusters to form, especially when the $CI$ is large. This behaviour is undesirable, and can be remedied with clusterhead contention as described in the next section.
5.6.2 Robustness of APROVE with Clusterhead Contention

The robustness of APROVE with clusterhead contention is now explored. The main issue surrounding APROVE’s channel error performance in the previous section, is the rapid increase in clusters that are produced in high channel error. By introducing clusterhead contention, the number of clusters is decreased, and the robustness of the algorithm is improved. Figures 5.25 – 5.28 present the clustering performance of APROVE with clusterhead contention.

It is observed from Figure 5.25, Figure 5.26, and Figure 5.27 that APROVE’s cluster stability falls off at a reasonable rate with increasing error as expected. The clusterhead contention procedure greatly improves the number of clusters produced in high channel error, as seen in Figure 5.28. In high channel error, when convergence is not achieved, self-clusterheads are created temporarily, but are then clustered automatically in the cluster maintenance phase. All temporary clusterheads within range of one another contend

![Figure 5.25: The average clusterhead duration as a function of channel error for both APROVE with clusterhead contention and MOBIC. Probability of channel error is swept over 0, 0.1, 0.2, 0.4, and 0.6, and CI is swept over 30, 60, 120, and 150s.](image)
Figure 5.26: The average cluster member duration as a function of channel error for both APROVE with clusterhead contention and MOBIC. Probability of channel error is swept over 0, 0.1, 0.2, 0.4, and 0.6, and $CI$ is swept over 30, 60, 120, and 150s.

Figure 5.27: The average rate of clusterhead change as a function of channel error for both APROVE with clusterhead contention and MOBIC. Probability of channel error is swept over 0, 0.1, 0.2, 0.4, and 0.6, and $CI$ is swept over 30, 60, 120, and 150s.
Figure 5.28: The average number of clusters as a function of channel error for both APROVE with clusterhead contention and MOBIC. Probability of channel error is swept over 0, 0.1, 0.2, 0.4, and 0.6, and CI is swept over 30, 60, 120, and 150s.

for clusterhead status, and the winning clusterhead will have the current highest CH\textsubscript{cnvg} value. As a result, the best current clusterheads are elected, even though convergence was not achieved (all of the CH\textsubscript{cnvg} messages are still negative).

### 5.6.3 Robustness of Asynchronous APROVE

The robustness of asynchronous APROVE is studied and compared to MOBIC by introducing channel error. In addition, asynchronous APROVE is compared to the previous APROVE formulations, which include APROVE with clustering interval (referred to here as just APROVE), and APROVE with clusterhead contention. The CI parameter for the previous APROVE formulations is set to CI = 120s.

The average clusterhead duration, average cluster member duration, average rate of clusterhead change and average number of clusters are presented in Figure 5.16, Figure 5.17, Figure 5.18, and Figure 5.19 respectively. It is apparent from these plots that
Figure 5.29: A comparison of the average clusterhead duration as a function of channel error for: APROVE, APROVE with clusterhead contention, asynchronous APROVE and MOBIC. APROVE and APROVE with clusterhead contention have $CI = 120s$.

Figure 5.30: A comparison of the average cluster member duration as a function of channel error for: APROVE, APROVE with clusterhead contention, asynchronous APROVE and MOBIC. APROVE and APROVE with clusterhead contention have $CI = 120s$. 
Figure 5.31: A comparison of the average rate of clusterhead change as a function of channel error for: APROVE, APROVE with clusterhead contention, asynchronous APROVE and MOBIC. APROVE and APROVE with clusterhead contention have $CI = 120$ s.

Figure 5.32: A comparison of the average number of clusters as a function of channel error for: APROVE, APROVE with clusterhead contention, asynchronous APROVE and MOBIC. APROVE and APROVE with clusterhead contention have $CI = 120$ s.
asynchronous APROVE has the best channel error performance. The cluster performance metrics are equal to or better than APROVE with clusterhead contention, and all performance metrics show great improvement over MOBIC. Asynchronous APROVE’s use of the clusterhead contention procedure, allows it to produce a very reasonable number of clusters, even when a high probability of channel error is present.
Chapter 6

Conclusions and Future Work

Every day motor vehicle accidents cause unnecessary loss of life. The economic and societal ramifications of motor vehicle accidents and traffic congestion are of global concern. Research and development in Intelligent Transportation Systems (ITS) has been making great headway into the future safety and ease of our roads. The induction of the DSRC standard, has laid the groundwork for vehicle-to-vehicle and vehicle-to-roadside communication. Vehicular networks will incite countless applications, among them: reduction of traffic congestion, collision prevention, and traffic management.

Vehicle Ad hoc Networks (VANETs) exhibit many challenges for networking and communication. VANETs are prone to a highly dynamic network topology, packet congestion, and the hidden terminal problem. All of these challenges can be alleviated by a clustered network. Many cluster-based MAC and routing schemes have been suggested for VANETs, although this research is lacking a stable VANET clustering algorithm. To achieve stable clusters in a highly mobile network, such as VANETs, mobility must be considered during both clusterhead election and cluster maintenance.

Motivated by the ample research in cluster-based MAC and routing schemes for VANETs, this work presented a novel and stable mobility-based clustering algorithm called APROVE. APROVE distributively elected clusterheads by using affinity propaga-
tion from a communications perspective. The algorithm found clusters such that both the proximity and relative mobility between each clusterhead and its cluster members was minimized. The clusters created were stable, in that they exhibited long average cluster member duration, long average clusterhead duration, and low average rate of clusterhead change.

Two different methods of message passing were proposed for APROVE. The first involved segregating the affinity propagation messages into separate broadcast packets, which resulted in a relatively high overhead. An improvement to this method was aggregated message passing, which used an equal broadcast period for all clustering related messages, and then sent them all in one packet. Simulations showed that the overhead of aggregated message passing was very reasonable, and comparable to MOBIC’s overhead.

Three different formulations for clusterhead selection and maintenance were proposed. The first method used a clustering interval parameter, which defined the period of clustering decisions. If a node’s clusterhead was lost in between clustering decisions, it joined another current clusterhead, or became its own clusterhead. Simulations showed that this formulation exhibited very high cluster stability, especially as the clustering interval parameter was increased. However, an increased clustering interval parameter also resulted in an increase in average number of clusters, which was not desirable. When simulated for robustness, with increased channel error, the average number of clusters increased rapidly, and reached unreasonable values for probability of error greater than 40%. However, the overall clustering performance was more robust than MOBIC, since MOBIC’s durations fell to almost zero with as little as 20% probability of error.

The second APROVE formulation improved the first formulation by using clusterhead contention during cluster maintenance. When more than one clusterhead came within range of one another, the winning clusterhead was the one with the highest CH$_{creg}$ value, which was determined by the underlying affinity propagation algorithm. Although cluster stability was reduced somewhat, the clusterhead contention protocol greatly reduced the
number of clusters formed with high clustering interval. In addition, the clusterhead contention formulation was far more robust to channel error, and even in high channel error, a very reasonable number of clusters were formed.

In order for the first and second APROVE formulations to operate effectively, the clustering intervals of all nodes needed to be synchronized, such that clustering decisions were made within the same iteration. Since this was a difficult requirement for ad hoc networks, a completely asynchronous formulation of APROVE was proposed. Clusterhead election in asynchronous APROVE was made based on a node’s current $CH_{cnvg}$ value. Clusterhead contention was also used to keep the number of clusters at a minimum. Simulations showed that asynchronous APROVE had reasonable cluster stability and a low average number of clusters. The robustness of asynchronous APROVE was also shown to be equal to if not better than the previous formulations.

The excellent clustering performance, low overhead, asynchronous operation, and robustness to error make asynchronous APROVE a very attractive clustering algorithm for VANETs.

### 6.1 Future Work

One area of future research involving the APROVE algorithm is in parameter optimization. Parameter selection was discussed, and parameters were tuned to the highway scenario, however, it is clear that the parameters are very dependent on the specific network’s topology and mobility. Further research can be done to optimize parameters given specific mobility characteristics.

The APROVE algorithm lays the natural groundwork for a cluster-based MAC scheme. The high cluster stability, reasonable overhead, and robustness to error, make APROVE an excellent fit for a cluster-based MAC scheme. Thus another future research direction is the implementation of a cluster-based MAC or routing scheme, which works in concor-
dance with the APROVE algorithm. A new cluster-based MAC scheme, or a modified existing MAC scheme, can be designed to work in unison with the APROVE algorithm, and provide the true throughput and delay characteristics of APROVE. Together, a cluster-based MAC scheme and APROVE, can solve many of the critical issues facing VANET networking today.
Bibliography


