Ad Hoc Networks 13 (2014) 535–548

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Clustering in Vehicular Ad Hoc Networks using Affinity Propagation

B. Hassanabadi [*], C. Shea, L. Zhang, S. Valaee

Electrical and Computer Engineering Department, University of Toronto, Toronto, ON M5S 2E4, Canada

ARTICLE INFO

Article history:
Received 8 July 2012
Received in revised form 29 September 2013
Accepted 8 October 2013
Available online 20 October 2013

Keywords:
Ad Hoc Networks
Vehicular networks
Clustering
Affinity Propagation

ABSTRACT

The need for an effective clustering algorithm for Vehicular 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 Vehicular 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 clusterhead duration, average cluster member duration, average rate of clusterhead 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.

1. Introduction

Vehicular Ad Hoc Networks (VANETs) are an emerging field of research that will help improve road safety, navigation, and congestion. VANETs will enhance driver safety and reduce traffic deaths and injuries by implementing collision avoidance and warning systems. In addition, VANETs will enable the dissemination of traffic and road condition. This will aid in navigation and relieve traffic congestion by providing a driver with live routes that avoid road hazards and bottleneck areas. The vast sensor network that VANETs will create, is inciting countless other applications, and making VANETs a hot topic in ad hoc networking today.

The VANET environment contains many challenges for communication, many of which can be addressed by a clustered network. As highlighted in [1], VANETs suffer from high mobility and high node-density, which lead to channel congestion and the hidden terminal problem. VANETs have a highly-mobile environment with a rapidly changing network topology. Clustering the vehicles into groups of similar mobility will reduce the relative mobility between communicating neighbor nodes, and simplify routing. VANETs demand a high frequency of broadcast messages to keep the surrounding vehicles updated on position and safety information. These broadcasts lead to the "broadcast storm problem" [2], which describes the resulting congestion in the network. Both [2,3] recommend a clustered topology to effectively alleviate this congestion. In addition, both delay-sensitive (e.g. safety messages) and delay-tolerant (e.g. road/weather information) data will need to be transmitted, necessitating Quality-of-Service (QoS) requirements. Clustering the network will aid in supporting these QoS requirements as shown in [4].

There has been much research on cluster-based VANETs in the recent literature, most of which has been focused on developing cluster-based MAC protocols, as in [5–11] and cluster-based routing protocols, as in [12,13]. In [6,11], the clusterhead (CH) takes on a managerial role and

[*] Corresponding author. Tel.: +1 647 710 7598.
E-mail addresses: behnam@comm.utoronto.ca (B. Hassanabadi), c.shea@utoronto.ca (C. Shea), lzhang@comm.utoronto.ca (L. Zhang), valaee@comm.utoronto.ca (S. Valaee).

1570-8705/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.adhoc.2013.10.005

facilitates intra-cluster communication by providing a TDMA schedule to its cluster members. In [11], adjacent clusters are assigned different CDMA codes to avoid interference between clusters. The work in [11] shows a substantial reduction in probability of message delivery failure, when compared to traditional 802.11 MAC.

By introducing clustering we create a hierarchy in the network. The communication can be divided into cluster member to clusterhead and clusterhead to clusterhead communications. By allocating different channels to different clusters, the effect of interference and hidden node terminals can be reduced. Using clustering, a local infrastructure-based network is formed in which a clusterhead acts as an access point for cluster members. The clusterhead can then coordinate the transmissions in order to reduce packet collisions and to maximize the throughput.

The recent research in cluster-based MAC and routing protocols for VANETs motivates the need for an effective VANET clustering scheme. The clustering algorithms suggested in this research have low complexity and take advantage of node mobility more effectively. For highly-mobile networks, mobility must be considered during the clustering process in order to ensure cluster stability. Since the clusters provide the foundation for cluster-based MAC and routing schemes, cluster stability is vital for achieving reliable communication.

In addition to stability, an effective clustering algorithm must be robust to the harsh channel conditions present in the VANET environment. As discussed in [1], VANETs 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 evaluate the robustness of the algorithm when channel error is present.

In this paper, we propose a distributed mobility-based clustering algorithm for VANETs called APROVE. The proposed algorithm possesses excellent cluster stability, where stability is defined by long clusterhead duration, long cluster member duration, and low rate of clusterhead change. In addition, our algorithm is robust to channel error and exhibits a reasonable overhead. We achieve this algorithm by utilizing Affinity Propagation (AP) [14]. Our clustering scheme uses vehicles' position (provided by GPS) and velocity information to form clusters with low relative mobility between the clusterheads and their cluster members.

An earlier version of APROVE was first introduced in [15]. In [15] we presented a preliminary version of the algorithm with basic simulation results where we compared the clustering performance with MOBIC [16], a well stablished clustering algorithm in mobile ad hoc networks. In this work, we propose a revised version of APROVE to improve and extend the our work as follows:

- Asynchronous APROVE is proposed. There is no synchronization overhead which is an obvious advantage particularly when there is congestion.
- An aggregated message passing algorithm is proposed in which responsibility and availability messages are aggregated into a single HELLO message.
- A cluster head contention subroutine is introduced to reduce the number of clusters being produced.
- Analysis of overhead, convergence, and channel error is presented. Furthermore, new simulation results are added to observe the effect of channel error and to characterize the overhead.
- In addition to MOBIC, performance comparisons are made with two recent clustering algorithms for vehicular networks: Aggregate Local Mobility (ALM) clustering [17] and Position-based Prioritized Clustering (PPC) [18].

The rest of this paper is organized as follows. Section 2 discusses the related work in VANET and MANET clustering. Section 3 presents the Affinity Propagation algorithm. Section 4 introduces the APROVE clustering algorithm, and the algorithm's operation is analysed and discussed in Section 5. Section 6 presents the simulation results, and finally Section 7 concludes the paper.

2. Related work

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 CH that it hears a HELLO beacon from (or join all CHs if Gateway nodes are allowed). If the node does not hear from a CH within a given time period, it will become a CH 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 [11,13] without regard for mobility. In [6], 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, [10] addresses mobility by first classifying nodes into speed groups, such that nodes will only join a clusterhead 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 CH 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 [6] as a reactive measure, in that it is only considered after two clusters collide. The use of cluster speed groups in [10] may improve stability, but the large variations in the predefined speed groups (e.g. 60–110 km/h) will still allow high relative mobility inside the clusters.

In order to achieve the necessary cluster stability, mobility should play an integral role in initial cluster formation. A well-known and effective mobility-based clustering technique for ad hoc networks is MOBIC [16], which is an extension of the Lowest-ID algorithm [19]. In Lowest-ID, each node is assigned a unique ID, and the node with the lowest ID in its two-hop neighborhood is elected to be the clusterhead. In MOBIC, an aggregate local mobility metric is the basis for cluster formation instead of node ID. This metric considers a node's relative mobility to each of its neighbors, and the node with the lowest local mobility is elected as the clusterhead. The relative mobility from one node to its neighbor is estimated by comparing the received power of two consecutive messages from the neighboring node. Clusterhead re-election only occurs when two clusterheads move within range of one another for a certain contention interval. When a cluster member moves out of range of its clusterhead, it joins any current clusterhead in its neighborhood, or forms a new cluster.

A refinement of MOBIC called Aggregate Local Mobility (ALM) clustering was proposed in ALM [17]. In ALM, nodes are assumed to be GPS-enabled and use the piggybacked position information of their neighbors to calculate their relative mobility metrics. When a cluster member moves out of range of all existing clusterheads, it will go to and remain in the undecided state for a period of time. It will become its own clusterhead only if no other clusterheads come into range while it is undecided.

Another approach is called Position-based Prioritized Clustering (PPC) [18], in which each node is assumed to use an on-board navigation system. Using the destination location, desired driving speed and traffic information, the node estimates the time a node will travel on a given road and the average velocity for that duration. The PPC algorithm uses the latter two factors in calculating a node's priority in becoming a clusterhead. It favors nodes with high travel times, but assesses a heavy penalty to nodes whose actual velocity deviates from its average velocity.

3. Affinity Propagation

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. A revolutionary new technique for data clustering is the Affinity Propagation (AP) algorithm [14], which has been shown to produce clusters in much less time, and with much less error than traditional techniques (such as K-means clustering [20]). Here, clustering error refers to the application-specific distance between each data point and its assigned exemplar. In Affinity Propagation, data points pass messages to one another, which describe the current affinity that one data point has for choosing another data point as its exemplar.

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, therefore an application requiring a minimization (e.g. Euclidean distance) should have a negative similarity function. Each node i also has a self-similarity, $s(i,i)$, which influences the number of exemplars that are identified. Individual data points that are initialized with a larger self-similarity are more likely to become exemplars. If all the data points are initialized with the same constant self-similarity, then all data points are equally likely to become exemplars. By increasing and decreasing this common self-similarity input, the number of clusters produced is increased and decreased respectively.

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 j's desire to be an exemplar for i based on supporting feedback from other data points. The *self-responsibility*, $r(i,i)$ and *self-availability*, $a(i,i)$, both reflect accumulated evidence that i is an exemplar.

The update formulas for responsibility and availability are stated below:

$$r(i,j) \leftarrow s(i,j) - \max_{j' \text{s.t.} j' \neq j} \{a(i,j') + s(i,j')\} \tag{1}$$

$$a(i,j) \leftarrow \min_{i \neq j} \left\{ 0, r(j,j) + \sum_{\forall i' \notin \{i,j\}} \max\{0, r(i',j)\} \right\} \tag{2}$$

$$a(j,j) \leftarrow \sum_{i' \text{s.t.} i' \neq j} \max\{0, r(i',j)\} \tag{3}$$

In (1), the responsibility of the candidate exemplar j for node i is updated by the similarity of the two nodes minus the maximum of the addition of the similarity and availability of other potential exemplars for node i. The maximum term quantifies how well the best candidate exemplar is suited to be the cluster head for node i. In (2) the availability of candidate exemplar j for node i is related to the self responsibility of node j and the responsibility of the candidate exemplar j for other nodes. Using the minimum function the availability is forced to be a positive number. The more node j is collectively responsible for other neighbors and itself, the more it will be available for node i as an exemplar. Similar to (2), in (3), the self-availability of node j is related to its responsibility for other nodes.

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: $m_k = \beta m_k + \alpha m_{k-1}$, where $\alpha + \beta = 1$ and α and β are damping factors between 0 and 1 and m_k is the responsibility or availability message. In Affinity Propagation, the exemplar of each node i is found as follows:

$$exemplar_i = \arg\max_j \{a(i,j) + r(i,j)\} \tag{4}$$

In (4), the exemplar for node i is defined to be the node with the maximum collective availability and responsibility for node i.

The algorithm may be terminated once exemplar decisions become constant for some number of iterations, implying that the algorithm has converged. Another useful feature of the algorithm is the ability to determine when a specific data point has converged to exemplar status in its given cluster. When a data point's self-responsibility plus self-availability becomes positive, that data point has become the exemplar.

4. Proposed VANET clustering scheme

The proposed clustering technique uses the fundamental idea of Affinity Propagation from a communications perspective and in a distributed manner. We call this algorithm, Affinity PROpagation for VEhiclar networks (APROVE). In our algorithm, each node in the network transmits the responsibility and availability messages to its neighbors, and then makes a decision on clustering independently. This results, in a distributed algorithm, where every node is only clustering with those in its one-hop neighborhood.

4.1. Similarity function

We design a similarity function for our algorithm that is tailored to the VANET environment and produces stable clusters. Our similarity function, shown below in (5), is a combination of the negative Euclidean distance between node positions now and the negative Euclidean distance between node positions in the future. This is a simple way to consider both node position and node mobility in cluster creation.

$$s(i,j) = -\left(\|\mathbf{x_i} - \mathbf{x_j}\| + \|\mathbf{x'_i} - \mathbf{x'_j}\|\right) \tag{5}$$

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

where $\mathbf{x_i}$ is a vector of node i's current position, and $\mathbf{x}'_i$ is a vector of node i's predicted future position. The function predicts each node i's future position in τ_f seconds from now, based on node i's current velocity $v_{x,i}$ in the x direction and velocity $v_{y,i}$ in the y direction. The *Future Prediction Period*, τ_f, can be tuned for different types of mobility.

The similarity function, $s(i,j)$ represents the log-likelihood that node j is the clusterhead of node i, and the self-similarity (input preference), $s(i,i)$, represents the prior probability that node i is a clusterhead. The self-similarities are tuned to the mobility scenario during simulations, such that a minimum number of clusters is produced for a given vehicle broadcast range. We assign the same self-similarity to all vehicles, which gives them an equal likelihood of becoming the clusterhead. However, it is possible to assign certain vehicles (such as large trucks) a higher preference, which will make them more likely to become the clusterhead.

4.2. Message updating and the neighbor list

The APROVE messages are broadcast periodically with a period of T_H, where time is denoted by $k = T_H \cdot \{0, 1, 2, \ldots\}$ seconds. The responsibility and availability messages associated with the current time interval are defined as $r_{i,j}(k)$ and $a_{j,i}(k)$ respectively. The messages associated with the previous time interval are defined as $r_{i,j}(k-1)$ and $a_{j,i}(k-1)$. Each node, i, will have its current transmitted messages at time k, however it will not always have its current received messages (due to delay, message error, or collision). Node i stores each of the neighbor's last received messages, along with other pertinent information, in a neighbor list. For notation purposes, a message transmitted from i to j at time k is denoted, $r_{i,j}(k)$, whereas the last received message at i from j is denoted $r(i,j)$.

4.2.1. The neighbor list

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

$(x,y)_j$:	position vector of node j
$(v_x, v_y)_j$:	velocity vector of node j
$s(i,j)$:	similarity for i and j
$r(j,i)$:	last responsibility received from j
$r(i,j)$:	last responsibility transmitted to j
$a(i,j)$:	last availability received from j
$a(j,i)$:	last availability transmitted to j
$\mathrm{CH}_{cnvg,j}$:	log-odds of clusterhead convergence for j
CH_j:	clusterhead status flag for node j
t_{expire}:	time that node j expires

Note that the $r(i,j)$ and $a(j,i)$ entries are equivalent to $r_{i,j}(k-1)$ and $a_{j,i}(k-1)$, respectively. Node i should compute its transmitted responsibility and availability messages, and then store them in the neighbor list at the end of every time interval, k, for use in the succeeding time interval, $k+1$. Each node i should also maintain a self-entry, N_i^i.

4.2.2. APROVE message update rules

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

$$r_{i,j}(k) = \alpha\left[s_{i,j}(k) - \max_{j' \neq j}\{a_{i,j'}(k) + s_{i,j'}(k)\}\right] + \beta\, r_{i,j}(k-1) \tag{6}$$

$$a_{j,i}(k) = \alpha \min_{j \neq i}\left\{0, r_{i,i}(k) + \sum_{j' \neq j,i} \max\{0, r_{j',i}(k)\}\right\} + \beta\, a_{j,i}(k-1) \tag{7}$$

$$a_{j,j}(k) = \alpha \sum_{j' \neq i} \max\{0, r_{j,i}(k)\} + \beta\, a_{i,i}(k-1) \tag{8}$$

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

4.3. APROVE message passing

In this section we present an "aggregated" message passing procedure for the APROVE algorithm, where all

APROVE related messages are broadcast in the same HELLO beacon. This approach improves the overhead of the "segregated" message passing procedure presented in [15], where availability and responsibility messages were broadcast separately.

Each node i will periodically broadcast a HELLO beacon containing all of the necessary information for the APROVE algorithm. The hello beacon broadcast period, T_H, was set to 1 s in our simulations. The HELLO packet for node i contains its: ID, position, velocity, current clusterhead status, responsibility and availability for each of i's neighboring nodes, and current clusterhead convergence status. Node i will calculate its responsibility for each neighbor j, $r_{i,j}(k)$, using (6) and its availability for each neighbor j, $a_{j,i}(k)$, using (7). These values are damped with the previous transmitted responsibility and availability messages with a damping factor of 0.5. Node i then stores $r_{i,j}(k)$ and $a_{j,i}(k)$ in the responsibility array, $\mathbf{R_i}$, and availability array, $\mathbf{A_i}$, respectively. These values are stored at the next available array index, n. The node ID associated with the nth element of the responsibility and availability arrays is stored in the index array, $\mathbf{I_i}$, at the same index n. This index array maps the array indices to their associated node IDs. Next, the self-responsibility and self-availability are calculated with (6) and (8) respectively. The responsibility, availability, and index arrays are broadcast in the HELLO broadcast beacon.

The HELLO packet also includes the $CH_{cnvg,i}$ value, where $CH_{cnvg,i}(k) = r_{ii}(k) + a_{ii}(k)$. Due to the nature of the affinity propagation algorithm, a node's self-responsibility plus self-availability gives the log-odd probability that it will become a clusterhead. A positive $CH_{cnvg,i}$ value indicates that node i has become the best clusterhead amongst its neighbors, and has thus converged to clusterhead status. For every iteration of the algorithm, each node i updates its $CH_{cnvg,i}$ value accordingly and broadcasts it to its neighbors in the HELLO beacon. This value indicates to i's neighbor nodes whether or not they should consider i as a potential clusterhead. The complete HELLO broadcast procedure is outlined in Procedure 1.

Procedure 1. 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 neighbor $j$
  using (6), ($\alpha = \beta = 0.5$).
  2. Calculate availability, $a_{j,i}(k)$ for each neighbor $j$
  using (7), ($\alpha = \beta = 0.5$).
  3. Store responsibility for neighbor $j$, in
  responsibility array: $\mathbf{R_i}[n] = r_{i,j}(k)$
  4. Store availability for neighbor $j$, in availability
  array: $\mathbf{A_i}[n] = a_{j,i}(k)$
  5. Store the ID of $j$ in the index array: $\mathbf{I_i}[n] = j$
  6. Update CH convergence values:
     $CH_{cnvg}(k) = r_{i,i}(k) + a_{i,i}(k)$
  7. Update $CH_j$ for each neighbor $j$
  8. Broadcast HELLO beacon:
     $\langle j, (x, y)_j, (v_x, v_y)_j, CH_j(k), \mathbf{R_i}, \mathbf{A_i}, \mathbf{I_i}, CH_{cnvg}(k) \rangle$
```

Upon reception of a HELLO beacon from node j, node i will calculate its current similarity with j, $s_{i,j}(k)$, using (5) and update the position, velocity and similarity fields in its neighbor list entry for j. A node only considers neighbors 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, $\mathbf{I_j}$. If found, it will read its specific responsibility and availability messages from the $\mathbf{R_j}$ and $\mathbf{A_j}$ arrays. These messages are stored in the received message fields, $r(j,i)$ or $a(i,j)$ of j's neighbor list entry, $\mathbf{N_i^j}$. Node i will also update the CH_j and $CH_{cnvg,j}$ fields in j's neighbor list entry. This routine is summarized in Procedure 2.

Procedure 2. Reception of HELLO Beacons

```
Upon reception of a HELLO packet from node $j$, node $i$
  will:
  1. Check if $j$ is traveling 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 $\mathbf{I_j}$ index array. If found,
  $\mathbf{I_j}[n] = i$, read the appropriate responsibility and
  availability messages at $\mathbf{R_j}[n]$ and $\mathbf{A_j}[n]$.
  4. Receive and store $CH_{cnvg,j}$ value.
  5. Add/update $j$'s neighbor list entry, $N_i^j$:
     $\langle j, (x, y)_j, (v_x, v_y)_j, s(i,j), CH_j, a(i,j), r(j,i), t_{expire}, CH_{cnvg,j} \rangle$
```

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 neighbor 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 neighbor list entries will become outdated, leading to performance degradation. The effects of channel error are discussed in Section 5.4. 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.

4.4. Clusterhead selection and maintenance

In this section, we introduce two different procedures for the selection and maintenance of clusterheads. The first APROVE method uses a Clustering Interval time, *CI*, whereby all nodes make their clusterhead decisions every *CI* seconds. This method requires some synchronization amongst nodes, such that the cluster decisions are all made in the same time period. The second method does not require a clustering interval, which allows it to operate completely asynchronously. For the rest of this paper, the first method is denoted *APROVE* and the second method is denoted *Asynchronous APROVE*.

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 1.

4.4.1. APROVE

In this first method, clustering decisions are made periodically with a period of *CI* called the Clustering Interval. Note that the T_H message period must be small enough to allow the algorithm to converge within a *CI* period. Preliminary simulations show that a neighborhood of 40 nodes can always converge in under 10 iterations. Therefore a T_H of 1s, requires a minimum *CI* of 10 s.

Every *CI*, if node *i*'s $CH_{cnvg,i} > 0$, then node *i* becomes a clusterhead: $myCH_i = i$. Otherwise, node *i* finds its clusterhead as follows:

$$\forall\, N_i^j \in \mathbf{N_i} : CH_{cnvg,j} > 0,\quad myCH_i = \arg\max_j \{a(i,j) + r(i,j)\} \tag{9}$$

Node *i* chooses its neighbor with the maximum received availability plus transmitted responsibility, but it only considers its neighbors with $CH_{cnvg,j} > 0$. A positive $CH_{cnvg,j}$, indicates node *j* will become a clusterhead. Therefore, by selecting its clusterhead amongst neighbors with $CH_{cnvg,j} > 0$, node *i* will be ensured a valid clusterhead. Clusterhead selection is summarized in Procedure 3.

Procedure 3. Clusterhead Selection in APROVE

```
Every $CI$ seconds, node $i$ will check the following:
  1. if $CH_{cnvg,i} > 0$
       then $myCH_i = i$
  2. else for all $N_i^j \in \mathbf{N_i} : CH_{cnvg,j} > 0$ do
       $myCH_i = \arg\max_j\{a_{i,j}(k) + r_{i,j}(k)\}$
       if $CH_{cnvg,j} < 0, \forall N_i^j \in \mathbf{N_i}$
         then $myCH_i = i$
```

In between clustering iterations of *CI*, we perform cluster maintenance to ensure the ongoing validity of each node's current clusterhead. Every T_{CM} (the period of cluster maintenance), node *i* purges its neighbor list of old entries by checking the t_{expire} fields. Next, node *i* checks the status of its clusterhead. If $myCH_i = j$, and node *j* was purged or is not currently a clusterhead, $CH_j = 0$, then node *i*'s clusterhead has been lost. If the clusterhead has been lost, node *i* chooses one of its neighbors that is currently a clusterhead as follows:

$$\forall\, N_i^j \in \mathbf{N_i} : CH_j = 1,\quad myCH_i = \arg\max_j \{a(i,j) + r(i,j)\} \tag{10}$$

The $CH_{cnvg,j}$ flag is not used here, since it indicates the potential clusterheads for the next round, not the current clusterheads. If node *i* cannot find another neighbor that is currently a clusterhead, it becomes its own clusterhead.

To avoid excess clusterheads being created in between cluster decision intervals, a clusterhead contention subroutine is performed during cluster maintenance. Clusterhead contention occurs when two clusterheads come within range of one another for more than the *Cluster Contention Time*, *CCT*. The *CCT* allows for temporary clusterhead contention, which occurs with passing clusterheads. During clusterhead contention, $CH_{cnvg,i}$ is used to determine the winning clusterhead. For example, if clusterhead *j* comes within range of clusterhead *i* for more than the Cluster Contention Time, node *i* compares $CH_{cnvg,i}$ with $CH_{cnvg,j}$. If $CH_{cnvg,i} < CH_{cnvg,j}$, then *i* relinquishes its clusterhead status, otherwise it remains unchanged. If *i* has to relinquish its clusterhead status, then it will choose a new clusterhead using (10). The cluster members that used to belong to *i* will see that *i* has reset its CH status flag, $CH_i = 0$, which will cause them to make new clusterhead decisions as well. Cluster maintenance is summarized in Procedure 4.

Procedure 4. Cluster Maintenance in APROVE

```
Every $T_{CM}$ seconds, node $i$ will check the following:
  1. if current_time $> t_{expire,j}$ for any $N_i^j \in \mathbf{N_i}$
       then Purge $N_i^j$ from neighbor list
  2. if $myCH_i = j$ and ($CH_j = 0$ or $N_i^j$ expired)
       then $lost_{CH} = 1$
  3. if $CH_i = 1$ and $CH_j = 1$ for any $N_i^j \in \mathbf{N_i}$ for more
  than $CTO$ seconds
       and $CH_{cnvg,i} < CH_{cnvg,j}$
         then $lost_{CH} = 1$
  4. if $lost_{CH} = 1$ then
       for all $N_i^j \in \mathbf{N_i} : CH_j = 1$ do
         $myCH_i = \arg\max_j\{a_{i,j}(k) + r_{i,j}(k)\}$
       else if $CH_j = 0, \forall N_i^j \in \mathbf{N_i}$
         then $myCH_i = i$
```

Table 1
A summary of the different clusterhead fields.

Field	Name	Description
$CH_{cnvg,i}$	Clusterhead Convergence Value (Log-Odds)	Indicates the log-odds that node *i* has converged to clusterhead status. If $CH_{cnvg,i} > 0$, node *i* will become a clusterhead when the next clustering decisions are made
CH_i	Current Clusterhead Status Flag	Indicates if node *i* is currently a clusterhead. 1: true, 0: false
$myCH_i$	Current Clusterhead	Indicates the index of node *i*'s current clusterhead. If $myCH_i = j$, node *j* is the clusterhead of *i*

4.4.2. Asynchronous APROVE

The main limitation of the previous APROVE method, is the synchronization that is required amongst the clustering intervals, *CI*. We now present a second method called Asynchronous APROVE, which eliminates the clustering interval and thus eliminates the synchronization requirement. Instead of *CI* controlling when clusterhead decisions are made, node *i*'s $\text{CH}_{cnvg,i}$ value indicates when *i* should assume or relinquish clusterhead status. If node *i* is without a clusterhead, and its $\text{CH}_{cnvg,i} > 0$, then node *i* will become a clusterhead. If node *i* has no clusterhead, and its $\text{CH}_{cnvg,i} < 0$, then it will choose the best current clusterhead in its neighbor list as in (10). In the event that no current clusterheads are found, node *i* becomes its own clusterhead. The clusterhead contention subroutine is also used. When two clusterheads come within range of one another for more than *CCT*, the clusterhead with the lesser CH_{cnvg} value relinquishes its CH status. The final difference between this method and the previous method, involves the hand-over of clusterhead status. If a node *i* belongs to clusterhead *j*, but over time $\text{CH}_{cnvg,i}$ becomes greater than $\text{CH}_{cnvg,j}$, 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 clusterhead role. The clusterhead selection and maintenance for asynchronous APROVE is summarized in Procedure 5.

Procedure 5. Clusterhead Selection and Maintenance in Asynchronous APROVE

```
Every $T_{CM}$ seconds, node $i$ will check the following:
  1. if current_time $> t_{expire,j}$ for any $N_i^j \in \mathbf{N_i}$
       then Purge $N_i^j$ from neighbor list
  2. if $\text{myCH}_i = j$ and $(\text{CH}_j = 0$ or $N_i^j$ expired$)$
       then $\text{lost}_{CH} = 1$
  3. if $\text{CH}_i = 1$ and $\text{CH}_j = 1$ for any $N_i^j \in \mathbf{N_i}$ for more
     than $CTO$ seconds
       and $\text{CH}_{cnvg,i} < \text{CH}_{cnvg,j}$
         then $\text{lost}_{CH} = 1$
  4. if $\text{myCH}_i = j$ and $\text{CH}_{cnvg,i} > \text{CH}_{cnvg,j}$
       then $\text{lost}_{CH} = 1$
  5. if $\text{lost}_{CH} = 1$ then
       if $\text{CH}_{cnvg,i} > 0$
         then $\text{myCH}_i = i$
       else for all $N_i^j \in \mathbf{N_i} : \text{CH}_j = 1$ do
         $\text{myCH}_i = \arg\max_j\{a_{i,j}(k) + r_{i,j}(k)\}$
       else if $\text{CH}_j = 0, \forall N_i^j \in \mathbf{N_i}$
         then $\text{myCH}_i = i$
```

5. Analysis of APROVE

In this section we provide some insight into the operation of the APROVE algorithm. APROVE's parameter settings are discussed including: Clustering Interval, *CI*, and Future Prediction Period, τ_f. The overhead of APROVE is shown to be reasonable by comparing APROVE's overhead to the overhead of MOBIC. In addition, the convergence of APROVE is discussed in terms of both coherency of clusterhead decisions, and oscillation during message updates. Finally, the behavior of APROVE in the presence of channel error is discussed, and APROVE's robustness is argued.

5.1. Parameter selection

The first APROVE formulation uses a Clustering Interval, *CI*. The *CI* parameter determines how often nodes elect a new clusterhead based on the affinity propagation messages being passed in the background. When *CI* is increased, the clusterhead duration will also naturally increase, since clusterhead decisions are being made at a lesser rate. However, 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 and selecting new clusterheads during the cluster maintenance phase. In [15], we reported that an increasing number of clusters are being formed with an increasing *CI*. This was caused by nodes losing their clusterheads and forming new temporary clusterheads during the cluster maintenance phase. In this work, we have solved this issue by introducing the clusterhead contention subroutine during cluster maintenance, which insures that within a given broadcast range, only one clusterhead is ever present.

With the addition of the clusterhead contention procedure in APROVE with clustering interval, it is reasoned that increased cluster stability and cluster performance will be achieved with a longer *CI*. In fact, the asynchronous APROVE method is essentially APROVE with an infinite clustering interval, along with some small adjustments to simplify clusterhead hand-over.

APROVE's second parameter is the Future Prediction Period, τ_f, used in the similarity function (5). 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 τ_f is to assume that individual vehicles will remain at a relatively constant speed for τ_f seconds. Thus, this parameter should be tuned to the network's mobility pattern. The τ_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 τ_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 τ_f should be increased. The proper value for τ_f is scenario dependent and can be found by empirical studies, as illustrated in the simulations.

5.2. Overhead analysis

In this section, we present the overhead for APROVE with aggregated message passing and compare it to the overhead of MOBIC. Each APROVE HELLO beacon includes the IP and MAC headers, the position and velocity information, the clusterhead 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 and y positions, 4 bytes for each of the v_x and v_y velocities, 1 byte for the CH flag, 4 bytes for the CH_{cnvg} value, and 4 bytes for each member of the availability, responsibility, and index arrays. Since these arrays are exactly as long as the neighbor list, we need 12 bytes for each member of the neighbor list.

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 neighbors. The current status uses 1 byte, the mobility metric uses 8 bytes, and 5 bytes are used for each member of the neighbor list (4 bytes for the neighbor index, and 1 byte for the neighbor status).

The size of the APROVE and MOBIC beacons are summarized below in (11) and (12) respectively.

$$\begin{aligned} size_{APR} &= \mathrm{IP\_HDR\_LEN} + \mathrm{MAC\_HDR\_LEN} + \mathrm{POS} \\ &\quad + \mathrm{VEL} + \mathrm{CH\_INFO} + 12N_{size} \\ &= 20 + 58 + 8 + 8 + 5 + 12N_{size} \\ &= 99 + 12N_{size} \end{aligned} \tag{11}$$

$$\begin{aligned} size_{MOB} &= \mathrm{IP\_HDR\_LEN} + \mathrm{MAC\_HDR\_LEN} + \mathrm{CH\_STAT} \\ &\quad + \mathrm{MOBILITY} + 5N_{size} \\ &= 20 + 58 + 1 + 8 + 5N_{size} = 88 + 5N_{size} \end{aligned} \tag{12}$$

where N_{size} is neighbor list size.

The size of the HELLO beacons, for both APROVE and MOBIC, increases as a function of neighbor list size. Although it is apparent from (11) and (12) 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 1 s. In MOBIC, however, nodes make both periodic broadcasts every 1 s, and event-based broadcasts. The event-based broadcasts occur every time a node decides to change its status to either clusterhead 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 neighbor list size is reasonable, APROVE will have a lower overhead than MOBIC. In high-density networks, if the neighbor list size becomes large enough, APROVE will have a higher overhead than MOBIC. A plot comparing the overhead of the two algorithms is found in Section 6.

5.3. Convergence analysis

In this section, we present and discuss two definitions of convergence for APROVE. The first refers to the convergence of the underlying affinity propagation algorithm to non-oscillating states. 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.

5.3.1. Convergence of APROVE to non-oscillating states

Affinity propagation is derived on a loopy factor graph [21]. Several sufficient conditions for the convergence of loopy sum-product algorithm were presented in [22,23]. Unfortunately, affinity propagation does not satisfy these conditions, and convergence cannot be guaranteed. When affinity propagation fails to converge, oscillations occur in the beliefs, which can be interpreted as a too-large step-size in gradient-descent minimization [24]. These oscillations can be solved by damping responsibility and availability messages [14], as performed in (6)–(8).

Another cause of oscillation in affinity propagation is the presence of degenerate cases. Degeneracies lead to multiple minima, which prevent convergence of the algorithm. Frey and Dueck [14] suggest adding a small jitter to the similarities to prevent these degenerate scenarios. 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 oscillations caused by degeneracies.

5.3.2. Convergence of APROVE to coherent clusterhead states

The second type of convergence applicable to APROVE, is convergence to coherent clusterhead states. 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 $CH_{cnvg,i}$ message, where a $CH_{cnvg,i} > 0$ indicates to other nodes that node i will become a clusterhead on i's next clustering decision. In addition, $CH_{cnvg,i}$ is used to resolve clustering contention, such that the clusterhead with the higher CH_{cnvg} value wins.

At every iteration, node i updates its convergence message as follows: $CH_{cnvg,i} = r(i,i) + a(i,i)$. Thus to ensure coherent clusterhead decisions, we must justify that $r(i,i) + a(i,i) > 0$ implies convergence to clusterhead status. The derivations of Affinity Propagation presented in [21,25] show that $r(i,i) + a(i,i)$ gives the posterior log-odds that node i is a clusterhead.

Therefore, a node i can determine if it is a clusterhead by using a threshold of 0 as follows:

$$\text{if } a(i,i) + r(i,i) > 0 \text{ then, } i \text{ is a CH} \tag{13}$$

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

$$\begin{aligned} &\log\left(\frac{P(c_i = i)}{P(c_i \neq i)}\right) > 0 \\ &\Rightarrow P(c_i = i) > P(c_i \neq i) \\ &\Rightarrow P(c_i = i) > P(c_i = j, \forall j \neq i) \end{aligned}$$

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 neighboring 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 takes into consideration the number of nodes a clusterhead has. The use of 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.

5.4. Robustness to channel error

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 does not have any adverse effects. 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 [9]) can be used to increase the message reception probability.

In cases of severe channel error, convergence of the underlying affinity propagation algorithm may not be achieved, and nodes are forced to become their own temporary clusterheads. However, the clusterhead contention protocol will cluster these nodes using the current CH_{cnvg} values. In a given broadcast range, the node i with the highest $CH_{cnvg,i}$ value (highest posterior log-odds) will become the clusterhead, even if that node was unable to converge to clusterhead status in the high channel error ($CH_{cnvg,i} < 0$).

6. Simulations

The APROVE algorithm was implemented in NS2, which has been highly validated by the networking research community. The NS2 simulations used the 802.11 MAC and the 914 MHz Lucent WaveLAN DSSS network card with a radio range of 250 m. The MOBIC code was taken from a legacy version of NS2 provided by [16]. The implementation of the PPC and ALM algorithms were done in NS2 according to the specifications in [18,17]. For all of the APROVE simulations, $T_H = T_{CM} = 1$ s. The self-similarities and τ_f parameters were tuned to the VANET scenario with simulations. These simulations gave optimal parameter settings of: self-similarity = −2000 and $\tau_f = 30$ s, which were used in all remaining simulations. The simulations were performed on a highway scenario with 100 vehicles. Each simulation ran for 500 s, however only the last 200 s were used for performance metric calculations. This was to ensure that the duration metrics (clusterhead and cluster member duration) had reached a steady state before measurements were made. The simulations were run on 8 unique mobility traces and the performance results were averaged.

6.1. Traffic scenarios

Realistic traffic models for the VANET scenario were generated using the MOVE (MObility model generator for VEhicular networks) tool [26]. MOVE is built on top of the open source micro-traffic simulator, SUMO [27]. The MOVE tool outputs realistic NS2 traffic traces, which were then used in the NS2 simulations.

Our proposed algorithm will not cluster vehicles moving in opposite directions. PPC does not make explicit provisions for bi-directional traffic and is designed for one-direction traffic. MOBIC and ALM will cluster vehicles moving in opposite directions. However, platoons passing each other in opposite directions will cause their clusters to merge and then reform, which degrades cluster stability and incurs more overhead. Our traffic scenario was designed in order to provide a fair comparison of APROVE and the other algorithms without any modifications. A rectangular looped 3-lane highway was chosen for the simulations' traffic scenario. The rectangular loop is 3 km long and 300 m wide, and the three lanes travel around the loop in a single direction. The rectangular loop was designed to be wider than the 250 m broadcast range, so that clustering could not occur amongst vehicles moving in opposite directions for any of the algorithms.

The vehicles were given different maximum speeds to provide a realistic highway scenario. Random maximum speeds can be assigned to the vehicles by giving SUMO a probability distribution. 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 140 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 vehicles enter the scenario sequentially, one second apart. In our simulations, all vehicles have a length of 5 m, an acceleration rate of 0.8 m/s^2, and a deceleration rate of 4.5 m/s^2. A minimum gap of 2.5 m is maintained between vehicles in the same lane, and vehicles are allowed to change lanes to pass each other.

6.2. Clustering performance metrics

We evaluate the cluster stability and overall performance of both APROVE and MOBIC using the metrics listed below.

1. **Average Clusterhead Duration** The average length of time that a node remains a clusterhead, once it has been elected.
2. **Average Cluster Member Duration** The average length of time that a node remains a cluster member of a specific clusterhead.

3. **Average Rate of Clusterhead Change** The overall average number of clusterhead changes per second.
4. **Average Number of Clusters** The average number of clusters that are present at any given time in the algorithm.

6.3. Overhead performance

The overhead performance of APROVE and the other algorithms is compared in Fig. 1. Specifically, the amount of information transmitted in the HELLO beacons. We do not observe any significant gap in terms of the overhead between these algorithms. Time synchronization was assumed in the simulation. Since the synchronization mechanism is out of the scope of this work, it was not implemented in the simulation. Thus, the overhead depicted in Fig. 1 does not include the synchronization overhead. Therefore, synchronous and asynchronous APROVE are represented in a single curve since the two methods transmit the same packet periodically and only differ in their clustering decisions.

6.4. Mobility performance

The clustering performance of APROVE and Asynchronous APROVE were evaluated against MOBIC, PPC and ALM by sweeping over node mobility. All algorithms were run on mobility traces with the following average maximum speeds: 11.1, 22.2, 33.3 and 44.4 m/s. Each data point presented was run eight unique times and averaged. The performance results are displayed in Fig. 2. In our simulations results, CM and CH stand for Cluster Head and Cluster Member respectively.

From Fig. 2d we can see that all of the clustering algorithms are relatively robust to changes in the average

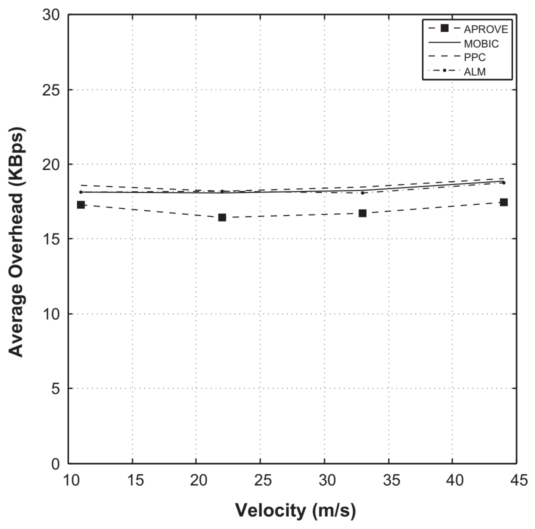

Fig. 1. A comparison of average overhead performance for APROVE, MOBIC, PPC and ALM as a function of velocity. The overhead counts the HELLO clustering beacons for the entire network and is measured in KBytes/s. The average maximum vehicle velocities spanned are: 11.1, 22.2, 33.3, and 44.4 m/s.

maximum speed in terms of those metrics. This is the desired outcome as all of the algorithms aim to account for the mobility of the network. The only significant difference is in the average rate of clusterhead change for the PPC algorithm, as shown in Fig. 2. Unlike the other simulated algorithms, PPC does not rely on an aggregate measure of network mobility but rather each vehicle's deviation from its predicted average speed. That the former metric typically changes at a slower rate than the latter could explain PPC's poor performance and stability in Fig. 2. In the same figure, we can see the gain of ALM over the original MOBIC, which can be attributed to its deferral of new clusterhead creation when a member loses its clusterhead.

In all the metrics shown in Fig. 2, the clustering performance of both APROVE formulations exceed that of the other algorithms, in forming fewer, more stable, and longer lasting clusters. In the case of APROVE, it is evident that an increasing *CI* leads to an increase in cluster stability. It is also noted that the Asynchronous APROVE formulation has comparable clustering performance to APROVE with a large clustering interval. In Fig. 2b and c, the performance of Asynchronous APROVE surpasses APROVE for all clustering interval settings.

The choice of synchronous or asynchronous APROVE depends on the metric of interest. As can be seen in Fig. 2, for some metrics asynchronous APROVE performs better (Average Cluster Member Duration and Average Rate of Cluster Head change) while for other metrics (Average Cluster Head Duration and Average Number of Clusters) the synchronized version (with optimized CI) is superior.

6.5. Performance with channel error

APROVE, Asynchronous APROVE, MOBIC, PPC and ALM were also simulated in poor channel conditions to evaluate the robustness of the algorithms. Simulations were performed with varying degrees of channel error present, and the performance was evaluated using the same four metrics described earlier.

Channel error was produced using a uniform error model with the following probabilities of error: 0, 0.1, 0.2, 0.4, and 0.6. We used the same mobility traces of Section 6.4 with the average maximum speed of 33.3 m/s. A uniform error model was used instead of the more realistic Nakagami model, which assumes that reception probability decreases as the distance of propagation increases. By using a uniform error model, the simulations overestimate channel error.

The simulation results for APROVE, Asynchronous APROVE, and the other algorithms with channel error present are shown in Fig. 3. Since APROVE is an iterative algorithm which require multiple rounds of message-passing to converge, one may expect that it is more sensitive to channel errors than the other algorithms which simply rely on updates to nodes' neighbor table. However, we can see from the figures in Fig. 3 that the APROVE algorithms outperforms the others despite the increasing channel error.

For neighbor table-based algorithms like MOBIC, successive channel errors leads to an erroneous deletion of a neighbor's entry from the table. When the deleted

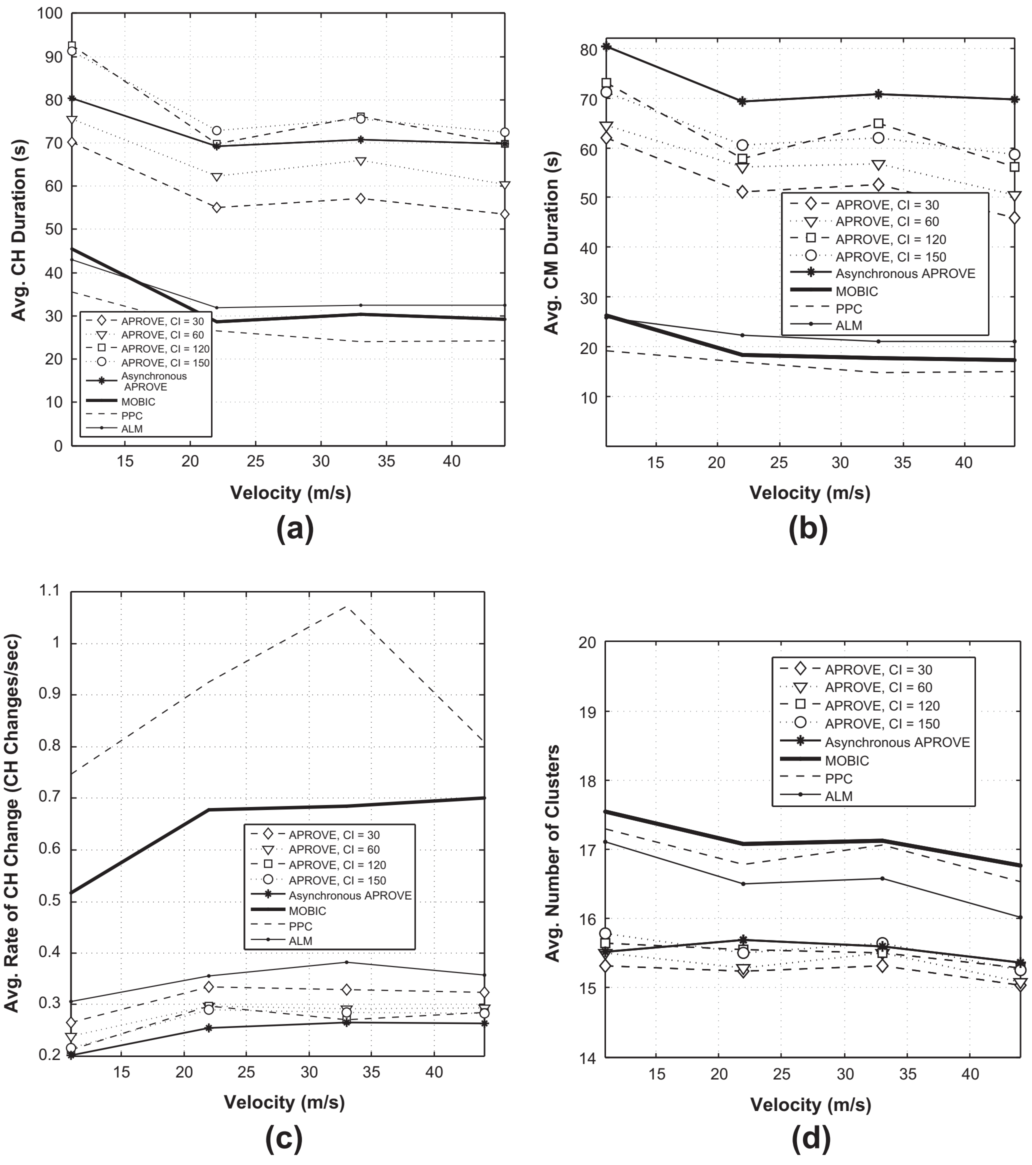

Fig. 2. The impact of velocity on clustering performance, comparing APROVE, Asynchronous APROVE, MOBIC, PPC, and ALM. APROVE with Clustering Interval is plotted with *CI* = 30, 60, 120, and 150 s. Both APROVE formulations have $\tau_f = 30$ s. (a) The average Cluster Head (CH) duration. (b) The average Cluster Member (CM) duration. (c) The average rate of Cluster Head (CH) change. (d) The average number of clusters.

neighbor is node's clusterhead, the node becomes its own clusterhead which increases the rate of change and the number of clusterheads and decreases the cluster member duration. ALM introduces an intermediate undecided state in this case, which can be seen in its improvement over MOBIC. Although APROVE also suffers from this, its clustering mechanism does not depend on the neighbor table directly. Instead, an APROVE node uses the availability and responsibility values calculated using network information transmitted from all of its neighbors. This provides a level of robustness to the APROVE algorithm at the cost of a larger broadcast packet.

From Fig. 3a and b we see that APROVE's clusterhead and cluster member durations are only significantly for channel error probabilities exceeding 0.2. When channel error is greater than 0.2, we see the expected performance deterioration in the clusterhead and cluster member durations for APROVE. However, even for an error probability of 0.6, the APROVE algorithms maintains a reasonable level of performance whereas the durations of MOBIC, PPC and ALM become negligible with very little error.

In Fig. 3c it is observed that the average rate of clusterhead change increases rapidly for MOBIC and PPC, at a lower rate for ALM, and stays reasonable for both APROVE formulations. In Fig. 3d, all algorithms produce an increasing number of clusters with increasing channel error. We note that ALM yields an improvement over MOBIC and PPC. However, the number of clusters generated by both APROVE formulations is much less than that of the others, even in high channel error conditions. As with the mobility

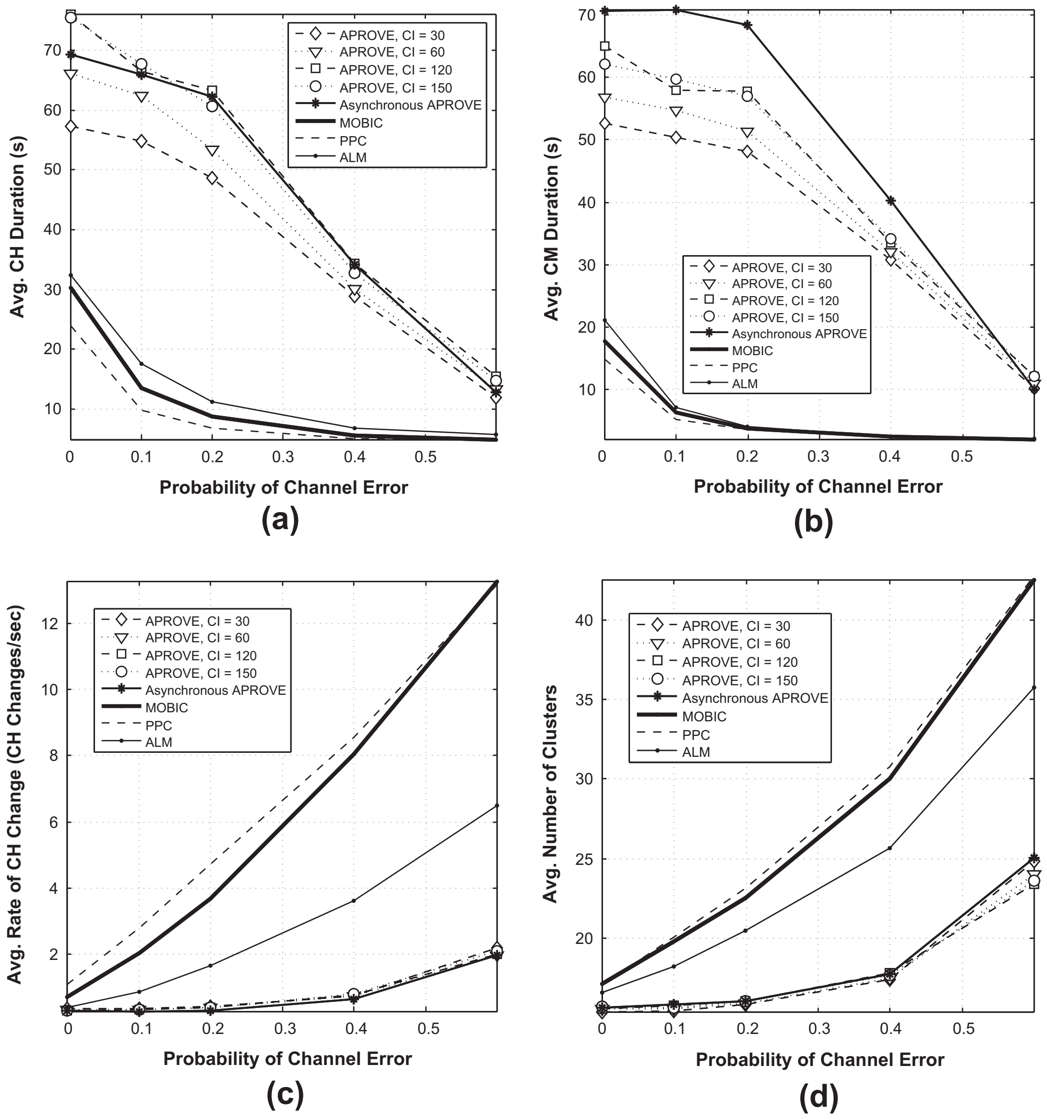

Fig. 3. The impact of channel error on clustering performance, comparing APROVE, Asynchronous APROVE, MOBIC, PPC and ALM. APROVE with Clustering Interval is plotted with CI = 30, 60, 120, and 150 s. Both APROVE formulations have $\tau_f = 30$ s. (a) The average clusterhead duration. (b) The average cluster member duration. (c) The average rate of clusterhead change. (d) The average number of clusters.

performance study, the performance of Asynchronous APROVE is very similar to the performance of APROVE with a long *CI*. Fig. 3b shows that in terms of average cluster member duration, Asynchronous APROVE has the best performance.

7. Conclusion

Motivated by the much needed research in cluster-based MACs and routing schemes for VANETs, we have proposed a novel and stable mobility-based clustering algorithm called APROVE. Our algorithm distributively elects clusterheads by using affinity propagation from a communications perspective. The algorithm finds clusters that minimize both the relative mobility and the distance from each clusterhead to its cluster members. The clusters created are stable and exhibit long average cluster member duration, long average clusterhead duration, and low average rate of clusterhead change. APROVE is robust to channel error, and it exhibits reasonable overhead.

Two different formulations of APROVE were proposed, denoted: APROVE and Asynchronous APROVE. APROVE used a clustering interval parameter, which required synchronization, whereas Asynchronous APROVE operated completely asynchronously. Both formulations used clusterhead contention during cluster maintenance, which reduced the number of clusters that were formed. Simulations showed that Asynchronous APROVE's clustering performance was comparable to, if not better than, APROVE's performance. Asynchronous APROVE's excellent clustering performance, robustness to error, and simple asynchronous operation, make it a viable algorithm for clustering in VANET's dynamic and harsh environment.

References

[1] M. Torrent-Moreno, M. Killat, H. Hartenstein, The challenges of robust inter-vehicle communications, in: 62nd Vehicular Technology Conference, 2005, VTC-2005-Fall, vol. 1, IEEE, 2005, pp. 319–323, doi:http://dx.doi.org/10.1109/VETECF.2005.1557524.

[2] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, J.-P. Sheu, The broadcast storm problem in a mobile ad hoc network, in: MobiCom '99: Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, ACM, New York, NY, USA, 1999, pp. 151–162. doi:http://doi.acm.org/10.1145/313451.313525.

[3] W. Chen, S. Cai, Ad hoc peer-to-peer network architecture for vehicle safety communications, IEEE Communications Magazine 43 (4) (2005) 100–107, http://dx.doi.org/10.1109/MCOM.2005.1421912.

[4] R. Ramanathan, M. Steenstrup, Hierarchically-organized, multihop mobile wireless networks for quality-of-service support, Mobile Networks and Applications 3 (1) (1998) 101–119.

[5] L. Bononi, M. Di Felice, A cross layered MAC and clustering scheme for efficient broadcast in VANETs, in: IEEE International Conference on Mobile Adhoc and Sensor Systems, 2007. MASS 2007, 2007, pp. 1–8, doi:http://dx.doi.org/10.1109/MOBHOC.2007.4428735.

[6] Y. Gunter, B. Wiegel, H. Grossmann, Cluster-based medium access scheme for VANETs, Intelligent Transportation Systems Conference, 2007, ITSC 2007, IEEE, 2007, pp. 343–348, doi:http://dx.doi.org/10.1109/ITSC.2007.4357651.

[7] Z. Rawashdeh, S. Mahmud, Media access technique for cluster-based vehicular ad hoc networks, in: 68th Vehicular Technology Conference, 2008, VTC 2008-Fall, IEEE, 2008, pp. 1–5, doi:http://dx.doi.org/10.1109/VETECF.2008.448.

[8] F. Farnoud, S. Valaee, Repetition-based broadcast in vehicular ad hoc networks in Rician channel with capture, in: INFOCOM 2009. in: The 28th Conference on Computer Communications, IEEE, 2008, pp. 1–6, doi:http://dx.doi.org/10.1109/INFOCOM.2008.4544661.

[9] B. Hassanabadi, L. Zhang, S. Valaee, Index coded repetition-based MAC in vehicular ad-hoc networks, in: 6th IEEE Consumer Communications and Networking Conference, 2009, CCNC 2009, 2009, pp. 1–6, doi:http://dx.doi.org/10.1109/CCNC.2009.4784947.

[10] O. Kayis, T. Acarman, Clustering formation for inter-vehicle communication, in: IEEE Intelligent Transportation Systems Conference, 2007, ITSC 2007, 2007, pp. 636–641, doi:http://dx.doi.org/10.1109/ITSC.2007.4357779.

[11] H. Su, X. Zhang, Clustering-based multichannel MAC protocols for QoS provisionings over vehicular ad hoc networks, IEEE Transactions on Vehicular Technology 56 (6) (2007) 3309–3323, http://dx.doi.org/10.1109/TVT.2007.907233.

[12] B. Wiegel, Y. Gunter, H. Grossmann, Cross-layer design for packet routing in vehicular ad hoc networks, in: IEEE 66th Vehicular Technology Conference, 2007, VTC-2007 Fall 2007, 2007, pp. 2169–2173, doi:http://dx.doi.org/10.1109/VETECF.2007.455.

[13] R.E.R.A. Santos, N. Seed, Inter vehicular data exchange between fast moving road traffic using ad-hoc cluster based location algorithm and 802.11b direct sequence spread spectrum radio, in: PostGraduate Networking Conference.

[14] B.J. Frey, D. Dueck, Clustering by passing messages between data points, Science 315 (2007) 972–976.

[15] C. Shea, B. Hassanabadi, S. Valaee, Mobility-based clustering in VANETs using affinity propagation, in: Globecom 2009, 2009.

[16] P. Basu, N. Khan, T. Little, A mobility based metric for clustering in mobile ad hoc networks, in: 2001 International Conference on Distributed Computing Systems Workshop, 2001, pp. 413–418, doi:http://dx.doi.org/10.1109/CDCS.2001.918738.

[17] E. Souza, I. Nikolaidis, P. Gburzynski, A new aggregate local mobility (alm) clustering algorithm for vanets, in: 2010 IEEE International Conference on Communications (ICC), 2010, pp. 1–5, doi:http://dx.doi.org/10.1109/ICC.2010.5501789.

[18] Z. Wang, L. Liu, M. Zhou, N. Ansari, A position-based clustering technique for ad hoc intervehicle communication, IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews 38 (2) (2008) 201–208, http://dx.doi.org/10.1109/TSMCC.2007.913917.

[19] C. Lin, M. Gerla, Adaptive clustering for mobile wireless networks, IEEE Journal on Selected Areas in Communications 15 (7) (1997) 1265–1275, http://dx.doi.org/10.1109/49.622910.

[20] J.B. MacQueen, Some methods for classification and analysis of multivariate observations, in: L.M.L. Cam, J. Neyman (Eds.), Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, University of California Press, 1967, pp. 281–297.

[21] D. Dueck, Affinity Propagation: Clustering Data by Passing Messages, Ph.D. thesis, University of Toronto, 2007.

[22] A.T. Ihler, J.W. Fischer III, A.S. Willsky, Loopy belief propagation: convergence and effects of message errors, Journal of Machine Learning Research 6 (2005) 905–936.

[23] J. Mooij, H. Kappen, Sufficient conditions for convergence of the sum-product algorithm, IEEE Transactions on Information Theory 53 (12) (2007) 4422–4437, http://dx.doi.org/10.1109/TIT.2007.909166.

[24] T. Heskes, Stable fixed points of loopy belief propagation are local minima of the Bethe free energy, in: S.T.S. Becker, K. Obermayer (Eds.), Advances in Neural Information Processing Systems, vol. 15, MIT Press, 2003, pp. 343–350.

[25] I.E. Givoni, B.J. Frey, A binary variable model for affinity propagation, Neural Computation 21 (6) (2009) 1589–1600. doi:http://dx.doi.org/10.1162/neco.2009.05-08-785.

[26] F. Karnadi, Z.H. Mo, K. chan Lan, Rapid generation of realistic mobility models for VANET, in: Wireless Communications and Networking Conference, 2007, WCNC 2007, IEEE, 2007, pp. 2506–2511, doi:http://dx.doi.org/10.1109/WCNC.2007.467.

[27] M. Behrisch, L. Bieker, J. Erdmann, D. Krajzewicz, Sumo – simulation of urban mobility: an overview, in: SIMUL 2011, The Third International Conference on Advances in System Simulation, Barcelona, Spain, 2011.

Behnam Hassanabadi received the Ph.D. degree in electrical and computer engineering from University of Toronto, Toronto, Canada, in 2013. His research interests are in the areas of wireless networking, network coding, and medium access control design.

Christine Shea received her B.A.Sc. degree in Electrical Engineering from Queen's University, Kingston, Canada, in 2007. She received her M.A.Sc. degree in Electrical Engineering from the University of Toronto, Toronto, Canada, in 2009. Her M.A.Sc. Research was completed in the Communications Group at the Edward S. Rogers Sr. Department of Electrical and Computer Engineering, at the University of Toronto. Her research interests include vehicular ad hoc networks and belief propagation with emphasis on the design and analysis of clustering algorithms for MAC and Routing protocols. She is a member of the Vehicular Communications Research Group in the Wireless and Internet Research Laboratory.

Le Zhang received the B.A.Sc. degree in computer engineering from the University of Waterloo, Waterloo, ON, Canada, and the M.A.Sc. degree in electrical engineering from the University of Toronto, Toronto, ON, Canada, in 2007 and 2010, respectively. He is currently a Ph.D. candidate at the University of Toronto. His research interests include congestion control, multi-hop forwarding and MAC protocols for vehicular networks.

Shahrokh Valaee (S '88, M '00, SM '02) received the Ph.D. degree in electrical engineering from McGill University in Canada. Currently he is a Professor and the Associate Chair for Undergraduate Studies and holds the Nortel Institute junior chair of Computer Networks in the Edward S. Rogers Sr. Department of Electrical and Computer Engineering at the University of Toronto. He is the founder and the Director of the Wireless and Internet Research Laboratory (WIRLab) at the University of Toronto.

Prof. Valaee is an Editor of IEEE Transactions on Wireless Communications, and an Associate Editor of IEEE Signal Processing Letters. He was the Technical Program Co-Chair and the Local Organizing Chair of the IEEE PIMRC 2011, and the Co-Chair for Wireless Communications Symposium of IEEE GLOBECOM 2006. He has served as a guest editor for several journals including IEEE Wireless Communications Magazine, Wiley Journal on Wireless Communications and Mobile Computing, EURASIP Journal on Advances in Signal Processing, and International Journal of Wireless Information Networks. His current research interests are in wireless vehicular and sensor networks, location estimation and cellular networks.