A NOVEL INTERSECTION-BASED CLUSTERING SCHEME FOR VANET

by

MICHAEL LEE

TRAVIS ATKISON, COMMITTEE CHAIR
XIAOYAN HONG
BRANDON DIXON
RANDY SMITH
ALEXANDER HAINEN

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2021

ABSTRACT

Currently, much attention is being placed on the development and deployment of vehicle communication technologies. Such technologies could revolutionize both navigation and entertainment systems available to drivers. However, there are still many challenges posed by this field that are in need of further investigation. One of these is the limitations on the throughput of networks created by vehicular devices. As such, it is necessary to resolve some of these network throughput issues so that vehicle communication technologies can increase the amount of information they exchange. One scheme to improve network throughput involves dividing the vehicles into subgroups called clusters. Many such clustering algorithms have been proposed, but none have yet been determined to be optimal. This dissertation puts forth a new passive clustering approach that has the key advantage of a significantly reduced overhead. The reduced overhead of passive algorithms increases the amount of the network available in which normal data transmissions can occur. The drawback to passive algorithms is their unreliable knowledge of the network which can cause them to struggle to successfully perform cluster maintenance activities. Clusters created by passive algorithms, therefore, tend to be shorter-lived and smaller than what an active clustering algorithm can maintain. In order to maintain a cluster with a low overhead and better knowledge of the network, this dissertation introduces a new clustering algorithm intended to function at intersections. This new algorithm attempts to take advantage of the decreased overhead of passive clustering algorithms while introducing a lightweight machine learning algorithm that will assist with cluster selection.

DEDICATION

This dissertation is dedicated to my family. I wouldn't have been able to make it this far without them.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1719062 and the Department of Education under Grant No. P200A180058.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

As small scale computing technologies become increasingly commonplace, more and more devices are capable of exchanging meaningful information that assists them in performing their designated task. As a result, a whole new field of study called Internet of Things (IoT) has been created that investigates how to connect different devices in such a way that their exchange of information provides new services that the devices had previously been incapable of performing. One such field of devices that has been investigated is vehicles. Modern vehicles can have a large number of onboard systems, including navigation assistance devices, steering assistance devices, braking assistance devices, and many other devices made to assist the users of the vehicle in the process of driving the vehicle. As such, it is a particularly interesting field to investigate such possibilities. Within this field, my research focuses on investigating Vehicular Ad-Hoc Networks, or VANETs. These networks connect vehicles in an ad-hoc manner with little to no reliance on existing infrastructure devices. However, removing infrastructure devices from the network introduces a number of new challenges to the environment.

Since ad-hoc networks no longer have dedicated central nodes to handle routing and packet forwarding across the network, each node must now be able to perform all such tasks on their own initiative. As such, each node in the network now needs to collect and distribute information that was previously only stored by routers. When this is used in a wireless environment, as it would be in a VANET, this requires the nodes to be able to handle to following tasks: packet forwarding, flooding, and updating the network. In the

case of packet forwarding, since each node in range of a wireless transmission will receive a transmission, they need to be able to look at the intended destination and decide whether or not to forward the packet. In the simplest case, if the receiver is the destination, it obviously should not forward the packet any further. However, if every node that is not the intended destination forwards the packet, the packet would be sent to far more nodes than is strictly necessary, and eventually every node connected to the network would have received the packet. This would accomplish the task, but is very inefficient and cannot be stopped early as the destination node has no way to indicate to the other nodes that it has already received the packet. As a result, the network needs to implement some sort of scheme that will allow nodes to determine whether or not they should forward a packet based on the current state of the network.

Flooding a network refers to the process of distributing a message to every node in the network. In the previous discussion of packet forwarding, I put forth the idea of every node forwarding a packet, and found that it sufficiently distributed the packet to every node. However, this is a highly inefficient process. It will result in many nodes attempting to send their packets at the same time, which will cause the messages to interfere with each other. Because of this interference, neither of the messages will be able to be received by any of the other nodes on the network. Additionally, some of the nodes will be broadcasting their packet exclusively to nodes that have already received it, meaning that they are using network bandwidth for no reason. To prevent this, nodes need to store some sort of network structure information, then use it to determine efficient paths across the network that reach all of the nodes while reducing the number of nodes needed to send their packets to the bare minimum.

Since both packet forwarding and flooding the network require some sort of knowledge of the network to ensure that the tasks are performed efficiently, nodes must also have a way to let other nodes know when they join and leave the network, as well as when the nodes they can connect to change. This information will need to be distributed to all

nodes on the network so that they can create and update their network maps, allowing them to determine optimal paths for packet forwarding and flooding.

One of the schemes used to establish a structure to the network in VANETs is clustering. This process seeks to identify sub-networks of vehicles that can easily communicate with each other. They also elect a cluster head to act as the controller of the network. Clustering schemes can be divided further into single-hop and multi-hop schemes. In a single-hop scheme, all nodes are within direct transmission range of the cluster head. Because of this, the tasks of packet forwarding and flooding within the cluster become very simple. Regardless of the intended receiver of a packet, the sending packet sends it to the cluster head. Then the cluster head sends it to the intended destination if it is a single destination packet, or sends it to all nodes if it is intended to be flooded. While this process is not necessarily the most efficient for forwarding packets to a specific node, it allows for a very simple scheme for forwarding packets and can flood the network with a large amount of efficiency. This is of particular importance in VANETs, as packets that require flooding are very common due to the type of information VANETs handle. VANETs are often used to pass important safety information about the road, which is most frequently relevant to all vehicles within a certain area. Additionally, VANETs are used to provide geographically specific information, such as nearby services or the traffic and situation of nearby roads, which is again relevant to all nodes within a certain region. These sorts of services and many others cause the primary data traffic in VANETs to be broadcast messages [92].

Multi-hop schemes work in a similar way, except each node maintains a path of connections to and from the cluster head, allowing messages to be passed to nodes outside of a single hop [87]. Clustering also allows for communication between clusters via designated gateway nodes, allowing for long distance communications while still maintaining the efficient methods of communication for nearby nodes.

The third task that clusters must perform, updating network information, is somewhat

more complex in these environments. While the algorithm can still rely on nodes being able to send a message saying that they joined a cluster, because vehicles are mobile, they may exit the cluster by leaving the range of it. In this case, the node may not be able to send a de-affiliation message, as they will leave the cluster when they leave their maximum communication range. Additionally, node mobility means that the network structure will be fluctuating constantly, requiring a large amount of updates if nodes want to maintain an accurate model of the current state of the network [58] [87]. As a result, finding a good solution to this problem is one of the biggest challenges in creating clusters in VANETs. As a result, VANETs require an effective approach that can maintain accurate network state information without overcrowding the network with update messages.

My research sets forth an algorithm used to identify vehicles within range of a cluster without requiring excessive network overhead. Rather than a general purpose algorithm, my algorithm was developed for a specific application, namely an intersection-based vehicle crowd-based VANET application. This application, called Vehicle Crowd-based VANET or VC-VANET for short, has the distinct feature of being run in geographically bounded regions around intersections. This is to say that this application will identify clusters of vehicles that gather at intersections, particularly those with traffic lights, and use these vehicles to establish a variety of network features. This can include cloudlet applications that provide many useful services to the connected vehicles [70].

Previously I mentioned that VC-VANET has the particular feature of being intended for use at intersections with traffic lights. However, the question remains: What is it about these intersections that would require a particular clustering algorithm rather than just a general purpose one? Well, one key difference is that many of the vehicles at any particular intersection are going to be stopped instead of in motion, which is different from the conditions that general purpose algorithms are expecting. In particular, stopped vehicles are far easier to group together than moving vehicles. If a general purpose algorithm were to be deployed in such a location, it would fail to connect as many vehicles together as it

could, limiting the performance of the network.

Besides the vehicles being stopped, intersections are also interesting due to the cohesion of vehicles once they start to move. After a light switches from red to green, vehicles will begin to move at a similar time and stay in close proximity for an extended period of time, only spreading out after a fairly large amount of time has elapsed. By taking advantage of this continued cohesion, the algorithm can maintain the larger clusters that were established at the intersection for a reasonably long section of the road. Then, if the algorithm is operating in an area with a large number of traffic lights, it may be able to hand off its intersection-based cluster to the clustering algorithm at the next light before the cluster from the previous light has had time to fully disperse. In such situations, vehicles could rely on my intersection-based clustering algorithm as the main clustering algorithm, only switching back to the general purpose algorithm when in an area with a lower number of traffic lights. As such, an intersection-based clustering algorithm could provide increased service to the connected vehicles over a general purpose algorithm for a significant portion of the total time that vehicles spend on the road.

Another feature of intersections on roadways is that they are stationary. Therefore, any clusters created by the algorithm will only operate within a limited range of a single geographical point, that point being where the intersection is located. This gives the proposed algorithm the ability to more specifically tailor services to vehicles connecting to each intersection, as their location is known and specific information can be tailored about the region around them without having to store a massive database of information about the entire globe. As a result, the algorithm can provide services such as local traffic maps and lists of local services in order of their proximity to the current intersection. This allows the user to make plans for their specific situation while keeping the service lightweight and able to run on the limited processing power of their onboard devices.

By providing this geographically bounded VANET system, there are many services that could be run on this system for users to take advantage of. For example, as previ-

ously mentioned, the algorithm could provide the users with current local traffic information. By recording the size and density of clusters at each intersection and passing this data to all other intersections, the algorithm could create a local traffic map that indicates to the user how long they can expect to wait at each intersection in the area. This information could either be interpreted directly by the user or passed to a navigation service that is run on one of the user's devices that is connected to the VC-VANET in order to better inform car routing decisions. The algorithm could also provide service to both local businesses and the users by having local businesses store information on their business on an RFID chip or other device capable of connecting to a VANET located outside their business. Cars driving past would pick up this data and carry it to local intersections. Then, it would be spread to all vehicles, including those going the other way, notifying them about the business. This could also provide the added benefit of an extremely accurate measure of distance to the business. The mileage could be tracked by the vehicle with the information, then shared to other vehicles when this data is transmitted. In an alternate, lighter weight scheme, the vehicle could record the timestamp where it initially received the signature of the business, then on transmitting the location also transmit the time elapsed from collecting the signature. This would allow vehicles to have an accurate depiction of how long it could take them to drive to the business. Small local businesses could effectively advertise to their consumer base with only a minimal investment required. As such, it is expected that many businesses would be interested in participating in such a service.

Other services VC-VANET could provide include vehicle platooning in intersection-dense areas. Vehicle platooning is the practice of grouping a number of vehicles together and having one vehicle send the navigation signals for the vehicles behind them to follow. Vehicle platooning is most often considered for implementation on highways, as the difficulty of driving is low and there are very few exits and entrances. However, by setting the platoons at each intersection based on the direction each vehicle is intending to leave

the intersection, this algorithm could provide such services in areas with more complicated road topologies. Additionally, intersections are currently a challenging place for emergency vehicles to cross as they sometimes must get through stopped traffic and cross during a red light, both of which force the vehicle to slow down. With VC-VANET, an advanced, beyond line-of-sight, warning can be provided about oncoming emergency vehicles that can assist in the process of letting the vehicle through.

VC-VANET could also be used to provide a number of driving assistance services at the traffic light. For example, if the algorithm could be integrated with the local traffic signals, it could warn cars when the light is about to turn yellow, allowing them to better decide whether to stop or attempt to make it through the intersection. This could help alleviate some of the known problems with the dilemma zone at intersections [119]. Conversely, the algorithm could provide notifications when the light turns green to assist with driver awareness. The algorithm could also work with the vehicles at the intersection instead of the traffic light itself to provide similar services, i.e. warning the user when the vehicle in front of them starts to brake, or braking automatically if the distance between the two cars is judged to be too small. When the intersection turns green, the algorithm could assist with the procedure to start moving again by sending a notification or automatically starting to move when the vehicle in front of the user starts to move.

Other potential applications include accident notifications to emergency services. A large portion of vehicular accidents and collisions occur at intersections, so VC-VANET would be particularly well positioned to notice them [80] [105] [50]. The cluster that detects the accident could then either use VANET systems or connect directly to the internet to notify emergency systems, based on the capabilities of the devices in the network and the permissions VC-VANET would have. Finally, since VC-VANET is geographically bound to intersections, it would create a predictable location where clusters of vehicles that could use internet services would be. This would make it easier to install infrastructure devices that provide internet to a geographical area. Then, the VC-VANET system

could help to manage internet connections and effectively distribute bandwidth to vehicles within range of the connection.

Now that I have established the importance of VC-VANET, I will investigate why creating an effective clustering system is helpful to the functionality of VC-VANET. I mentioned earlier that clusters can be used to establish a network topology, but how does this help the network, and why does the network need such a thing? The primary reason why clusters and other network topologies are employed in VANET environments is due to a problem with scalability in VANETs. In a fully unstructured ad-hoc network environment, the throughput of the network decreases as more devices are added to it. In normal networks, the nodes can rely on infrastructure devices to provide structure and assist in communications to mitigate this problem, but such luxuries are not available in a VANET system. Thus, clustering seeks to divide out a small portion of the network, reducing the size of the effective network from all VANET-capable devices to only those devices that are within range of the cluster. This dramatically reduces the network size and dramatically increases throughput. However, it also reduces the number of devices that a network can connect to, reducing the utility of the network. To compensate for this, cluster-based networks also have systems in place for communications between clusters. This allows the algorithm to connect to a large number of devices while only increasing the effective number of devices by the number of clusters that each cluster connects to. However, the limited channels of a single connection between each cluster requires a large amount of monitoring to ensure that the capabilities of this connection are being used to its upmost. This can be accomplished by caching frequently requested information to reduce the number of redundant requests, and employing other such strategies that are familiar from normal computer networking. In such a way, clusters provide a network topology and capability more similar to traditional networks while still operating in a VANET environment. However, there are still some other concerns, including a limited connection time between clusters and the high mobility of clusters, but I will not be discussing those in this dissertation.

As mentioned, clusters rely on their ability to divide the network to provide better service. My algorithm sets forth a way that the network can decide on the size of the clusters. Why does the size of the cluster matter? There are three factors that can cause a cluster to perform sub-optimally. First, if the size of the cluster is too small and contains too few vehicles, the network provides almost no features, and the larger scale topology is similar to what it was before clustering was implemented due to the extremely large number of clusters. This problem is reduced as the size of the cluster grows; therefore, the algorithm needs to ensure that there are a fairly reasonable number of vehicles included within each of its clusters. The second factor is network connections. If the cluster covers too large of a geographical region, wireless network connections within the network can start to get unreliable, greatly limiting the ability of the network to pass information within the cluster. If the network is running any sort of distributed application, this can cause a significant decrease in performance as response times and reliability will become very low. To avoid this, the algorithm must maintain a reasonably small geographical area for its clusters. The third factor is scalability. As mentioned previously, network throughput in a VANET decreases as the number of devices increases. Even if the algorithm finds a very large number of vehicles within a small enough geographical area that all of them can communicate easily, it would want to avoid connecting all of them to one cluster as doing so would cause the cluster's performance to decrease significantly. Thus, the algorithm seeks to find the point of compromise between all of these problems so that the network can operate normally.

To assist me in the process of determining optimal cluster sizes, I had access to data collected by the Alabama Department of Transportation. This data provided me with the timestamps when vehicles entered a number of intersections around the area of the University of Alabama. The algorithm uses these timestamps to determine the density of vehicles at the intersection based on how frequently the "vehicle entering the intersection" events are occurring. This bed of samples was very helpful to my research, as the different in-

tersections had wildly varying densities based on their location as well as the time of day, giving me a large corpus to work from. Additionally, it provided many intersections in a corridor, which will be used in later works.

With the assistance of this data, I developed my algorithm, Radius Approximator (Rad-Ap), for use with VC-VANET. This application uses data on vehicles arriving at the intersection to predict the current density of vehicles at the intersection, then uses this data to approximate the size of the v-crowd currently at the intersection. By approximating the threshold distance using density, an effective network area can be established more quickly while also reducing the strain on the network that may be caused by using more traditional methods. A VC-VANET system can use my Rad-Ap application to determine the number of vehicles currently located at an intersection while only requiring a minimal set of input data. Rad-Ap was developed with intersections that have traffic lights, but can be extended to other locations as well. I will also introduce a clustering scheme that uses the Rad-Ap algorithm to achieve a more accurate model of local conditions in a passive clustering scheme. Once I have finished introducing the algorithm, I will perform tests on the algorithm. These tests will examine how well Rad-Ap approximates the size of the cluster that can be created as well as the rate of transmission failures that a cluster using Rad-Ap will see. I hope to show that Rad-Ap can achieve at least a 95% success rate in most network scenarios while keeping the suggested value within 93% of the optimum value.

A review of the related literature will be presented in Chapter 2. Chapter 3 will discuss the methods of my proposed algorithm and the choices leading to them. Chapter 4 will introduce the tests used and examine the results. In Chapter 5, conclusions will be presented. Finally, Chapter 6 will discuss some potential future directions for this work.

CHAPTER 2

LITERATURE REVIEW

The literature review will examine the field of clustering in VANETs. It will begin by examining exactly what clustering entails, and look at some of the advantages that clustering affords the network. It will then discuss some of the features provided by the wireless transmission protocol that clusters can use to their advantage, before investigating the development of VANET clustering algorithms and the current state of the field.

2.1 Clustering in VANETs

In general, clustering algorithms have a set of functions necessary for them to function properly. The first of these is neighborhood discovery. Neighborhood discovery refers to the ability of the clustering algorithm to learn information about the nodes around it [118]. This information can vary from more approximate knowledge of how many nodes are connected with very little knowledge of their position, or can be very detailed, containing the location of each nodes, their capabilities for network transmission, their current speed and heading, and other information that can be useful in setting up clusters intelligently [118] [12] [91]. This neighborhood discovery can also include information about nodes outside of the cluster, as doing so can be useful for setting up routing protocols, which this dissertation l will discuss later [104].

Once a cluster has discovered information about its neighborhood, it must perform a cluster head selection. The cluster head is the leader of the cluster, and most of the operations of the cluster are performed through it [26]. When nodes want to pass information

to the rest of the cluster, they most often have to do so through the cluster head. As such, selecting the correct cluster head can be integral to the operation of the cluster [36]. The algorithm will generally use the information collected in the neighborhood discovery phase to determine the best cluster head to select for the algorithm's operation [3]. Once the optimal cluster head has been decided, it will announce that it will be the cluster head of the algorithm to all other nodes within range [40].

Once the cluster head has been announced, the remaining nodes will have to perform the next task of the algorithm, which is affiliation. During the affiliation stage, nodes that are not a cluster head will receive the announcements of all cluster heads within range of them, then decide which one to join. This choice is generally driven by signal strength, though it can sometimes be influenced by other factors like vehicle mobility patterns and cluster services [26] [22] [11] [39]. Once a node has affiliated itself with a cluster, it must then announce to the cluster that it has joined the cluster [14]. This allows the clusters to know which vehicles it is responsible for sending information to as well as how crowded its current cluster is, which may impact future decisions made by the clustering algorithm [22].

Once these steps have been completed, the cluster has been set up completely with a leader and a list of its followers. However, the task of clustering is not yet complete due to a simple problem: vehicle mobility. Since the nodes of the cluster are constantly moving relative to each other, the cluster needs to perform periodic maintenance to maintain its functionality [40]. The types of maintenance that must be performed vary based on whether the node is a cluster head or a normal node in the network [12]. If the node is a cluster head, it must be able to add and remove nodes from the network. Nodes may be added when they first enter into the range of the network or when their connection quality passes a certain threshold. Nodes may be removed from the network if their connection quality becomes too poor, they move too far from the cluster head, or if the cluster gets too crowded, as clusters will start to decrease in performance once they pass a cer-

tain threshold value of vehicles [91] [21] [16] [56]. Also, the cluster head is often responsible for handing off its control to the next cluster head when it is determined that a different cluster head would be sufficiently better or when the cluster head leaves the network [32]. Nodes of the network have the primary maintenance task of monitoring their connection to the cluster head. If it gets too poor or a better cluster head enters into their range, they may autonomously elect to leave the network [78].

2.1.1 Motivation for Clustering

There are two primary advantages of dividing a vehicular network into clusters that cause doing so to create a more efficient network. The first of these is creating a more efficient network topology for flooding. Flooding, or sending a message to all nodes in the network is a particularly important task for VANET systems, as much of the data collected by the system pertains to all vehicles on the road [54]. Information about a vehicle ahead braking suddenly will impact a whole chain of vehicles behind it, and information about local congestion can help all vehicles find a more efficient route, for example [64]. In an unstructured network environment, flooding is accomplished by having each vehicle that receives the message broadcast it to all vehicles in range, which will perform the same task. However, this system is very inefficient as it results in many vehicles receiving the message more than once, and thus consumes a larger portion of network resources than is strictly necessary. Clustering can help reduce the inefficiency of this system by using the network topology to determine more efficient routes for the dissemination of this information [66]. Most typically this takes the form of the cluster head broadcasting it to all nodes in the cluster, as most clustering algorithms require all nodes in the cluster to be within range of the cluster head [6].

The other big advantage of clustered networks comes about from creating a more efficient system for long-distance routing. Long-distance routing is the primary reason why the total network throughput of ad-hoc systems decreases as the number of vehicles increases [111] [27]. By establishing clusters, the network can create single bridges between

clusters, typically between the cluster heads, that all routing between clusters is performed over. This simplified topology allows the network to increase the efficiency of long distance routing in the network significantly [79]. Additionally, if passing messages across long distances starts to increase the network load to the point that the cluster head can no longer distribute information to all of the nodes within the cluster, the fact that it only has a single point of contact with the outside means that the cluster head can elect to no longer pass forward any long-distance messages. This allows it to focus on distributing vital information to the set of nodes in its cluster [42] [27].

2.2 802.11p

As VANET systems and algorithms were being developed, it became apparent that the current network algorithm, 802.11ac, was a less than ideal algorithm for operations in VANET environments. As a result, a new algorithm, 802.11p, was developed to provide an algorithm that was more suited for the challenges of a VANET system [53]. This algorithm introduced many changes at the physical and network level, but for my topic, I will only be discussing the changes at the network level, as they will have a direct impact on how my clustering algorithm will function [53] [2] [76].

2.2.1 Basic Service Set Revisions

One of the major features of previous algorithms that 802.11p revised was the Basic Service Set (BSS). The BSS was a part of previous networking algorithms that allowed a group of computing devices to communicate with each other over a single shared channel [24] [53] [71]. These BSSs set up a single redistribution point, and each node connected to the BSS sends and receives information only with the redistribution point of that BSS. Any traffic that needs to be distributed to other nodes on the BSS will be routed through the redistribution point [88]. This allows a client-server type of relationship to be set up in an unstructured network. However, these BSSs have the distinct disadvantage of requiring a long handshaking procedure to initialize the BSS, as when a listening node receives

a beacon it must perform a large number of interactive steps. This includes authenticating itself with each of the other devices on the network and associating with the set [76] [31]. This takes up a long period of time and will consume a large amount of the network's capabilities if used in a low throughput environment like VANETs [53] [71]. There is a variant on BSSs that was made for ad-hoc environments called Independent BSS (IBSS), but it unfortunately requires the same extended handshaking procedure of normal BSSs [24].

2.2.2 WAVE Mode

Since the services these systems provide could be useful for VANETs if the time and resource costs were lowered, 802.11p modified these systems to increase their utility in VANET environments. The first focus of these changes was to lower the time required for the handshaking process by decreasing the number of packets that nodes have to exchange to enter a BSS. As such, the first change made was that BSS services will always use a single frequency so that nodes attempting to connect to the network do not have to scan for it. Additionally, BSSs will always use the same Basic Service Set Identifier (BSSID). As the name might suggest, a BSSID is an identifier that connecting nodes can use to determine which BSS they have connected to [53]. When a BSS packet is received by a node, they accept it if and only if the BSSID of the packet matches one of the BSSIDs of the BSSs to which they have connected [31]. This mode of operations allows vehicles to pass messages to this common BSS at any point in their travels, allowing them to broadcast important safety information to all nearby vehicles without needing to join the BSS first. This greatly reduces the lag time between a vehicle receiving important safety information that is relevant to all vehicles on the road and when it will first be capable of broadcasting such information [76]. IEEE 802.11 already reserves a wildcard BSSID intended to be used for management frames, which is specifically the binary value composed of all ones. 802.11p repurposes this wildcard BSSID so that it can be used both for management frames and important safety information [53]. Since it repurposed a preexisting BSSID, this additional service does not reduce the number of BSSIDs available to systems that are

using BSSs in 802.11p [76]. This mode of communication is referred to as Wireless Access in Vehicular Environments (WAVE) mode [31] [53].

While WAVE mode provides a convenient distribution system for safety information and other such information that should be distributed to all nodes in the network, it does not provide a system that is helpful for clustering. This is the case since nodes can only indiscriminately broadcast to all nodes that are interested rather than a more specific set of interested nodes within an area. As such, 802.11p also introduces a new system for these services that does not require the extended handshaking process of the old BSS system. This new system is a variation of a BSS that is called a WAVE BSS. The basic process of establishing a WAVE BSS works as follows: First, a station transmits an on-demand beacon. This form of beacon does not have to be retransmitted periodically and provides a list of services that the WAVE BSS offers. A node that could be interested in joining the WAVE BSS can receive the beacon, use the list of services provided by the WAVE BSS to decide whether it wants to join the WAVE BSS, then configure itself to join the WAVE BSS using only the information contained in that on-demand beacon. Since this process only requires the exchange of a single beacon, it removes the handshaking process required for previous BSS implementations entirely. This allows systems to join a BSS with very little lag from the time when the node first enters the range of a WAVE BSS [53] [71]. Since this greatly reduces the overhead of the network in terms of both time and network capacity and total time connected to a cluster can be very short in the high mobility environment of VANETs, these changes can cause a very large percentage-wise increase in the total throughput of VANET applications using BSSs [76].

2.3 Active Clustering Techniques

Clustering techniques for VANET can be broadly classified into two sets of algorithms: active algorithms and passive algorithms. Active algorithms require the nodes to communicate with each other in order to advertise their location and capabilities, then use this

information to help set up the network. In order to provide a timeframe for nodes to communicate, active algorithms have a clustering phase of the network communications. During this timeframe, nodes will typically advertise their name, location, speed, and other more application-specific information that the algorithm uses to decide the cluster head. The cluster head then advertises itself to the other nodes on the network, at which point the nodes decide which cluster head to join. At this point, the algorithm enters its normal phase of network operation [27] [11] [39] [115] [93]. Passive algorithms, on the other hand, do not have the detailed exchange of information at the start, and instead use approximate methods to determine the cluster head. This cluster head will advertise itself, allowing other nodes in the area to enter and join at their own discretion [27] [84] [66]. These systems will also typically have the old cluster head elect the next one to ensure the cluster head maintains all its previous information while still keeping network correspondence low [66] [112].

Active algorithms and passive algorithms have some trade-offs between their network capabilities. Active algorithms maintain a more rigorous map of the internal network structure, allowing them to better evaluate such things as the optimal cluster head and how to rout packets across clusters for long-distance communications. However, this comes at the cost of a much higher network overhead, as the network needs a handshake when initially being set up and a large amount of continued communication and handshaking to perform maintenance on the network, including adding new nodes when they come within range and dropping nodes when they leave the area of the network [27] [84] [45]. Additionally, they require the extra clustering phase before they can begin operating, and many models of active networks assume that all nodes are stationary during this phase, which is less than ideal when the system is attempting to setup clusters on the move in a vehicular environment [95] [94] [6] [46].

Passive algorithms, on the other hand, have a much reduced handshaking procedure, often relying on using the WAVE BSS system to advertise their network in a single beacon

and simply having the cluster head broadcast to all nodes in the BSS, allowing nodes in range to listen to the message if they decide to join the BSS [84]. Any other control messages are embedded at the start of other packets the nodes in the network send, greatly reducing the overhead required to setup the network. However, this greatly reduced overhead comes at the cost of a decreased knowledge of the network topology. Since the nodes do not spend any time collaborating on their positions, the cluster head only knows approximately how many nodes are in the current network based on the responses to their messages [113] [84]. This decreased knowledge of node topology means that cluster head selection will not necessarily be optimal, as the network collects few metrics that can be used to assist in the election of the cluster head [27].

2.3.1 General Ad-hoc Algorithms

This section will investigate some of the previous clustering algorithms created for this field of study. This section will begin by investigating the first clustering algorithm for ad-hoc networks, DARPA packet-radio network [33] [55] [72] [25]. This algorithm set out to provide a way for a relatively small number of nodes that are in motion to create their own small network between them that also provides access to outside networks. It also wanted to ensure that the network could be made secure and was resistant to some jamming attacks. This first algorithm contained many of the features that are still seen in modern algorithms, such as a distributed control structure for entering into the clusters and a cluster head that works as a leader for the cluster. It also implemented gateway nodes: nodes that can be used to connect to other clusters. These could be broadly divided into two categories of gateway nodes: 1) gateway nodes that are actually within two different clusters and can communicate with both, or 2) gateway nodes that were within range of another cluster's gateway node, and by corresponding with them can communicate with another cluster [33].

This algorithm was later used to implement mobile ad-hoc networks. The first of these selected the cluster head by finding the node with the highest number of connections in

the network [37] [89] [107]. In order to prevent cluster heads from being established very close to each other in crowded areas of the network, this algorithm only considered links to nodes that were not already within clusters. As a result, all cluster heads will necessarily be out of range of each other. This algorithm provided a series of gateway nodes that could be used to provide long-distance routing, but did not directly provide an algorithm for doing so [37]. As a result, it was later extended to add support for routing [22]. Additionally, it was extended again to provide the ability to offer channel access management for additional security [34].

After this algorithm was released, many future algorithms were made based on the concepts used. One of the most influential of these was DMAC, a Distributed Mobility-Adaptive Clustering Algorithm [27] [11]. DMAC was based on a Distributed Clustering Algorithm (DCA), an algorithm first published in the same work documenting DMAC. DCA assumed that nodes were stationary while the clustering was first being performed. Each node would advertise its own connection quality as well as number of connections. This connection quality parameter would be used to determine a weighted value for each node. This weighted value would, then, be used to select which node would become the cluster head. As a result, even if a node was not the most central node in the region, it could still become the cluster head if it could send and receive a greater quantity of data, unlike the previous algorithms, which were based solely on location. This total connection quality weight was determined by imagining the network as a graph, and summing the values of all edges connected to the node, where the value of the edge was the amount of data that could be sent across their connection. The cluster head would be elected to be the node with the highest sum of all their edges. Once the cluster heads were selected, nodes would select a cluster head within range based on which of the cluster heads had the largest total sum of weight, then affiliate themselves with that cluster. If a node had already joined a cluster and a better cluster head arises within range, nodes were capable of switching to join the superior cluster head. Additionally, if all of a node's neighbors left

the cluster, it would then switch modes back to its initial state, evaluating its own state to see if it is worthy of becoming a cluster head. If it is, it would do so, and if not it would joining the best nearby cluster head instead [11] [57] [1] [17].

One of the advantages of this algorithm is that nodes now only have to communicate with their one-hop neighbor. Since the previous algorithm required clusters to be well-separated, nodes had to communicate with their two-hop neighbors to ensure that the next nearest cluster was set up far enough away from them. Meanwhile, DCA only required nodes to collaborate with their immediate neighbors to determine best weights for cluster heads, then gave nodes an algorithm for selecting the best cluster head so they would not enter a sub-optimal cluster if two arose within range of them [11] [1].

One of the weaknesses of DCA, however, was that it did not have any metrics to account for mobility. As such, it was extended to become DMAC. DMAC introduced a new metric to cluster head selection that took into account node mobility. Now, if a node had the best connection but was rapidly moving away from the rest of the group, DMAC would not select it as a cluster head. It would instead select a more stable node. Additionally, the maintenance phase of the network was updated to check for new nodes that had moved within range and old nodes that had their connection fail due to moving out of range. As a result, DMAC was capable of working within a mobile environment [11].

DMAC was later evaluated for its effectiveness in mobile environments [39]. It was found that the lifespan of clusters in the network was quite poor, preventing a stable clustered environment. However, node speed was found to have little impact on the life of the clusters. The created clusters were also highly unstable, as nodes would constantly reaffiliate from one cluster to another as the total connection quality varied slightly among the cluster heads [39] [114]. As a result, Generalized DMAC (G-DMAC) was introduced, which increased the amount that mobility impacted cluster head selection. Additionally, it added some contingencies to help reduce node reaffiliation, helping the network become more stable. It also somewhat increased the efficiency of the initial step of creating the

clusters [39].

2.3.2 Seperating MANET from VANET

As research in the field continued, it became apparent that these algorithms and their variants were not the most suited for vehicular environments [100] [105] [98]. These algorithms were optimized for mobile ad-hoc networks, or MANETs, which are in general based around the concept of many network nodes that are being carried around by humans rather than vehicles [7]. As these nodes are human-borne, they move much slower than nodes in a vehicular network tend to [49]. As a result, MANET algorithms would often require an increased setup time to connect nodes to the network [109] [18]. This works in a MANET environment where the minimum time it would take for a node close to another node to move out of range is relatively large, in the range of tens of seconds to a minute, but in a vehicular environment where nodes move much quicker, this timeframe can be reduced to single seconds or less, depending on the range the connection can maintain. As a result, MANET algorithms often struggle in the faster paced vehicular environment [51] [98].

The other reason why MANET algorithms are not the best suited for use in VANETs is due to the decreased randomness in node movement [82] [9] [81] [5]. While vehicles may move quickly, they have to follow a much more rigid series of paths and rules that dictate where they can move [81]. As a result, algorithms particularly adapted for VANET environments can take advantage of this rigidness in vehicular movement patterns to improve their performance [82] [9] [67]. Meanwhile, MANETs largely have to assume that its nodes move randomly, preventing them from taking advantage of any of this behavior. As a result, MANET algorithms fall behind VANET algorithms in performance in a vehicular environment due to their lack of optimization [7].

2.3.3 Weighted Network Metric-Based Techniques

This section will investigate VANET techniques by looking into some common active clustering techniques and discussing their particular advantages and disadvantages.

Active technique can be broadly classified by the methods they use for cluster creation and cluster head selection. Some of the more common methods used are Weighted Network Metric-Based Techniques, Precedence-Based Schemes, and Timer-Based Approaches. Many other approaches have also been proposed, but these methods constitute the majority of clustering algorithms currently being developed in the field. The method used will have a large impact on the performance of a particular clustering algorithm, and will offer certain advantages and disadvantages. This section will begin by investigating the advantages of Weighted Network-Metric Based Techniques.

Weighted network metric-based techniques are some of the techniques that were first introduced for use in VANETs, and as a result are some of the most frequently used [27] [47] [90] [86]. They have each node evaluate its own suitability as a cluster head, compare itself to the other nodes around it, then announce itself as a cluster head if it is the best in the area. Nodes will then join a cluster head based on which cluster head they can exchange messages with has the highest fitness [90] [27].

One such algorithm that uses this method of clustering is the Vehicular Weighted Clustering Algorithm (VWCA) [28]. VWCA bases its selection partially around "distrust", a value that represents how likely the node is to drop connection or leave the area. As the node performs "distrustful" acts, like losing connection or starting to pull away from the rest of the vehicles, its distrust value increases, lowering its likelihood of being selected as the cluster head [28]. This differs somewhat in implementation from some of the other weighted algorithms that have been previously discussed in that it is searching for the node with the lowest number rather than the highest. The Adaptive Mobility-Aware Clustering Algorithm based on Destination (AMACAD) also uses a metric where lower numbers are better. It uses the distance between nodes to determine the ideal cluster head, while also factoring in the relative velocities of the vehicles into its distance selection. Their relative velocities give an indication as to what their distances will be in the future [86]. AMACAD is also interested in using the final destination of each node in its

determination of the ideal cluster head, as this allows the network to predict many of the turns the vehicles will make, giving it a reasonable idea of the long term behavior of all of the nodes. There are many other algorithms that are based around distance and relative velocity, some of which also add in more metrics to be considered [4] [19] [97] [83]. For example, TDMA Cluster-based MAC (TC-MAC), also considers turning direction, while User-Oriented Fuzzy-logic-based Clustering (UOFC) attempts to predict future relative velocities using the acceleration of the vehicles [75] [6]. UOFC also attempts to predict the driver's intended destination to understand how the vehicle's path will change in the long term [75].

Meanwhile, there are plenty of VANET algorithms that still use high numbers as their fitness method, such as Mean Connection Time Clustering (MCTC) and Multihoming Clustering Algorithm for VANETs (MCA-VANET) [23] [110]. As its name might suggest, MCTC primarily considers how long a node has stayed connected to the nodes around it. A node that has spent a long period of time maintaining a large number of successful connections has proven that it has a good connection and maintains a stable position in the middle of other vehicles, making it a fine choice for a cluster head [23] [60]. Meanwhile, MCA-VANET uses the number of consecutive "Hello" messages received as its primary decision-making metric, where a "Hello" message is the message sent by a node as it enters the region of the network. This makes sense as a metric, as a node that has received more "Hello" messages would tend to be still around the most nodes and has given evidence that it is in a good location to add new nodes to the network [110].

2.3.4 Rawashdeh's et al. Algorithm

One of the more commonly referenced active clustering VANET algorithms is an algorithm put forth by Rawashdeh et al. [93]. A key feature of this algorithm is that it divides the nodes within transmission range into different categories based on how far they are from the cluster. Vehicles within the maximum possible transmission distance are handled by its long range control channel, while those closer to the source pass their commu-

nications across shorter range service channels. The short range channels are primarily intended for communication within a cluster, while the long range channels are used to correspond with far away nodes and possibly other clusters [93]. This classification of vehicles into different categories based on their distance allows the cluster to reduce how much it interferes with transmissions of nearby vehicles and clusters. It will only use its maximum distance channel when it is required to do so, and will otherwise attempt to limit its transmission range.

This algorithm has all nodes share their information by periodically broadcasting their data across the long range channel. The sent information includes the node's position, velocity, number of nodes within transmission range, heading, and the ID of the cluster they are currently subscribed to, if any. As nodes listen to these messages, they classify their neighbors that are within a transmission radius of twice the range of the shorter range communication channel based on whether they are considered to be stable within that range or are not stable. Neighbors that are considered to be stable are ones that are traveling in a similar direction with a similar velocity, and thus seem likely to remain within their current range increment for a reasonable period of time [93]. This excludes out nodes that could be traveling the opposite direction or in a much faster lane, helping to increase the stability of formed clusters as they have to deal with less transient nodes connecting to their cluster.

When it comes time to form a cluster, the vehicles classified as stable neighbors within two transmission ranges are separated further into nodes that currently have a greater velocity than the current node and those that have a lesser velocity. The node with no vehicles slower than it or only vehicles that are in different clusters that are slower than it will send an InitiateCluster message to all of the stable neighbors slower than it. All of these nodes that receive this message and are not currently in a cluster respond to this message, and begin calculating their suitability as cluster head. Cluster head suitability is calculated based on the other nodes' relative positions and velocities, as well as how many other

nodes the current node can connect to within the shorter transmission distance. Normalization is performed on each of the parameters to ensure that no one parameter dominates the result. Once a node has calculated its suitability as cluster head, it waits for a period of time inversely proportional to how suitable that node is as a cluster head as well as being proportional to that node's historical congestion. Since a cluster head will have to pass most the messages of the cluster, the system attempts to avoid selecting nodes that have been shown to spend a large amount of their time processing other information, as they will have less of their network capabilities available to pass messages for the cluster. Then, if, while the node is waiting, another node announces itself to be cluster head, it heeds the request of that node and joins it as a member of the cluster. If not, the node's timer will expire, at which point the node will announce itself as cluster head, and the other nodes within the shorter transmission range will join it. The ID of the newly formed cluster is set to be the same as the cluster head's personal ID [93].

Nodes can also choose to affiliate with a previously existing cluster. For this process, the node considering joining will check their relative positions and velocities. If this node determines itself to be a stable member of the neighborhood within the shorter transmission range, the node will send a JoinCluster message to the cluster head, and the cluster head will add it to the list of the members of the cluster. If the joining node is within range of two possible cluster heads, it will calculate how long it will spend in transmission range of each of the cluster heads based on their current relative positions and velocities, then send the JoinCluster message to the cluster head it will spend more time near. When a node leaves a cluster, it will enter the non-clustered state, and attempt to join another nearby cluster or join other non-clustered vehicles to create a new cluster, depending on the current state of the network. Both the cluster head and the leaving node will be able to tell when the node leaves the cluster, as they will still be periodically sending beacons across the longer transmission range. These periodic beacons allow the nodes to see that they have grown too far apart [93].

This algorithm also allows for the clusters to merge. This occurs when two cluster heads detect each other within the shorter transmission range and they are evaluated to be stable neighbors of each other. When this occurs, the cluster head with fewer component members gives up on being a cluster head, allowing its members to return to the non-clustered state. Once they return to the non-clustered state, the vehicles may choose to join the nearby cluster or form their own cluster, based on the preexisting rules for non-clustered nodes [93].

2.3.5 Rawashdah et al's Algorithm Limitations

This algorithm, while popular, showcases how active methods require significant network overhead. The broadcasting of node information will be the highest source of overhead in this system, as it requests that nodes send their information periodically in beacon messages across a very large transmission radius. As the number of nodes within this radius grows, it becomes easy to see how nodes inside this radius will have to send and receive an extremely large number of packets on this channel. Due to the high mobility of VANET environments, the beacons must be transmitted at a frequent rate, with the DSRC recommending beaconing every 100 milliseconds [48]. With 802.11p's specifications of a maximum transmission range of one kilometer, this means that vehicles within two kilometers of roadway will all be contending with any given node for transmitting packets across the channel [2]. This can include a very large number of vehicles even in the scenario of a highway without any congestion or other nearby roads that could have vehicles also creating their own clusters. Since DSRC also recommends a packet size of 500 bytes, each vehicle will require a channel bandwidth of 40 Kbps to send their beacons. This means that any environment with over 150 vehicles within a two kilometer span will exceed the DSRC recommended throughput of 6 Mbps for a control channel, causing a large amount of network information to be lost [48] [93] [84]. These metrics don't even take into account time for backoff between network nodes or network bandwidth wasted due to packet collision, so can be seen how this command channel will easily become con-

gested.

While it is apparent that command channels can become congested when using this algorithm, it is a separate issue from the utilization of the channel used for transmissions within the cluster. This is the case because data transmissions are intended to take place on a separate channel than the control channel, and thus will not be interfered with by the overhead of the control channel. However, it is worth noting that this algorithm does require two channels to function properly, as each node can only connect to a relatively small number of channels at once. Because of this limitation, algorithms utilizing too many channels can also be a serious drawback for nodes to consider. During cluster formation, only one node sends a notification to start the cluster formation process, and only one node sends a message to notify the other nodes that it has chosen to be a cluster head. However, each node joining the cluster must send the cluster head a JoinCluster message, meaning that the total network utilization is proportional to k, the number of vehicles in the cluster. Leaving the cluster does not require any messages to be sent, but joining an existing cluster requires a message for each node that joins. Additionally, merging two clusters requires only a single UndoCluster message broadcast to all members to disassemble the smaller cluster, but afterwards the unclustered nodes will either join the existing cluster or attempt to form their own cluster, both of which processes require a number of transmissions proportional to k [93] [84].

2.3.6 Weighted Network Pros and Cons

Weighted network metrics can ensure that cluster heads are only created outside of the range of any other cluster heads, eliminating many problems that can arise from cluster heads contending with each other [27]. Additionally, they provide a relatively simple solution to the problem of generating clusters. On top of this, they can be tuned for a particular network situation, as methods that incorporate multiple metrics can have the relative importance of these metrics adjusted to better match the situation. This makes them a flexible approach to the problem of cluster creation that can provide a guaranteed level of

service with a relatively simple implementation [27] [114] [103].

One of the biggest difficulties facing weighted network selection strategies is the dynamic environment of VANETs. The constant changes in the network conditions within a vehicular network causes the weights of the nodes to fluctuate, possibly causing the "most fit" node to change in the process, which can cause the network to want to hand off its cluster head. Additionally, since these changes are only the result of minor fluctuations, they can quickly revert, causing yet another cluster head hand-off. This problem, if left unchecked, can cause a copious amount of the network's time to be spent on re-selecting the cluster head [84] [102]. This causes weighted network approaches to have to start applying filters to the network to reduce the likelihood of cluster head hand-off from temporary changes in the network [15] [32]. Some methods use machine learning to assist in selecting long-term cluster heads, rewarding selections based on packets routed and length of time spent as cluster head [44] [99]. Other algorithms simply require the new cluster head to outperform the old one by a certain threshold, indicating that this change in the network is significant and unlikely to simply revert after a short period of time [15]. Yet other approaches implement a timer system, where a certain time period must pass before cluster head hand-off is allowed to occur. If other nodes can maintain a better weighted value for a long period of time, it indicates that the network has changed in a way less likely to quickly revert [116]. Other algorithms seek to reduce the excessive merging of clusters, and thus will only do so after the two cluster heads come within a certain distance of each other [108] [62].

Another problem with this method is the selflessness of cluster selection. Nodes will join the most fit cluster head within range regardless of whether it is the best cluster head for them [27] [10]. This can lead to nodes that are connected to very good cluster heads that they can only maintain a very poor connection with, leading them to be unable to gain or grant much information to or from the cluster [84]. This kind of behavior is less than ideal, as it would be best if each node were capable of using its connection to the

best of its ability. This problem and the algorithm's difficulties with unstable networks can lead it to be unsuited for many situations.

2.3.7 Precedence-Based Schemes

A popular alternative to weighted network selection strategies is precedence-based schemes. These operate in a way somewhat opposite to weighted network selections in that instead of a node advertising how fit it is, each node will make their own evaluations as to the best cluster head for them to join. Much like the weighted network selection schemes, these will use weighted sums of a number of metrics. However, since the node that wants to join is making the observation, this allows it to make a selfish selection that is best suited to its particular needs and network connections. This eliminates the previous problem where weighted selection schemes could cause nodes to connect to cluster heads that they couldn't actually exchange much information with due to connection quality [40] [20]. Additionally, it allows a node to select a cluster by the services provided. If a node is in need of particular information, it can seek it out [115]. On top of this, it can allow the network to be set up in such a way that nodes do not have to particularly join cluster heads, and can instead join nodes they have a better connection to that themselves have a clear connection to the cluster head [40].

Precedence-based schemes can employ many of the same metrics as weighted network schemes. For example, Robust Mobility Adaptive Clustering (RMAC) employs a precedence-based scheme that selects from its neighbors based on distance between them, their relative velocities, and whether or not they are already a cluster head [40]. The obvious new metric here is the evaluation of whether the node is already a cluster head. When cluster heads were being selected in weighted network schemes, the algorithm was already under the assumption that there were no more fit cluster heads nearby. Also, from how this metric is worded, it can be seen that nodes will be biased towards cluster heads, but if a nearby node has a much better connection than any cluster head in range, it might be elevated to cluster head status to provide a cluster to the nodes nearby it [40].

Algorithms based around precedence-based schemes provide greater stability than weighted network schemes, particularly for nodes that would otherwise be on the edge of the network in a weighted network scheme [27] [38]. As a result, these schemes can be used to form hierarchical structures from the cluster node outward to serve a larger cluster area [29] [61]. However, methods that create a hierarchical structure have been shown to have a predisposition towards forming chains in the network rather than the more ideal structure of a tree. This results in a very inefficient network transmission structure in many environments. This reduces the amount of information that can be passed through the network, and increasing the total number of hops required when used to run a long-distance routing algorithm [27]. This problem has been mitigated in some works by declaring a maximum number of hops that can exist between a cluster head and one of its members [35].

Another disadvantage of precedence-based schemes is that the lack of coordination between cluster heads can result in an increased number of cluster heads being created when compared to weighted network schemes [27]. Since creating each cluster head in a network and setting up hand-off methods requires a certain amount of network overhead, an increased amount of cluster heads means that the overall overhead of the network will increase [27] [84] [106]. If this increase in overhead becomes excessively large, it can result in a greatly decreased functionality for the VANET system it is running.

2.3.8 Timer-Based Approaches

A different scheme for cluster head selection and node affiliation is a timer-based approach. Timer-based approaches are based around a simple idea: a node waits in an unclustered state for a set period of time. If during this time it has received no beacons from a cluster head within range, the timer expires and the node announces itself as a cluster head. On the other hand, if the node does hear a cluster head within this time period, it will not announce itself as a cluster head and instead immediately affiliates itself with the cluster head it received a beacon from [95]. In some approaches, like Cluster-Based Location Routing (CBLR), the time a node waits is constant and based on other network

wait times like Arbitration Inter-Frame Spacing (AIFS) and round-trip times [95] [13] [45]. In other approaches, this wait time is instead randomly selected in hopes of reducing the odds of two nodes declaring for cluster head at the same time [30]. In some algorithms, instead of the node joining immediately when it hears a cluster head, it will employ some of the methods used in precedence-based schemes to evaluate how well the advertising cluster head works for it, and only join it if it passes a certain threshold [95]. This allows these timer-based approaches to avoid some of the problems encountered in weighted network schemes where nodes that have very bad connections with a faraway cluster head will still attempt to join it. As a result, they suffer a lower quality clustering experience [45] [1].

The main advantages of timer-based approaches lie in the speed and simplicity of the algorithm. Since cluster heads will simply form if there aren't any cluster heads in range, nodes don't need to spend time passing messages to assess who among them would be the best cluster head to elect [95]. Additionally, beacons can contain less information about the cluster head when methods similar to precedence-based schemes are not being employed because nodes do not base their choice of a cluster head on any metrics that the cluster head needs to advertise. It is merely done based on whether there is a cluster head in range or not [6]. As a result, these methods provide a much more lightweight approach to setting up VANET clusters and no longer require a separate phase of operation for when the network is first being set up within an area [45] [95] [79].

Unfortunately, this increased simplicity and lowered overhead come with some drawbacks. One of the main problems timer-based approaches can encounter is the hidden node problem, where two network nodes outside of the range of each other repeatedly broadcast their packets to a node in the middle simultaneously and get no feedback indicating why that node keeps failing to receive their packets [6]. In this particular situation, the hidden node problem would arise when multiple nodes outside of range of each other reach the end of their timers and broadcast their intentions to be cluster head at the same time. This will cause them to accidentally interfere with the other nodes' broadcasts and causing

none of the declarations to go through. This causes the nodes that between the declaring nodes to be unable to detect the cluster heads that have already declared near them during their wait timeframe, possibly causing them to declare unnecessarily and increase network overhead [6].

2.4 Passive Clustering Techniques

This section will focus on some of the passive techniques that have been created for VANET systems. Passive techniques were first introduced in a study that sought to increase the energy efficiency of ad-hoc networks [65] [27]. This study was for mobile ad-hoc networks, not vehicular networks in general, but it laid some of the groundwork for passive techniques that were later adapted to VANET settings. This approach solved some of the issues with the original algorithms for active clustering, including the need for clusters to be logically separated from each other as well as the problems with the cluster setup phase in which the nodes of the cluster had to remain stationary while the network was being set up, and could only begin to move once the initial clusters had been established [11] [65]. Since nodes are normally mobile over their lifetimes, being able to set up a cluster while the nodes are in motion is very important to the performance of any mobile ad-hoc network, but particularly so for vehicular ones. This initial approach put forth two ways for clusters to advertise their information: 1) the information could be embedded in the MAC field of distributed packets, or 2) the information could be put in the protocol field of the network layer information of the packet. While the MAC field would give quicker access to the information, the paper recommended putting the information in the network layer due to the need to change hardware to allow for variations in the MAC field [65] [2]. Nodes that wish to join a cluster listen to all packets in promiscuous mode, searching for the information in the packet that specifies details of a nearby cluster. Once it finds a suitable cluster in range, it can join that cluster [65]. This process eliminates the need for explicit beaconing, reducing network overhead.

This algorithm also introduced the First Declaration Wins strategy for cluster head selection, in which any node that is not in a cluster can choose to declare itself to be a cluster head, at which point other nodes will join it [77] [65]. While this method is not particularly refined, it allows for the establishment of cluster heads with no overhead and relies on the declaring vehicle being fairly near the middle of the cluster in the average case [65] [63]. Much like the first active MANET algorithm, nodes could be in one of four states: 1) INITIAL, a node not yet in any cluster, 2) ORDINARY, a normal member of a cluster, 3) CLUSTER_HEAD, the cluster head, or 4) GATEWAY, a node that is capable of transmitting information to another cluster. These states will be included in the packet header for each packet that the node passes, so that other nodes in the network can get a rough estimate for the number of nodes in the cluster and the roles of nodes around them [65].

This initial approach was later modified to better match the requirements of a VANET network. These variants on the passive clustering technique were VANET Passive Clustering techniques, or VPCs for short. They modified the algorithm by changing the way the algorithm selected and transferred cluster heads. In order to maintain the primary characteristics of passive techniques, this algorithm kept the method of First Declaration Wins (FDW) for the initial cluster head selection. However, once the cluster head was selected this way, nodes would begin to communicate and learn information about each other, then use this information to transfer the cluster head. This information would be embedded within the traffic of other packets the nodes were sending, so transferring this information would not significantly increase the strain on the network. Some of the information this type of network could collect includes vehicle density, link quality, and link sustainability [112]. Once this information was collected, much like a weighted network scheme, the algorithm would compute the node with the highest fitness. In order to prevent some of the transient changes that plague naive weighted network implementations, the network would wait a period of time inversely proportional to the difference in fitness between the old cluster head and the new proposed cluster head. Because this relationship is inversely

proportional, the network can still transition quickly when a much better cluster head candidate is detected, but will be slow to transition to a new head if it is only slightly better. This allows it to sure that its slight improvement will be long term enough to actually benefit the network. If, after the time period has elapsed, the new proposed cluster head is still better than the old head and the old cluster head did not veto the transfer, the cluster head will transition to the new proposed node [112].

Other methods of cluster head selection have been adapted to work with passive clustering methods. For example, the algorithm called Cluster Formation for IVC looked at using an approach based on relative speed like some of the earlier active algorithms discussed in this dissertation [59]. It did this by grouping vehicle speeds to regions that were similar enough, then having each node on the network be aware of the speed group to which they currently belong. The nodes would then affiliate with the cluster head that belonged to that grouping once it saw packets being sent that advertised in the header that they were of an appropriate speed group. If the node's speed changed sufficiently, it would pass a speed threshold, and attempt to join a cluster in a speed group more similar to its own [59].

Another passive technique, called PassCAR, used a weighted random selection scheme based on the location, velocity, cluster state, and link lifetime of nearby nodes [113]. Nodes would collect these packets to learn information about the network structure. When the time came to select a new cluster head, nodes would wait a varying amount of time before declaring themselves as the cluster head. The time they wait is inversely proportional to how effective they believe they would be as the cluster head. This allows for a more accurate cluster head selection once enough information has been passed across the network. In a relatively stable vehicular environment, like a highway scenario, vehicles will be able to make increasingly informed decisions about the next cluster head during the cluster head hand-off process that occurs when the old cluster head leaves [113].

2.5 Hybrid Active-Passive Clustering Techniques

One of the biggest challenges of passive techniques is that they can only accurately grasp the state of the network around them when the network traffic they are handling is high. When the traffic around them becomes low in quantity, they struggle to maintain any semblance of understanding of their network conditions, and thus begin to perform quite suboptimally. Since network traffic tends to be sporadic, this severely limits the utility of passive clustering algorithms. As a result, investigations were begun into the possibility of using a hybrid active-passive clustering technique to help make up for some of the deficiencies of passive techniques [84].

One such clustering scheme was appropriately called a Hybrid Active-Passive VANET Clustering Technique. This technique worked by having nodes operate in one of two modes: active or passive. In an active mode, both the cluster heads and the nodes in the cluster would transmit beacon messages to notify other nodes in the cluster of their correct status, as well as notify nearby nodes that might be interested in joining the cluster. Active mode will be used when the algorithm is first initiated, as well as whenever the node's association with the cluster is lost. Meanwhile, passive mode will be used while the cluster is being maintained and the current network is being utilized enough to maintain a passive network. In passive mode, the cluster head itself still transmits beacons, but the other nodes in the cluster cease to do so, dramatically lowering the overhead of the algorithm, as well as reducing the probability of packet collisions.

When this clustering technique is operating in active mode, it will use Basagni's method to maintain and set up clusters [11]. This method was chosen since it was one of the more commonly used ones when this algorithm was established. Additionally, Basagni's algorithm had many favorable features for this Hybrid Active-Passive algorithm, including a multi-channel approach, efficient cluster head selection, and cluster formation. The author notes, however, that their method could be used with any active algorithm in the general case. The switch from active mode to passive mode will occur when the congestion of the

control channel gets to be too high. In order to know when the control channel is becoming congested, some of the channel congestion control algorithms in VANETs must be investigated.

Estimating the parameters of a VANET's connection is difficult to represent properly due to the dynamic propagation conditions of VANETs in realistic scenarios. Obstructions from buildings, infrastructure, and differences in elevation affect how network signals will propagate in ways that are difficult to predict [52] [96] [84]. As a result, it becomes difficult to tell the current maximum capacity of the channel being monitored and how congested the channel currently is. This also makes it difficult for proactive congestion control models to function properly, as they attempt to determine the ideal parameters for network connections to reduce the congestion on the network based on the conditions of the current network. These differences that set it apart from other networks are why the field of congestion control must also be handled separately in VANETs.

Congestion control in VANETs tends to rely on schemes based on one of two methods: reactive schemes or hybrid schemes. Reactive schemes function much as they sound. They monitor the current condition of the network and adjust certain variables of the network connection when congestion is detected to help the network function more optimally. These schemes monitor a number of different types of variables, including channel utilization level, occupancy time, and number of queued messages. Congestion is determined to be occurring once these variables pass a certain predefined threshold. Hybrid schemes are a mix of proactive and reactive methods, where they use proactive methods to monitor and adjust variables that can be easily accommodated through proactive techniques, but use reactive methods on other variables that are less suited for a proactive algorithm in a VANET environment. Zang et al. implemented a system designed specifically for use in VANETs which measured the channel utilization level based on the version of the Distributed Coordination Function (DCF) used in WAVE, the transmission standard for VANETs [117]. In this technique, a node in the network could continuously listen to

the channel and periodically calculate the channel utilization by dividing the total time it took a node to send a packet, including the DIFS time and the average backoff time, by the control channel interval, which is typically set to 100 milliseconds in 802.11p [117] [53] [85].

This hybrid active-passive algorithm uses this same formula to determine the approximate channel utilization of the control channel when it is acting in active mode. Once this rises over a certain threshold, the system switches to passive mode, modifying a Boolean value in its beacon so that nodes, upon receiving the beacon, know that the system is now acting in passive mode and they should stop beaconing. This method of notifying nodes removes the need for a separate message to let nodes know to switch to passive mode or to active mode, helping reduce the overhead of the system [84].

While the system is operating in passive mode, nodes besides the cluster head will no longer send beacon messages, and will instead embed their mobility information in their normal network messages, which, as is the case in most cluster scenarios, are only intended for the cluster head. Nodes will also schedule a packet at a less frequent interval that is intended to be broadcast to all of their neighbors and contains their mobility information so that the other nodes in the cluster can maintain some of their neighbors' mobility information [84].

Now that this dissertation has discussed active and passive systems, some of the tradeoffs between them have become apparent. The current state of passive algorithms is such that they will be very inaccurate when data communication is low, and current hybrid algorithms still require nodes to frequently explicitly update their information. As such, I seek to create an algorithm that uses only the bare minimum of active information to create a prediction of the current network capacity, then base its cluster selection off of that prediction.

CHAPTER 3

METHODOLOGY

3.1 Proposed Clustering Algorithm

In order to create a better VANET clustering algorithm based around the principles of a passive network, I propose a new version of VANET Passive Clustering (VPC) that uses a lightweight learning algorithm to gain an approximate grasp of the current network conditions even when network traffic is low. My new clustering algorithm kept the cluster head selection algorithm of VPC, but added in two new capabilities: the ability of the cluster head to exclude nodes from the cluster and a cluster density approximator.

The need for the cluster head to be able to exclude nodes from the cluster may not be clear, so I will elaborate on it further. In any ad-hoc network, the total throughput decreases as the number of nodes increases. While clustering largely prevents this from being a concern, in sufficiently dense scenarios an excessive number of nodes can still become problematic. Since, as stated earlier in the dissertation, this algorithm is being developed for use at intersections, it is still very possible that the density will exceed the threshold number of nodes where an increased amount is better for the network. "Excluded" nodes will operate in a state in which they can only send high priority messages, such as emergency alerts and safety information, to the cluster. The node will still be capable of receiving information from the cluster, if desired. This process is performed by having the nodes evaluate their own fitness as the cluster head, which represents how capable the node is of communicating with the cluster. The cluster head will then include the threshold value

nodes must meet to attempt to communicate with the cluster in its beaconing. Since the current thresholding system has smaller numbers represent better connections, nodes with a value beyond the beaconed number will be considered to have been told to back off. These nodes should only resume sending messages when their fitness value surpasses the threshold, whether it be due to changes in the threshold or their fitness.

The other reason that the ability to exclude nodes from the cluster was introduced was to reduce incidents where far off nodes with weak connections to the network attempt to send and resend messages many times. These incidents can consume a large amount of the channel bandwidth when there are a large number of nodes closer to the center of the cluster with a much better connection that are forced to wait on a low-performing member of the cluster. Since nodes selfishly choose to join the cluster based on their own evaluation of the network, this creates a sort of doubly selfish network, where only nodes that want to join and are allowed to stay in can be part of the network. This doubly selfish network provides many of the benefits of precedence-based schemes while avoiding some of the pitfalls that they can encounter.

The other feature of my new clustering scheme is the cluster density approximator. This approximator has two proposed modes of operation: first, a lightweight active mode in which the approximator collects "Hello" messages from when a node first enters range of the cluster and uses this to generate an approximate density at the intersection based on the number of "Hello" messages received within a timespan. Due to the way this approximation is handled later, "Hello" messages that are not received due to interference will not affect the average case, so there is no need for a handshaking to confirm that the "Hello" message is received. The second mode of operation is a fully passive mode in which the density approximator counts the number of unique message originators that it has received during a timespan. By averaging these results over a longer period of time and relying on the lower mobility situation at traffic lights, the algorithm can help to mitigate some of the problems with sporadic network traffic in fully passive clustering networks.

In an ideal form of operation for this clustering method, the second method should be as accurate as the first, as nodes should contribute any useful information they have accrued to the clustering algorithm upon joining the network. As in a fully passive mode, this report can be used to also announce affiliation within the network. However, in the early stages of operation of this algorithm, the number of services may not be sufficient to require each vehicle to send new information to the cluster upon joining. In such scenarios, the first mode of operation would provide greater accuracy in establishing the current density of the intersection. Due to this reason and to simplify testing (as the tests will no longer have to establish payloads for join messages to piggyback off of), both modes of operation were introduced.

When a node is first recorded as joining a cluster, the cluster head will temporarily record the ID of the joining node. During the time in which this ID is recorded, any messages signifying the node is in the cluster will not be counted as a new node joining the cluster, and will instead refresh the timer associated with that node ID. If the node timer runs out, that node is considered to have left the cluster, allowing that node to rejoin and be counted as such if it returns to the area of the cluster again. This system will reduce the impact of nodes that are on the edge of the network and consider themselves to be frequently leaving and re-entering the cluster. Nodes that do not consider themselves to have left the cluster only need to indicate that they have entered the cluster on the first message, as nodes that are sending information to the cluster head are assumed to either be in the cluster or have sufficiently important information that cluster alignment does not matter.

It is important to note that this algorithm is intended to be used at traffic lights. As a result, the algorithm must consider how to handle the cases of green lights as opposed to red lights. The answer is simple enough: one of the evaluation components of nodes considering joining an algorithm must be their current velocity and acceleration. If they appear to be decelerating with the intent to come to a stop, then they can join the cluster.

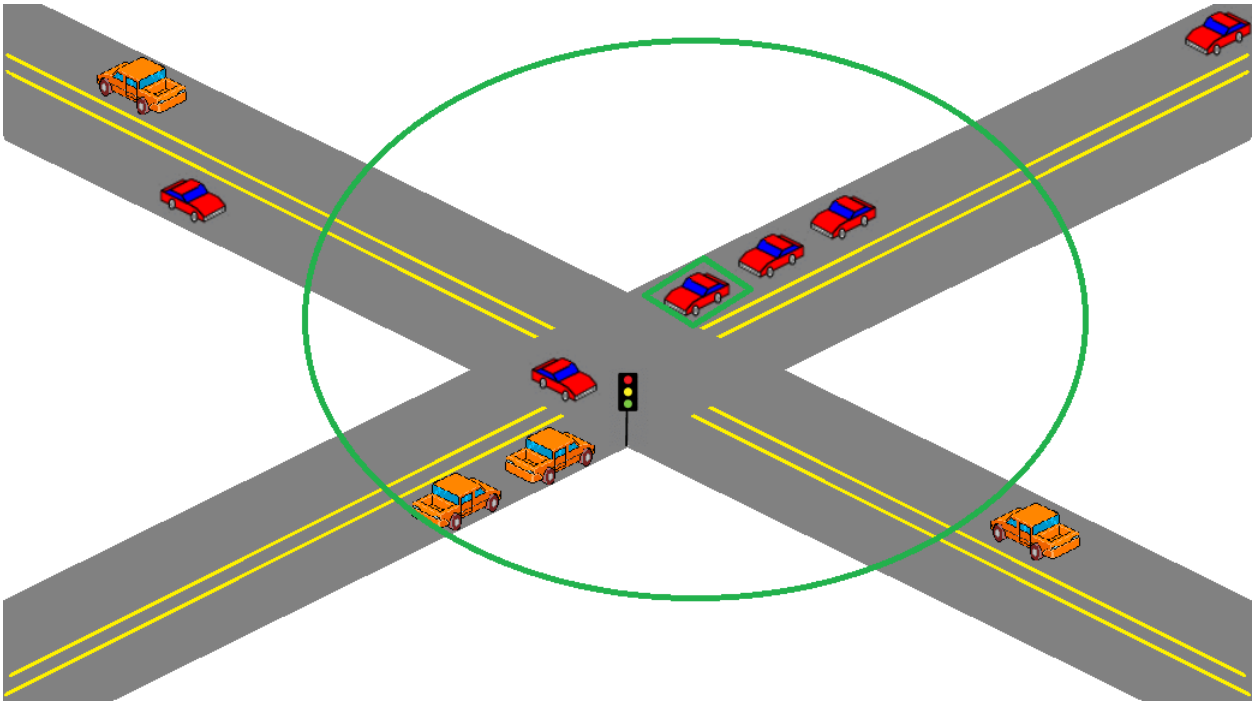

Figure 3.1: Representation of the algorithm's radius value

If they are not slowing down, then that indicates that their light is green and they would gain no benefits from using my clustering algorithm. An additional benefit of my system is that if they begin decelerating but do not come to a stop due to the light changing, they will still catch up to the vehicles in front of them and be able to use my system's cluster, so the deceleration evaluation threshold can be pretty generous.

In Figure 3.1, we can see a representation of how the radius would select vehicles on the network. The vehicle in the green diamond represents the cluster head in this scenario, and the green circle represents the radius from the cluster head in which nodes can pass messages to the cluster. We can see how the system excludes nodes outside of the area due to them being too far from the cluster to reliable send messages. However, if they receive any messages successfully, they may still learn information from the cluster. Nodes like the vehicle in the intersection would most likely elect to not join the cluster, due to still being in motion, but could still listen in on the cluster's messages to collect information.

In a more dense environment, vehicles within a reasonable transmission range may be

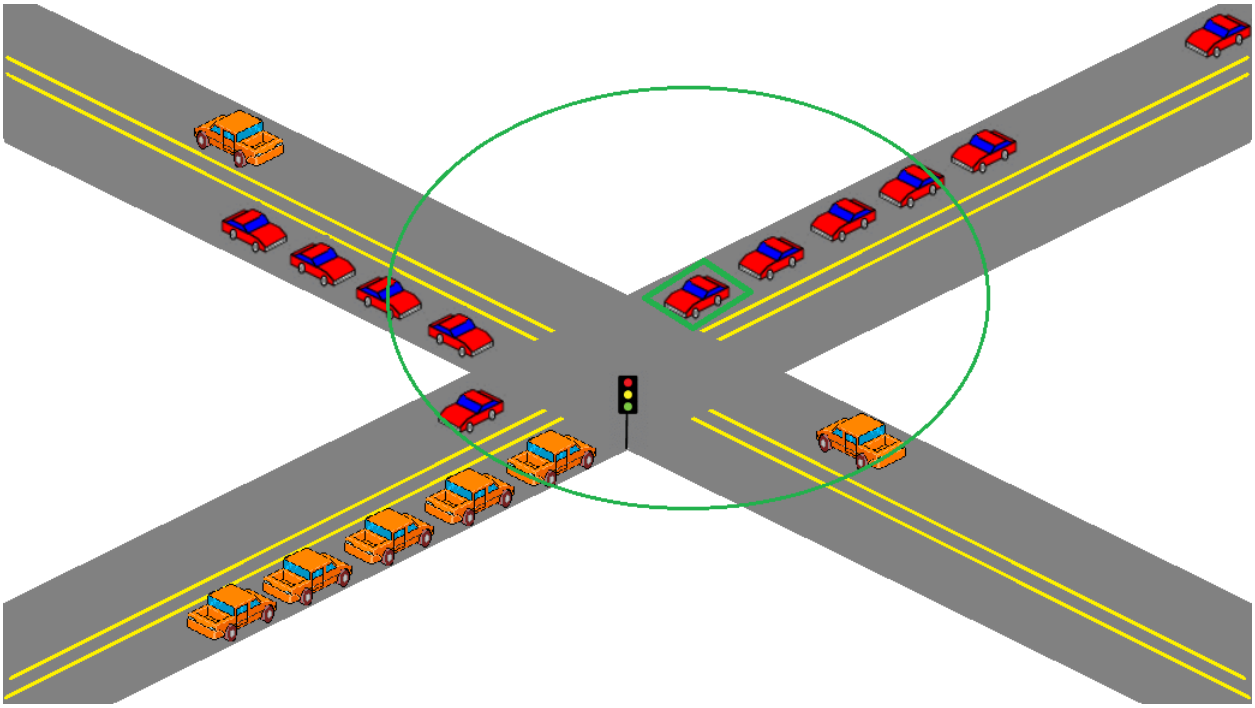

Figure 3.2: Representation of the algorithm in a dense environment

excluded from the cluster to prevent the network from becoming overcrowded. In Figure 3.2, we can see an example of how this would work. Only the vehicles within the green circle in this scenario are allowed to transmit messages to the cluster. While some of the vehicles outside of that area are perfectly within normal transmission range, allowing all of those such vehicles to transmit would result in fewer packages actually being transmitted, due to packet collisions. Instead, only the vehicles that are most likely to be able to successfully transmit their packages are allowed to do so.

This boundary where the number of successful transmissions begins to decrease will vary based on many factory. The qualities of an intersection themselves may affect them, as a four-lane intersection will become overcrowded on a smaller radius than a one-lane intersection. An intersection may also find itself in trouble if extending the radius past a certain point causes it to include another intersection that contains a large set of cars with a much weaker connection to the cluster head. Even things like the road's speed limit can affect the intersection, as a slower road may be able to include more cars that are in

Characteristics
Lightweight Algorithm
Unit Agnostic
Affected Less by Parameter Selection
Records and Distributes Training Data
Highly Accurate
Secure

Table 3.1: Algorithm Characteristics

motion, while a faster road would encounter many failed messages if moving cars were included. Compounding these conditions, a single-road intersection with a moderate number of slow-moving cars that are moderately spaced out would have a very high radius, while a massive intersection filled with fast-moving cars would have an extremely small radius. Because of this, we divide up our solution so that a unique instance applies to each particular intersections. However, a stable value is not enough for each intersection, as the number of vehicles currently at each intersection will also affect the range of vehicles that can be allowed to transmit messages. Hence, our algorithm needs to be able to understand and adjust to different conditions of the intersection.

3.2 Rad-Ap Characteristics

Next I will discuss my density prediction algorithm, Rad-Ap. This section will investigate the features of this algorithm in detail, as well as the creation and development process for the algorithm.

The first step in creating this algorithm was to determine the requirements that Rad-Ap needed to work in a VC-VANET environment. Six overarching characteristics were identified that Rad-Ap had to satisfy in order to be considered fully functional. These characteristics can be found in Table 3.1. Not satisfying all of them could lead to an unsuited algorithm that doesn't function properly or even an algorithm that is completely unable to be run for its intended purpose [68] [69].

3.2.1 A Lightweight Algorithm

The first characteristic identified was the need for a lightweight algorithm. Since Rad-Ap is going to be run as part of a network management system, reducing the processing power required by the algorithm to the bare minimum makes the system far more responsive and able to handle large amounts of traffic. Additionally, the system is going to have to be transferred across cluster heads frequently because it operates within geographically bounded areas that vehicles will only be within for a relatively short period of time. As such, the total size of the algorithm in terms of both program size and memory must be kept quite low. Furthermore, the system is intended to be run on a combination of embedded devices in cars as well as larger devices, such as smartphones, which may be contained within the vehicle. If the system is being run on embedded devices, all resources must be kept to a minimum as these systems often do not have very large amounts of processing power and system memory. Additionally, when the algorithm operates on larger devices within the vehicle, such as smartphones, these devices are often run on only battery power. In such situations, it is advisable to keep power consumption as low as possible. This requires reducing the amount of resources that the algorithm uses as much as possible. Due to all these factors, an algorithm created for this particular situation cannot require too much from the capabilities of its host system.

3.2.2 Unit Agnostic

The next characteristic identified was the need for an entirely unit agnostic algorithm. This requirement was not mandated by the environment that the system is being run on; however, it will greatly increase the utility of Rad-Ap. In a real world system, it is quite possible that units of measurement could change from one region to the next. If my system only worked with a certain type of measurement, it would greatly decrease the utility of the system. Making the algorithm fully unit agnostic, however, is not a trivial task, as one of the possible ways to measure distance in a network is round-trip time. Since this will often be a very small unit of time, it is possible that it could be stored in units of mil-

liseconds or as just normal seconds, which would give considerably different ranges of operation. As such, I had to take care that the algorithm would work for values both in the thousands and less than one, which meant that constant values for alteration had to be avoided to the maximum extent possible. After all, if the algorithm at some point in its operation increased its recommended radius by one, but the expected output values are all considerably less than one, the algorithm would be hard pressed to function properly for those values. This is why I had to take special care that Rad-Ap was agnostic to the range of the values it would be outputting.

3.2.3 Affected Less by Parameter Selection

Another characteristic identified is the need for Rad-Ap to run effectively without requiring the user to spend excess effort on parameter selection. In other words, I wanted the algorithm to be able to perform at least reasonably well even when considerably subpar parameters were chosen for the environment it was being run on. In order to accomplish this, I wanted the algorithm to be able to determine characteristics of the environment to some extent and modify its parameters such that it could tune them to be more in line with the conditions of the environment. This can greatly reduce the amount of trial and error required to employ this algorithm in a real-world condition, as well as allowing it to be deployed completely untrained into most any environment. This condition to greatly increase the speed at which this algorithm can be deployed allows it to serve as an initial approach to the problem when these networks are first used in a real-world environment.

3.2.4 Records and Distributes Training Data

Since this algorithm is designed to be very easy to use as a first approach, Rad-Ap also needs to be capable of recording and distributing data that other locations using this algorithm, or even future algorithms, can use as assistance in solving this problem. By fulfilling this criteria, the algorithm would be easier to spread after the initial deployment. Future versions of Rad-Ap could use the data collected by the first deployed version of the algorithm, allowing them to start with more optimally tuned parameters. Addition-

ally, this improves the lifespan of the algorithm's service. If later down the road it becomes clear that improvements on the current algorithm used are necessary, future research can create a better informed algorithm using the data collected by the current version of it. Furthermore, the ability of this algorithm to spread information can be used directly for other purposes, such as creating traffic maps of the area and finding optimal routes for cars to get to their destination while reducing wait time in traffic to a minimum.

3.2.5 Highly Accurate

The fifth characteristic for this algorithm was for it to be as accurate as possible. This means that I had to ensure that Rad-Ap produced as large of a recommended value as possible without causing the number of failed transmissions on the network to drop significantly. The reason for this requirement is fairly straightforward: the closer the algorithm is to the optimum value for the cluster size, the better the associated clustering algorithm can perform. It is in everyone's best interests to ensure that the algorithm can produce values as close to the optimum as possible in as short of a timeframe as possible. In addition, I prioritized keeping the network's failure rate low over maximizing the cluster size in the early phases of the algorithm so that it would not attempt to tie up too many vehicles and cause a large amount of network traffic even while it was initially learning about the environment.

3.2.6 Secure

Finally, Rad-Ap needs to be sufficiently secure. There are two main sub-tasks I had to accomplish to sufficiently secure the algorithm. First, I had to ensure that passing the algorithm bad data could not cause the algorithm to enter an error state. To achieve this, I must ensure that the algorithm can successfully parse all forms of input and does not fail when handling numbers and values outside of the realistic scope, including negative values and other such odd inputs. The other sub-task was to ensure that the data the algorithm stores does not contain any sort of information that can be used to put users of the system at risk. This data is going to be passed across the network often, so even with proper

transmission security protocols there is still the risk of it being accessed. I identified potentially dangerous information as any information that could help an attacker locate a specific vehicle or notice trends in the travel patterns of a particular vehicle. Removing any such data from the algorithm allow users to connect to the system without concern of being tracked or otherwise inconvenienced by malicious agents.

3.3 Initial Version

This lead to the initial idea for Rad-Ap: An Additive Increase, Multiplicative Decrease algorithm. These algorithms are often employed in network environments due to the very low computational cost of the algorithms, and they work much as they are named: When the network has a good result, you increase the load of the network by a single step. On a bad result, you decrease it very sharply, often halving it or otherwise reducing it by a multiple of the current load. These algorithms maintain a high success rate by shying away from failures very quickly, and manage to spend the majority of their time near the optimal capacity of the network. Additionally, they require no additional data collection from the network, reducing the required overhead to extremely low levels. However, this algorithm alone does not suffice to solve the problem. Since it increases additively, the initial value for the amount to be added has a strong impact on the network performance. The smaller the initial value is, the higher the success rate of the algorithm is while the lower the performance of the algorithm is. More troublingly, the additive increase makes strong assumptions about units. In a worst case scenario, if the amount added is greater than the minimum functional value, the algorithm fails completely. Such a failure is possible in this system. If round-trip time in seconds is used as the primary measurement, then the optimum threshold could be a value less than 0.1. In such an environment, if the algorithm had an additive increase of 1, it would never be able to appropriately encapsulate such values. The recommended value would be stuck at a minimum of 1, which is a considerably suboptimal value when the true value lies less than 0.1. Additionally, the algorithm can

not avoid such a problem by simply making the added value extremely small, as doing so could cause the system to be highly inefficient when run on units that are not in such a small range.

In order to solve this problem, I switched from an additive increase to a multiplicative increase. It allows Rad-Ap to ignore the impact of unit size on the results of the operation and maintain a constant success ratio and network utilization regardless. The format for the increase equation on a success is:

$$NewThreshold = OldThreshold * (1 + IncRate)$$

On a failure, use instead

$$NewThreshold = OldThreshold * (1 - DecRate)$$

where *DecRate* must be less than one and *IncRate* must be less than *DecRate*. However, this once again was not enough on its own to be a full solution to the problem. Vehicle densities at intersections go through a large amount of variation over time, as you would not expect vehicular density at rush hour to be the same as the density at two in the morning. This variation means that the average optimal cluster size for the intersection is also going to be changing over time, going through many different phases within a day or even within an hour. If Rad-Ap tries to employ a straight Multiplicative Increase, Multiplicative Decrease algorithm in such conditions, the algorithm is always going to be playing catch-up, forcing it to operate in such a way that its numbers can vary quickly. This means that the algorithm is going to have to maintain an only acceptable success rate with its increase value and its decrease value relatively close to each other.

3.4 Classifying by Density

To prevent Rad-Ap's Multiplicative Increase, Multiplicative Decrease algorithm from always having to chase after the optimal condition as the density changes, I divided up the algorithm such that a separate instance of the Multiplicative Increase, Multiplicative Decrease algorithm is maintained for a large set of small ranges of densities. This way the algorithm can ensure that the optimal value that it is seeking remains stable within a relatively small region. Since the optimal value will now be relatively constant, it becomes far more feasible for the algorithm to slowly hone in on its location. In order to accomplish this, there are two things the algorithm must be able to do. First, the algorithm needs to have an effective way of estimating the density of vehicles at the intersection. If the algorithm has an inaccurate representation of the density, then the algorithm will be predicting on an optimum value that can vary despite the predicted density being constant, which greatly decreases the effectiveness of the algorithm. Second, the algorithm needs to establish a methodology that allows Rad-Ap's Multiplicative Increase, Multiplicative Decrease algorithm to slowly increase its precision so that it can attempt to hone in on the optimal value instead of maintaining a constant set of fluctuations.

In a normal network, determining the density of devices near the network is fairly trivial. Simply broadcasting a ping to all devices in range and then waiting for their responses allows you to determine exactly how many are listening to your network. However, there are a number of different factors that make this more difficult in a VANET environment. First of all, the throughput of the network is very limited, so steps are taken to minimize it as much as possible. The larger the overhead of the algorithm, the worse it is for network performance. However, a single ping and response, while it will consume the network's capabilities for a short period, is not enough to ruin even the small throughput of a VANET over a long timespan. This is where the second part of the problem comes in. Vehicles are very mobile, and will be entering and leaving the network frequently. They often leave the network without being able to complete the handshaking procedures be-

fore leaving the network. This means that even if the algorithm pings all the vehicles in range, the high mobility of the nodes will make that information useful for only a very short period of time. Additionally, while the algorithm can expect a signal from vehicles when they connect to the network, the fact that it cannot rely on being able to tell when they leave the network means that it cannot simply update its count for number of vehicles on the network based on the entry and exit events that the network sees. As a result, if the algorithm were to use the method of pinging all vehicles in range, it would have to repeat this ping very frequently. While a single ping does not consume a significant portion of network resources, having to ping repeatedly will cause a very large percentage of the network's resources to be consumed on maintaining the network. This reduces the utility of the network significantly, making it far less useful to the end user. As a result, the algorithm needs a solution that allows it to estimate the number of vehicles connected to the network without consuming such a large portion of network resources.

To accomplish this, I turn to the results of a previous research which investigated how to best approximate vehicular densities at intersections using the intersection events recorded by the Alabama Department of Transportation [43]. As mentioned previously, these intersection events record when the under-road vehicle detectors detect a vehicle passing above. In other words, each event represents a vehicle entering the intersection. While using the data collected from the infrastructure device doesn't intuitively sound related to how VANETs can get their data in a live environment, VANETs can also collect the events of vehicles entering the intersection by recording the event of vehicles joining their network. As a result, this study is very helpful to the efforts of determining the vehicle density of intersections in a lightweight way.

This study put forth the following systems for estimating vehicular density: First, the algorithm separates out weekdays and weekends. Weekdays have fairly consistent patterns of vehicular density, but they do not line up with weekends densities at all. Meanwhile, while there is still some difference between Sundays and Saturdays, they do follow largely

the same patterns as each other. As a result, dividing up the system into weekdays and weekends allows for a great increase in its ability to predict the density. Next, the algorithm divides the time again based on the current hour of the day. Vehicle traffic varies greatly over time, but the variance within a single hour is fairly low. Additionally, since traffic is influenced by human patterns and many events begin and end at the turn of an hour, many changes in traffic patterns occur on or near the ends of the hour. Once the algorithm has divided up the data into its time category, its stores the most recent 168 events associated with that time category and use those to estimate the current density. The study showed that of the number of stored events they tested, this level had the best mix of precision and responsiveness, while also limiting the number of events the algorithm has to store as much as possible. In order to calculate the density, the number of items in the queue is divided by the time difference between the most recent item and the least recent item.

$$Density = SizeOfQueue/(MostRecentTimestamp - LeastRecentTimestamp)$$

In order to efficiently access the most and least recent items, a double-ended queue is used. Because the hours are divided out to process them separately, the most recent and least recent item could span a division of one or more days despite there being car entries during that period. For these cases, the distance is calculated in such a way that each queue considers time to elapse only as a single cyclic hour. Now that the algorithm has a good estimation of the current density, it can ensure that it is holding the optimum value stable for its Multiplicative Increase, Multiplicative Decrease algorithm so that it can make progress towards the correct value.

In Figure 3.3, there is a basic representation of how the algorithm works. It receives a timestamp representing a vehicle entering the intersection and uses this information

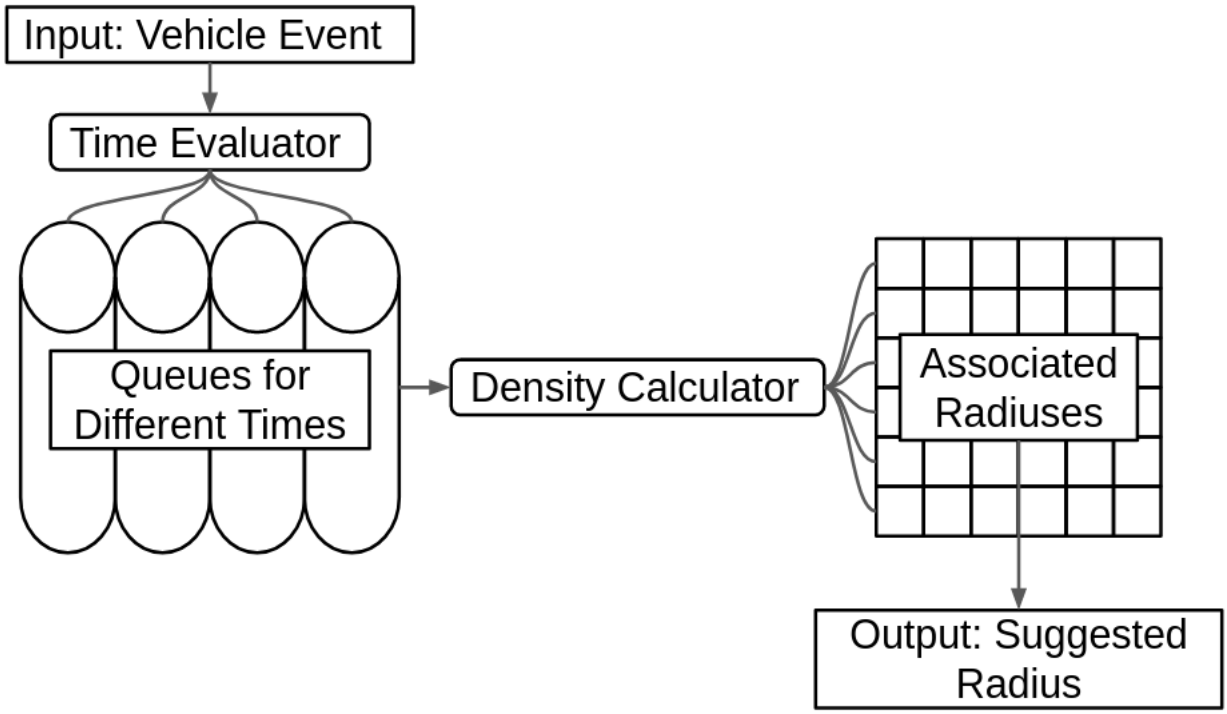

Figure 3.3: Representation of the algorithm's data structure

to adjust its information on current intersection density for whichever timeslot the vehicle belongs to. When it receives a request to report the current recommended radius, it uses these queues to determine the current density. It then queries its internal data structure, storing the cutoff values or radii to determine which one belongs to the current density and reports this information to the requesting entity. When it receives feedback on whether a transmission was a success or not, it follows a similar process; however, instead of reporting the retrieved cutoff value, it updates it based on whether the transmission was a failure or a success.

3.5 Increasing Precision

Now that I have investigated how to make the optimum value stable, I will look at the steps my Multiplicative Increase, Multiplicative Decrease algorithm can take to allow it to slowly increase its precision over time. In order to accomplish this, I introduced a new variable, *Goal* [41] [8]. *Goal* represents the desired success rate for the program to reach as

it operates. The higher it is set, the higher the success rate is. However, the average value suggested decreases somewhat as the desired value increases because the algorithm spends longer and longer periods of time underneath the radius where failures start to occur due to increased interference or the end of signal range. Additionally, if the value of *Goal* is set higher than the system can reasonably achieve due to contention from other messages and the general reliability of message passing across the network, it can cause a drastic decrease in the generated radii as the algorithm seeks to make it lower and lower to reach its desired success value. This dissertation will later discuss some of the methods used to mitigate this concern in this system.

For Rad-Ap, the *Goal* value is implemented by looking at the number of sequential messages that have been successfully passed across the network. When this number is sufficiently larger than its *Goal* value for number of successes, the algorithm increases the rate that its Multiplicative Increase, Multiplicative Decrease algorithm increases at that particular density. It does this by increasing its value for *IncRate* by 1% at a time. This *Goal* value corresponds to a success rate because a system with a success rate of 95% must average 19 successful messages passed in a row across all of its messages. In a much similar fashion, when the number of sequential messages is sufficiently smaller than *Goal*, the algorithm decreases its rate of growth, allowing it to spend more time in the range of numbers where it will be able to succeed more easily. To accomplish this, *IncRate* is instead decreased by 1%. Since the decrease rate stays constant, this will slowly increase the success rate when it is too small, as well as slowly decreasing it when it is too large. This is indicative of the system only operating within an area where it is extremely likely to succeed, and thus not covering as much area as it could. This type of behavior causes a notable decrease in the performance of the algorithm. When making this algorithm, I took care to ensure that the point at which the algorithm would speed up its growth and the point at which it would decrease its growth were sufficiently far from each other. If these two points were to instead be very close to each other, the system would have a large

amount of variance in performance. This would be the case because any minor statistical variation in number of results in a row could cause the system to adjust its *Goal* value, which would then force it to adjust back almost immediately. Having the values be somewhat separated helps reduce the likelihood of statistical variance causing the system to increase or decrease its growth rates notably beyond what would be optimal for its current situation.

3.6 Increasing Growth Rate

Rad-Ap can now grow towards an optimum value and settle towards this value once it has approached it successfully. However, depending on the configured parameters, it could take quite a while to reach the optimum value. As a result, I created a new system to help speed up the process. In order to do this, I introduced a new variable to each Multiplicative Increase, Multiplicative Decrease algorithm instance that tracked the number of times it had sequentially received a success or a failure. As this number increased, so too would the growth rate of the algorithm in whichever direction the sequential signals were being received. If many successes were accrued in a row, it would increase the pace at which the algorithm increased; if multiple failures were accrued in a row, the rate of decrease would increase. This allows the algorithm to quickly adapt to the optimum value even when the initial value is significantly far away. This makes the final increase and decrease formulas as follows:

$$NewThreshold = OldThreshold * (1 + IncRate * SequentialSuccesses)$$

$$NewThreshold = OldThreshold * (1 - DecRate * SequentialFailures)$$

As a result, in most runs of the proposed algorithm *IncRate* started at 0.0005 and *DecRate* started at 0.1.

In Figure 3.4, there is a demonstration of how this algorithm grows the radius asso-

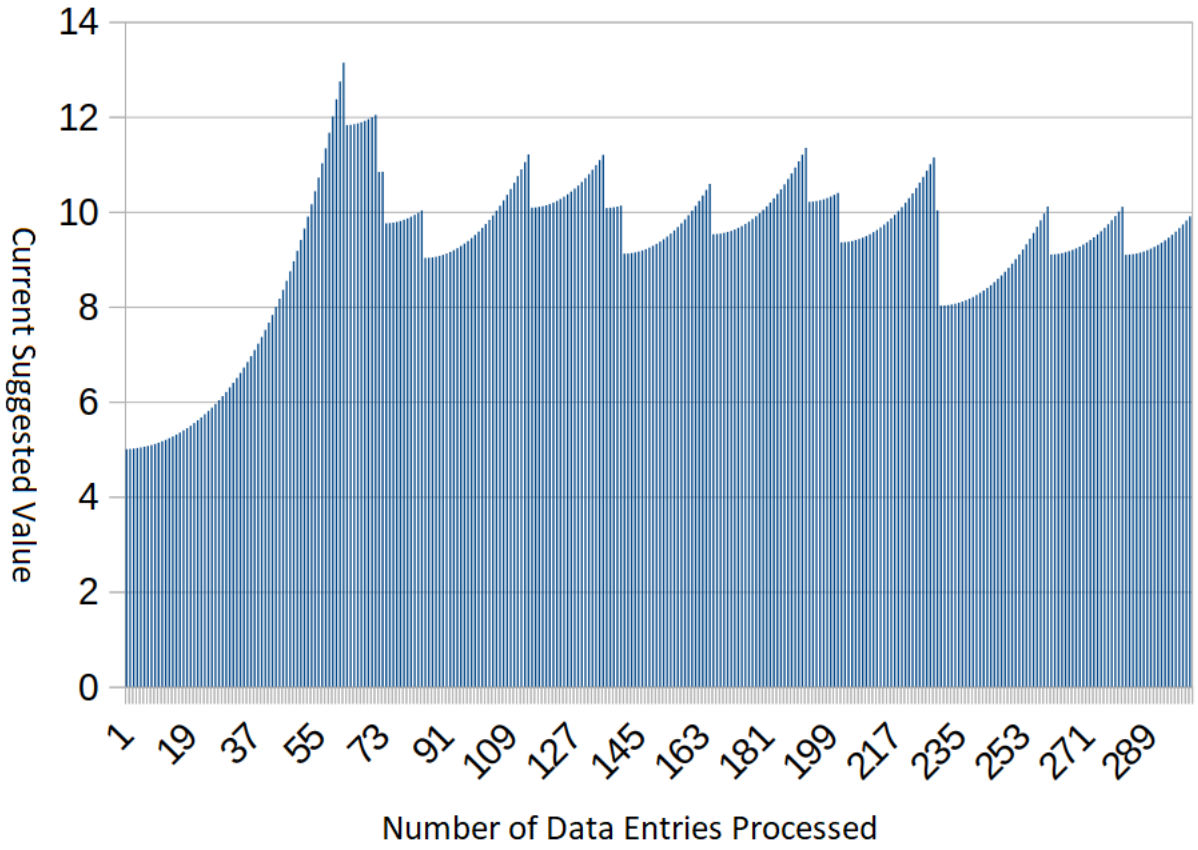

Figure 3.4: Demonstration of Rad-Ap modifying a radius across a series of responses

ciated with a particular density as it receives a series of successes and failures. For this demonstration, I had the algorithm have a chance of failing starting at 10, then had that chance increase until it was guaranteed to fail beyond 15. As seen, the algorithm grows exponentially towards the value, backs off a reasonable distance so that it will be underneath the value where it encountered the failure, and grows slowly back towards it. It can be seen how this system can work for very high levels of precision here as well, as it increases in very small increments after encountering a failure at the fairly low value of 10.

In Figure 3.5, another demonstration of how the radius increases as it receives reported successes and failures is presented, but in this case, the example focusing on another aspect of Rad-Ap. I removed the randomness in where the algorithm would fail, making it so that it would fail if and only if the suggested radius was beyond 10. By doing this, it can

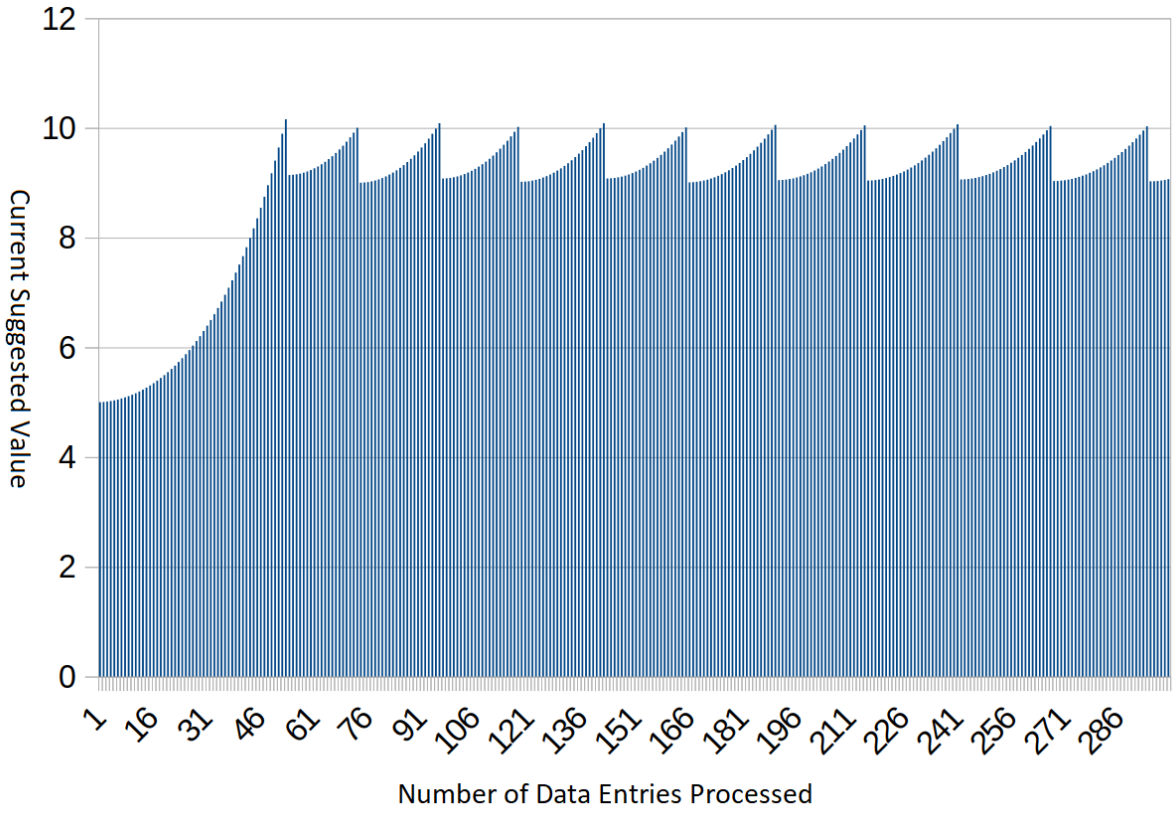

Figure 3.5: Demonstration of Rad-Ap modifying precision across a series of responses

be seen the goal value start to come into effect. If you look closely, you will notice that the distance between the spikes is slowly increasing over time, meaning that the number of successes between each failure is slowly increasing. This is due to the algorithm noticing the disparity between the current success rate and the desired success rate indicated by the goal. When it notices this, it begins to increase more slowly, increasing the precision of the algorithm and allowing the success rate to rise closer to the desired value.

3.7 Data Saving

In order to facilitate the process of cluster head handoff, Rad-Ap includes a data saving process that saves both the currently stored values for the intersection and its internal parameters. Once this data has been saved, a new instance of the algorithm can be started in the same state as the previous one by loading this data. This can be used both

for newly elected cluster heads when the previous one was retired or leaves as well as for nodes that are not affiliated with a cluster but want to start a new cluster of their own. This data will only be fully accurate when used in clusters at or formed in the intersection it was operating within, but can also be migrated to nearby intersections that have not yet collected any information to assist them in the start up process. This will help them train as nearby intersections are likely to follow similar traffic patterns. Even if the nearby intersection has very different traffic, starting with the state from a nearby intersection could still speed up the training process when compared to starting completely untrained, unless near-optimal initial parameters for the system have already been discovered. The save file is also currently human readable, so that near optimal initial parameters can be discovered by reading and processing the save file data. Once such efforts have been concluded, I plan on adjusting the save file system to store the data more efficiently to make passing it across the network easier. In order to ensure that the save file stays within the cluster, the cluster head will occasionally broadcast it to all nodes as a data packet, and will pass it less frequently when the node is stationary at the intersection. Since losing some of the most recent data is not a huge setback to the network, this packet is passed significantly less frequently than a beacon. The packet will also be passed if the cluster head is still in the cluster when a node declares that it will be the next cluster head.

3.8 Recovery System

One potential problem for Rad-Ap, as it has been discussed it so far, is that if the goal value is set to be higher than the system can feasibly achieve, it can cause the performance of the system to degrade considerably. While it would be simple to say that it is on the user of the system to configure it with the correct parameters, I wanted to add a recovery system. That way, if the system detected that its initial parameters were significantly off, it could adjust them to be closer to the optimal value. As a result, I implemented a system where if the system had adjusted too far to account for its inability to reach the goal,

it would detect this and lower the goal slightly. If this new goal was still too high, the system would continue to adjust until a value that worked was found. As a result, this is not guaranteed to find the optimal goal for the environment the system is running in, but it will find a goal that works if its initial goal is not feasible. As such, passing in an accurate goal parameter is still preferable, but failure to do so will not cause the system to be completely incapable of performing anywhere near optimally.

3.9 Additive Increase Mode

One limitation of my system as described up to this point was that it was always operating in a multiplicative increase mode. This is a shortcoming in that for the same success rate, a multiplicative increase will have a lower average radius. This is because the multiplicative increase will form an exponential curve when the values it tries are graphed, while an additive one will form a linear progression. As Figure 3.6 shows, the area under an exponential curve will be less than a linear curve when they start at and progress to the same value.

This smaller area under the curve means that my algorithm will accept fewer potential transmissions, which reduces the total throughput of my system. In order to reclaim this set of transmissions, I added the ability for my algorithm to switch to an additive mode of operation.

My algorithm will still start in multiplicative increase mode, as normal. This allows it to quickly adjust to the optimal value. Once it has reached the optimal area and collects a significant number of data point indicating that the networks is stably operating in this area, it will switch to additive mode. When in additive mode, after the radius is decreased it will set a value equal to the amount that the radius decreased divided by the value of *Goal*. This value will be the amount that the radius will increase by for each successful step until it decreases again. This number was chosen since *Goal* is the number of successful messages the user wants the program to achieve in a row. Since the algorithm

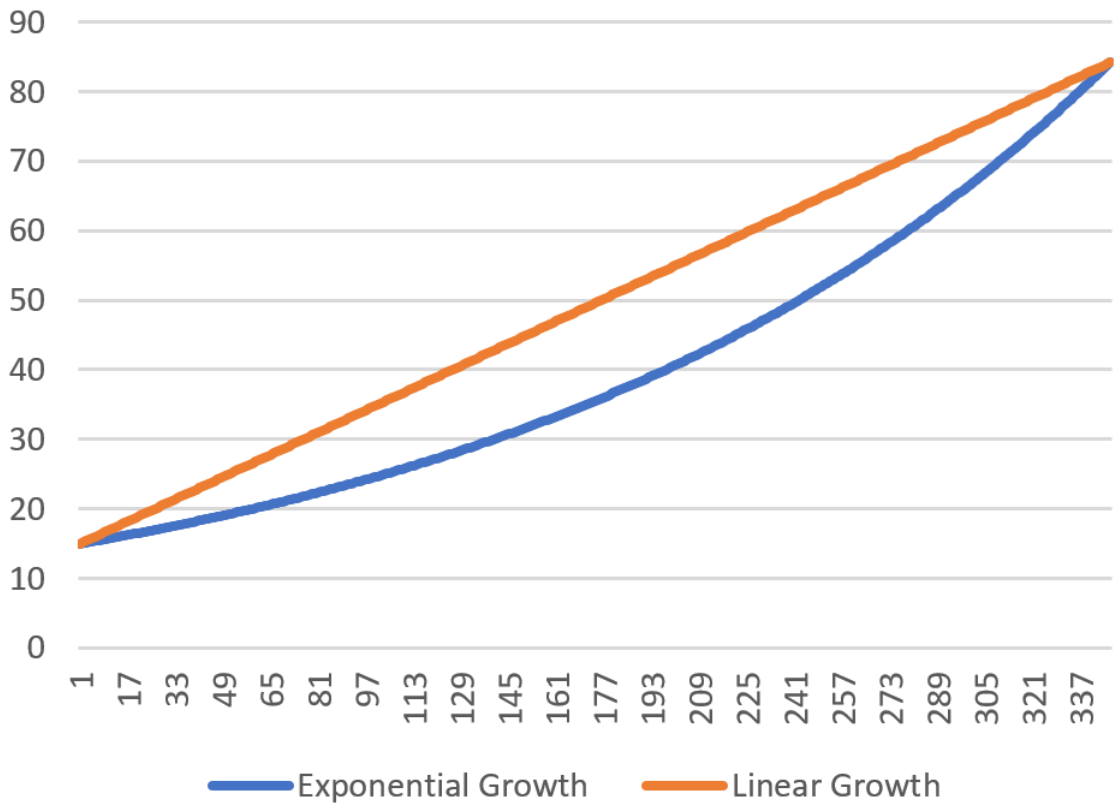

Figure 3.6: A comparison of the growths of linear and exponential curves

increases by the amount it decreased divided by *Goal*, it is expected that it will be able to step upwards *Goal* times before reaching the same value and failing there again. in practice the variance means that this is not exactly true, but in the average case the algorithm will manage to have *Goal* successes between each failure in this mode of operation, exactly as desired.

3.10 Securing Rad-Ap

As mentioned previously, one of my overarching goals in designing this algorithm was to ensure that it was sufficiently secure against attacks. When investigating this, there were two primary possible results of an attack that were identified: attacks that seek to gain the information contained within the algorithm and attacks that seek to cause the al-

gorithm to misbehave or crash. I will now discuss how the algorithm defends against these different types of attacks.

When it comes to attacks to gain the information contained within the algorithm, I could not establish a defense that stopped nodes from being able to access the information. The cluster head would, by necessity, need access to the information to perform the computations, and there is no reason to assume that the cluster head is not a malicious entity. As a result, instead of focusing on how to prevent nodes from accessing this data, I focused on ensuring that this data contained nothing that an attacker could use against the target maliciously. As a result, I sought to ensure that no information that could be used to identify a node was stored past a transient period where the node ID was used to prevent duplicate entries. As a result, the only information the algorithm stores on nodes is their timestamp of entry into the intersection. Even then, the algorithm does not store it for a particularly long period of time. This information could easily be collected by any node listening in promiscuous mode at the intersection, so the algorithm does not put the user at a notably increased amount of risk in this regard.

Attackers attempting to cause the system to perform incorrectly or completely crash was a concern. I took a number of precautions to help limit the risk of this occurring. The first precaution was that timestamps will be collected by the cluster head using its internal clock. This will prevent users from creating bogus timestamps to send to the cluster head, causing it to report absurd sequences of vehicle arrivals. It also will help reduce any inconsistencies caused by differences in clocks between vehicles. However, as mentioned previously, there is no guarantee in this system that the cluster head is not a malicious agent. As a result, the algorithm could not assume that the data it received as input would always be correctly formatted. In order to alleviate these concerns, I added a number of input validation systems, as well as including a maximum input size to prevent any buffer overflow attacks.

The next point of concern was if the cluster head itself reported bogus timestamps. As

a result, I implemented a number of preventative measures to help reduce the effects this could have on Rad-Ap. First, the algorithm ignored any timestamps less than the previous timestamp reported as timestamps should only be reported in temporal order. Any timestamps out of order are either due to data error from differing clocks between different cluster heads or a sign of a malicious attack. In either case, these timestamps should be discarded. Secondly, if any timestamps were significantly later than they should be, they will have their consideration deferred. Timestamps are considered to be significantly later if they are beyond a certain percentage of the time that the next stamp would be expected to come in based on the current density for that intersection. Basing it on the current density helps prevent any issues that would arise from using the same constant value for an intersection at rush hour and an intersection at two in the morning, as they would have very different expected times for the next reported timestamp. Consideration for these is only deferred instead of ignored since these could be the result of statistical variance or sudden changes in road conditions due to accidents or construction. When consideration is deferred, the values are stored in a separate queue. If any values less than the deferred values but greater than the previous value are reported, the deferred queue is discarded and its contents are ignored. If the deferred queue grows to be large enough, the contents are added to the actual algorithm and used to calculate the current density as normal.

3.11 Current Limitations and Shortcomings

By using these methods, Rad-Ap can mitigate some of the impacts of malicious agents or faulty data on the performance of the algorithm. However, there is still some room for improvements in the security of the algorithm. Currently, there is no accountability built into the cluster head, so if the cluster head advertises having certain services but does not actually provide any of the services, the algorithm has no methods for detecting or preventing this. As a result, the current state of the algorithm relies on the network being composed primarily of cooperative members in order for it to function fully. Future re-

search will look at some of the methods to help improve cluster head accountability and prevent malicious agents from having as much impact on the system.

One limitation of the system is that it is built on an active machine learning algorithm. While this enables the system to be more flexible and adaptive, it does mean that it will necessarily consume more resources than on a system that had completed training an offline algorithm and only needs to use the generated result. However, since the data necessary to train such an algorithm is not yet available, the choice of an active algorithm is reasonable for such a system.

CHAPTER 4

RESULTS

4.1 Tests Run

In this section, this dissertation will investigate how effectively my cluster selection algorithm can identify the correct vehicles to include and exclude from the network in a variety of different test cases. For these tests, it will be assumed that a single factor is being passed across the network that indicates a node's fitness. In practice, Rad-Ap will have more than one factor influencing it, but these factors will be reduced to a single value, thus the way in which the algorithm functions will not be significantly different. In this case, I model this single factor as the distance between the node and the cluster head. Nodes outside of this range are considered to be told to backoff by the system, while those within the range are part of the cluster. Since I am modeling this threshold as a range, I will be referring to it as a radius for communication from here onwards.

For the first round of tests, I conducted studies on the original version of Rad-Ap. This version of the algorithm lacks the ability to slowly increase the precision of the algorithm over time. That means that it has a more limited potential to increase its training as the amount of data for it to train on increases than the full version of the application that this dissertation discussed previously. Additionally, it cannot transition into an additive increase mode, and only has the multiplicative increase scheme. However, the core Multiplicative Increase, Multiplicative Increase algorithm and divisions of time periods still functions the same as the new algorithm.

To add validity to the tests, I also compared my algorithm to two other algorithms. When selecting these algorithms, I first had to make sure that the algorithms could be trained online. An algorithm that runs offline would not be able to reactively adapt to new data values, and should only be considered once a large amount of data has been considered. I then selected two different type of algorithms: an algorithm lighter-weight than my own, and an algorithm that has greater demands on the environment that it is run in than my own. If my algorithm runs worse than the lightweight algorithm, then that means there is no need for my algorithm, as there exists an algorithm that is both faster and better at solving this problem than my own. If my algorithm performs worse than the heavyweight algorithm, then it might be worth considering whether the difference in running time is significant enough to not simply run the more effective algorithm.

For the lightweight algorithm, I selected a naive version of a Additive Increase, Multiplicative Decrease algorithm. For the sake of fairness, I made sure that the rate of increase of the Additive Increase, Multiplicative Decrease algorithm was reasonable when compared to the optimal value. I did not want it to experience any of the aforementioned problems where it was either unable to hone in on the true value in a reasonable timeframe or unable to hone in on it at all due to its increase value being too coarse of a scale for it to reach the ideal. For the Additive Increase, Multiplicative Decrease algorithm to compare against, I chose to use Transmission Control Protocol's (TCP) congestion control scheme. I chose TCP congestion control as it is a commonly used Additive Increase, Multiplicative Decrease algorithm that is also used in the field of networking to adjust to channel capacity, albeit for a different purpose.

For the heavyweight algorithm, I selected Vowpal Wabbit's (VW) linear regression algorithm. This algorithm was selected as it is a commonly used implementation in the field, and supports online learning [73] [74] [101]. To give this algorithm further advantages, I also pre-trained the algorithm on over 8 million data points tested by my Rad-Ap algorithm, as well as their solutions, which are a ground truth in this environment. The model

was then trained on these data points, and its "suggestion" was considered to be the point where its confidence in the result being a success decreased below 96%. This value was chosen after testing showed that higher thresholds would result in more noise values being picked up, causing the model to over-fit the data, while lower values afforded much smaller increases in the average radius. The model was tested on a reduced subset of the data that did not include any of the data points used in training the model. As a result, its optimum values will be slightly different from other algorithms.

In order to run these tests, I collaborated with the Alabama Department of Transportation and used a subset of their collected vehicle traffic data to set up the test corpus. I selected 54 different intersections, some of which were near each other and others of which were further away. Eighty-four weeks worth of vehicle incident data was gathered from these intersections. Vehicle incident data records when a vehicle enters an intersection as well as which detection device they triggered. This data can be used to determine which lane they were in. After I collected this data, I had over 540,000,000 records of vehicles entering the intersections that I could use to model vehicle traffic patterns and test the algorithm.

A simple test program was set up where it would take the reported data on intersection entry events from the Alabama Department of Transportation, and model each one as the timestamp that a vehicle entered the range of the cluster, allowing the programs to update their current density information. In the case of the naive Additive Increase, Multiplicative Decrease algorithm, this information was ignored. The test program would use the current density of the intersection to determine a maximum value for the range of the cluster to be extended. This value would then be adjusted slightly such that it could be within a range of values instead of always being the same value for a particular density. I call this range modification a "jitter" value, as it represents how network conditions are not fully consistent and may not always fail at the same point. If the algorithm reported back a suggested radius less than this value, the test program would report that

the communications had occurred successfully, and the algorithm would adjust the radius for future occurrences based on this feedback. If the radius exceeded this value, the test would say that some sort of network interference occurred or some of the nodes were out of transmission range, causing the transmission to result in a failure. The algorithm would similarly adjust in this case as well.

4.1.1 Natural Tests

I set up a number of different testing scenarios in order to see under what conditions Rad-Ap would performed well, and under which ones Rad-Ap would struggle or be completely unable to achieve the desired result. The first set of scenarios was designed to test the effect that the starting value would have on the performance of the algorithm. To accomplish this, I created three test cases, referred to as Natural Tests 1, 2, and 3. All of these test cases were designed to be a fairly reasonable approximation of a real network scenario and have the following properties: cluster radii start at the value of 80, a purely arbitrary number, and increase as the density increases proportionally to the square root of the density. Square root scaling was chosen as the metric to represent the distance from the cluster head; therefore, the total number of vehicles that can be included within a radius will be increased proportionally to the square of the radius. Conversely, the radius will increase proportionally to the square root of the number of vehicles in the cluster. The jitter for this was then set to provide an approximate variance in the cutoff value of plus or minus 5.4 percent. This gives a fairly unstable cutoff value, with the total range of possible values having just under 11 percent variance. While it would be more realistic to use a bell curve distribution for these random values, I used an even distribution as it would make the network slightly more unstable than a real world environment. This makes it so the test case is expected to be more difficult for the algorithm to run in than it would be in the real world. If it can perform well in this test case, it can also perform well in the real world. If I had instead used a bell curve that was more stable than the bell curve of the real world, I could become convinced that the algorithm was capable of func-

tioning when it in fact could not, which would be greatly problematic.

For Natural Test 1, I start the initial value at 5, which is less than the optimal value by a reasonable amount. This is intended to model the case where the algorithm is deployed completely untrained, as it would be wise to start with a fairly conservative initial value and allow the algorithm to grow to where it should be. Natural Test 2, on the other hand, was designed to model the case where the algorithm was deployed after some information on the typical value of the radius had been collected. As such, Natural Test 2 starts with an initial value of 80, making it start just slightly under the cutoff value for most densities. Natural Test 3 is made to look at the scenario where the initial value is chosen very poorly. As a result, the initial value for Natural Test 3 is 9,000,000, more than 100,000 times what the cutoff value will be. This test was made as a stress test to ensure that there was no initial value chosen that would render the algorithm completely unable to run. It was important for my previously stated goal of making the algorithm unit agnostic. If this algorithm were to be run by a system that used data collected on an average cluster size that was using an algorithm with a very different unit scheme on accident, it is possible that the initial value could end up extremely far from the true value. This forms the primary motivation behind my inclusion of extreme test case in this battery of tests.

4.1.2 Network Condition Tests

The next set of tests created were designed to test how Rad-Ap could perform on a number of different network conditions. Since the first test battery only looked at different initial configurations of the algorithm, I created these tests that would instead investigate how effectively the algorithm could perform in network scenarios besides the one I initially envisioned. This is important to test since network conditions can be very different depending on the conditions of the area the network is in, the nodes on the network, or even the packets sent across the network. As a result, even if the first set of tests were a perfect envisioning of the average network conditions, only passing those tests would not necessarily give any indication of how well the algorithm would perform in many of the scenarios

in which it could be operating. These reasons are why I created this second battery of test where the network conditions are simulated to be more extreme or even less extreme than my initial presumptions.

This battery is formed by four tests, with comparisons to Natural Test 1 as a default network test case for the other tests to compare against. First, I created a test case, which will be referred to as Network Test 1, where I removed the jitter value that I used to modulate the cutoff threshold. With this value removed, the network will be much more stable, so ideally the program would be able to perform notably better on this more stable network. Network Test 2 was then designed to model the opposite scenario - one where the network was significantly less stable than the previous tests. To model this, I tripled the jitter range so that it could now be plus or minus 16.2 percent of the cutoff value, approximately. Now with this extended range the cutoff value has a total possible range of over 30 percent of its average value. I expect that the increased range of the random jitter will make it more difficult for the algorithm to hone in on an optimal value as it could reach values anywhere in that range before suddenly getting cut off.

I then created Network Test 3, a test where the jitter value is removed and instead the transmission has a five percent chance of failing at any value that is less than the cutoff, with values greater than the cutoff still being reported as failures. This is intended to model how networks can encounter many scenarios where packets will randomly fail to send, including packet collisions, data corruption, or just lost connections. Since there is no guarantee that every communication within the transmission range will be able to successfully complete, I added this variable to model that concept of failures unrelated to the suggested radius. The value of five percent was selected because all versions of my algorithm were looking to attain a success rate of at least 95%. By adding a five percent failure rate, we make this impossible to achieve, forcing my algorithm to adapt. The last test in the network condition test battery is a test that combined the tripled random jitter of Network Test 2 with the five percent random failure rate on all transmissions of Network

Test 3. This creates a test scenario with an extremely unstable simulated network, allowing me to test whether the algorithm is capable of running even in conditions where the network variance is high and the successful transmission rate is low.

4.1.3 Scaling Rate Tests

Finally, I created a battery of scaling rate tests. In the initial description of the Natural Tests, I discussed how I chose to have the threshold values scale with the square root of the density since in the model where the threshold values represent a radius, it made the most sense. However, since this may be run on other thresholding systems, they may use other metrics like number of links or connection quality. In these cases, the threshold value may not necessarily scale with the square root of the density anymore. As a result, I created these scaling rate tests to test how well Rad-Ap would perform if the threshold no longer scaled with the square root of the density, but instead some other rates.

In Scaling Test 1, I set up the test case so that the threshold value would not scale to the density at all. This could be the case if the threshold value algorithm were a very basic implementation that gives only a single cutoff point regardless of the current network conditions. While it is obviously far from the ideal type of thresholding system for the network to use, it is possible that it could be run in such conditions. For Scaling Test 2, I modify it so that the threshold scales linearly to the density. This scaling could be seen in threshold systems based on the connection quality and/or the number of connections nodes have. In such a system, increasing the number of nodes in the network would increase the threshold value for each node in the network proportionately. Since the maximum number of nodes in an optimum cluster would remain constant, the value at which vehicles must be excluded would grow with the average threshold value. Since these metrics grow proportionately to the number of nodes in the network, so too would the cutoff value. However, there is a chance that values like this could fall in such a way that they would instead scale proportionally to the density of the intersection squared. As a result, I added Scaling Test 3, a test which replaces the square root of density scaling with density

Test Name	Class's Initial Value	Random Jitter	Failure Chance	Scaling Rate
Natural Test 1	5	5.4%	0%	$\sqrt{n}$
Natural Test 2	80	5.4%	0%	$\sqrt{n}$
Natural Test 3	9,000,000	5.4%	0%	$\sqrt{n}$
Network Test 1	5	0%	0%	$\sqrt{n}$
Network Test 2	5	16.2%	0%	$\sqrt{n}$
Network Test 3	5	0%	5%	$\sqrt{n}$
Network Test 4	5	16.2%	5%	$\sqrt{n}$
Scaling Test 1	5	5.4%	0%	0
Scaling Test 2	5	5.4%	0%	n
Scaling Test 3	5	5.4%	0%	n^2

Table 4.1: Experimental Tests Conducted

squared scaling.

With the various test cases established, I was able to begin the testing process. Each test was run five times, with the final result being the average of all of the tests run. During this process I found that variance across runs was very low, so it seems likely that the average values are indicative of the true values that this algorithm will produce in the average case on these tests. For the initial version of this algorithm, the primary goal in these test cases was to achieve a success rate of at least 95 percent. In the case of the tests with an inherent five percent random failure rate, I lowered this goal as it would require the algorithm to result in no failures due to its growth. Rather than this being a sign of success, be a sign of its failure to work in this environment, as it would mean that it was incapable of ever getting up to the cutoff value. The secondary goal was to get the average distance of the optimal value as close to the optimal value as possible. Ideally, I would like this to be within seven percent for all of the test cases.

4.1.4 ns-3 testing

An additional set of tests were run using ns-3. ns-3 is a network simulation software that enables us to have a more precise simulation of vehicles and their mobility patterns, as well as how these affect their network connection. I simulated a 4-way intersection that had approximately 57,000 vehicles enter the intersection over 100 hours. The intersection

Test Name	Algorithm Used	Average Threshold Distance	Optimum Threshold Distance	Success Rate
Natural Test 1	RadAp	82.6147 ft	86.9851 ft	95.0813%
Natural Test 2	RadAp	82.6241 ft	86.9851 ft	95.0812%
Natural Test 3	RadAp	85.3820 ft	86.9851 ft	95.0799%
Natural Tests	TCP CC	62.3182 ft	86.9851 ft	98.0626%
Natural Tests	VW	69.2725 ft	86.9541 ft	96.0094%

Table 4.2: Natural Test Results

was simulated to have a traffic light, so depending on the entry and exit road of the vehicle they may be forced to stop at the light for some period of time. The intersection was simulated such that the connecting roads continued outward for a considerable distance, around 700. I gave each vehicle 10 packets that were one kilobyte in size each that they would attempt to send as they entered the intersection. The vehicles were each installed with a device capable of sending and receiving packets using 802.11p's network protocol. All vehicles sent their transmissions over a single data channel. Meta information for RadAp was not sent over the data channel. This is typical for a real world implementation, as 802.11p sets aside a separate channel for communications about the network. RadAp v3 was then used to moderate the vehicles that were allowed to send packets, as it would in a real-world application. I measured the number of packets sent successfully, the number of packets that were not sent successfully, and the number of packets that my algorithm prevented vehicles from sending. I ran this test a total of 10 times and averaged the results.

4.2 Results

4.2.1 Natural Test Results

The first tests I conducted were the natural tests, the results of which can be found in Table 4.2. In the table, it can be seen that in Natural Test 1 Rad-Ap achieved an average suggested radius of 82.6147 feet, while the optimum cutoff distance was at 86.9851

feet, putting the average suggested radius within 5.0243% of the optimum value. Additionally, it achieved a success rate of 95.0813%, just over the goal value of 95%. As this is the test that is intended to most closely resemble the expected conditions of real world scenarios, this test should be the most indicative of how the class would be expected to perform if deployed untrained in a VANET system. Thus, it would appear that the class will be capable of maintaining a high success rate while managing to keep its suggested radius reasonably close to the optimum value, just under where communications will be cutoff in such an environment.

Natural Test 2 saw slightly better performance on average, which matches my expectations for this test, as it represents how the class will perform once data on the intersection has been gathered and used to adjust future versions of the class. However, the difference was surprisingly small, with the average threshold distance being only 0.0094 feet higher than the first test case. Furthermore, the success rate was actually 0.0001% lower, though this difference could have been caused by slight statistical variance. This makes sense when considering how both versions of the algorithm would have run. The untrained variant would start at a lower value, which would bring down its average, but the time it spent within this lower range would have a much higher success rate, increasing the average success rate. Thus, it is reasonable that starting at a higher value increases the average radius while slightly decreasing the success rate. It can also be seen how both versions had similar performances in the long term, as well as how only a small portion of the time spent running the test values was actually spent reaching the optimum value. It spent most of its time performing similarly to how the pretrained version would perform. However, pretraining would seem to still cause the algorithm to perform better for the initial set of values.

The third test case (Natural Test 3) produced some interesting results. In this test, it can be seen that the success rate was slightly lower than the previous tests, with a total difference in success rates between it and Natural Test 1 of 0.0014%. However, the average

radius was markedly closer to the optimal value than any of the previous tests, with a difference of only 1.8430%. Rather than this being indicative of such a starting value being highly suited to the performance of the algorithm, I believe this to be the high initial values that the algorithm will start at for each density throwing off the average value. This would be reasonable because they are over 100,000 times the average value in the result, so a few values in this range as each density steps down from them could cause the average to be dragged upward by a few points. Even so, it can be seen that the algorithm is still perfectly capable of reaching a reasonable average radius and success rate even when starting on a far off value. Thus it would appear that the algorithm is capable of performing reasonably despite the initial value chosen, though it performs better in the initial stages of the algorithm if it starts nearer to the goal value.

When comparing these results to the Additive Increase, Multiplicative Decrease algorithm, it can be seen that the success rates are much higher for the Additive Increase, Multiplicative Decrease algorithm. However, this has a serious trade-off in terms of average radius. While I do consider success rate to be more important than average radius, going from being within 5.0243% of the optimum radius to being only within 28.3576% is a difficult difference to ignore. This difference is even larger than it may seem in this case, since we assume in these tests that increasing the radius will increase the number of nodes proportionally to the radius squared. As such, in Natural Test 1 Rad-Ap seems to be excluding around 9.7962% of potential transmissions, whilst TCP congestion control excludes approximately 53.9596% of potential transmissions. As a result, it would seem that for most systems the algorithm would be preferable, however, there might be environments that prefer the higher success rate of TCP's congestion control algorithm. Note that I only have one test case for TCP's congestion control algorithm here, as TCP is not intended to be started with different initial values. Even if it were, it would very quickly run identically to how it does with no initial value. As such, I only included a single test for TCP's congestion control algorithm here. In future test batteries where the environment is being

changed, I will include a test of TCP's congestion control algorithm for each case.

Variance across multiple runs of these test cases was incredibly low, with all runs having an average range of less than 0.0248% in the suggested threshold distance, and a range of less than 0.00126% in the success rate. As such, differences between the tests are very likely to be indicative of an actual difference in performance between the tests rather than the result of statistical variance.

In the case of Network Test 1, the standard deviation was 0.006959 feet for the average radius, and 0.0002049% for the success rate. This means that the average radius we saw for TCP congestion control was around 2,917 standard deviations from the mean. Testing for non-equivalence at a significance level, or alpha, of 0.01, I obtained a p-value of less than 0.000000001, meaning the difference is considered to be significant at an alpha of 0.01. The success rates were closer to each other, but due to the smaller standard deviation were statistically further, with TCP congestion control being 14,550 standard deviations from the mean. As a result, the p-value was even lower in this case, meaning the test was once again significant for non-equivalence at an alpha of 0.01.

I once again elected to have only one case for Vowpal Wabbit, as we are primarily interested in how Rad-Ap performs with different initial value for these test cases. As a result, for the Natural Tests we see that Vowpal Rabbit achieves a much greater proximity to the optimum radius than TCP congestion control did, being within 20.3344% of the optimum radius. However, it is still far from the proximity to the average radius that Rad-Ap achieved. Like TCP congestion control, it had a greater success rate than Rad-Ap, but was much lower than TCP congestion control's at only 96.0094. As a result, it appears that Rad-Ap still has an advantage over this algorithm, but which is preferred may be based on the environment.

4.2.2 Network Condition Test Results

The next battery of tests focused on how network performance would impact Rad-Ap, using the same goals as the first three tests. Due to changes in the test cases, the average

Test Name	Algorithm Used	Average Threshold Distance	Optimum Threshold Distance	Success Rate
Network Test 1	RadAp	84.7879 ft	91.9851 ft	95.1379%
Network Test 1	TCP CC	68.1645 ft	91.9851 ft	97.8747%
Network Test 1	VW	86.2005 ft	91.9541 ft	95.8563%
Network Test 2	RadAp	75.2294 ft	76.9851 ft	94.8653%
Network Test 2	TCP CC	57.2289 ft	76.9851 ft	97.8915%
Network Test 2	VW	65.1075 ft	76.9541 ft	95.2500%
Network Test 3	RadAp	57.3099 ft	91.9851 ft	94.0315%
Network Test 3	TCP CC	29.1211 ft	91.9851 ft	94.8940%
Network Test 3	VW	78.3317 ft	91.9541 ft	91.8750%
Network Test 4	RadAp	52.0310 ft	76.9851 ft	93.9759%
Network Test 4	TCP CC	28.2624 ft	76.9851 ft	94.8287%
Network Test 4	VW	62.0742 ft	76.9541 ft	91.1063%

Table 4.3: Network Condition Test Results

cutoff value and thus the optimal threshold distance changes slightly from test to test. As mentioned previously, it would not be possible for the class to reach a 95% success rate on the tests that had a 5% random failure rate unless they had a 100% success rate on all the transmissions that did not randomly fail; therefore, those tests were not expected to reach that goal. Results of the network tests can be found in Table 4.3.

The first of the network tests (Network Test 1) completely removed the random jitter found in Natural Test 1, and achieved an average suggested threshold distance of 84.7879 feet. Since this test removed the random jitter, the true optimum value was further away than on previous tests, at 91.9851 feet, meaning that the algorithm only got within 7.8243% of the optimum value. This puts it slightly outside of my previously discussed goal of a threshold within 7% of the optimal value for all of the test cases. The success rate was also higher than the previous tests, at 95.1379%. However, this is somewhat less impressive than it was for previous test, as there was no randomness involved to decrease the success rate. While the performance of this algorithm could yet be improved by passing in better variables, I noted this as a failure of my initial algorithm. The results of this test and a couple of the tests in the next section are what led to the introduction of the goal

value that allows the program to slowly increase its precision over time.

When comparing against TCP congestion control, it can be seen that TCP congestion control has gotten closer to the optimum value in this test case by a larger amount than Rad-Ap did, with an increase of 9.3813%, while Rad-Ap only had an increase of 2.4209% from its radius in Natural Test 1. This puts its average radius within 25.8962% of the optimal radius, a difference of 2.4614% from its performance in Natural Test 1. In exchange, the success rate fell, but only a very small amount, around 0.1711%. As a result, it can be seen that TCP congestion control was better at adjusting to the more stable environment than Rad-Ap. However, the distance between TCP congestion control's average radius and Rad-Ap's average radius is still quite stark.

Vowpal Wabbit directly outperformed Rad-Ap in this test case, having both a closer proximity to the optimum value, being within 6.2570% of it, as well as a higher success rate. It appears that removing the random jitter made it much easier for Vowpal Wabbit to find the dividing line between the successes and the failures. As such, it performed extremely well in this test case.

Network Test 2, which increased the random jitter to have triple the range of the original test, had the least distance between the optimum threshold distance and the average suggested threshold distance, with the average value at 75.2294 feet and the optimum value at 76.9851 feet. However, this did decrease the success rate to less than my goal of 95%, at 94.8653%. It can be seen from the tests run thus far that the algorithm has a trade-off between success rate and proximity to the optimum distance. Where this exchange is centered can be affected by changing the values of the class, but these tests give a reasonable idea of the sort of relation between the values one can expect to see when the class is used in a real world environment. Hopefully these tests can inform future decisions of class variables so that they will match the needs of the particular network environment they desire.

In this test case, TCP congestion control had a larger decrease in average radius than

Rad-Ap did decreasing by 16.0429% from the previous test's average radius, while Rad-Ap only decreased by 11.2734%. Since the optimum radius also decreased from the previous test case, this puts TCP congestion control within 25.6624% of the optimum radius, its best ratio yet. Additionally, its success rate also increased from its rate in Network Test 1 by 0.0168%. This is particularly interesting when comparing this result to the algorithm's result for Network Test 1, as Rad-Ap instead saw a decrease in success rates. This makes TCP congestion control a bit more stable in this circumstance; however, since Rad-Ap had its best average radius so far in this test case, its difficult to say that its lack of stability hurt it significantly in this case. However, the fact that there is some instability here in Rad-Ap may be indicative that there are test cases in a similar vein that will cause Rad-Ap to suffer a much decreased performance. As a result, it may be interesting to investigate this in the future.

In this test case, Vowpal Wabbit's performance was more in line with what we saw in the natural tests. It actually got slightly closer to the optimum radius than it did in the Natural Tests, getting within 15.3944% of the optimum value, but in exchange its success rate fell, reaching only 95.2500%, putting it only slightly higher than Rad-Ap's success rate. As a result, it was nearly directly outperformed by Rad-Ap, having a much lower average radius and only a slightly higher success rate. As a result, we can see that the jitter value has a large impact on how well Vowpal Wabbit can classify the data.

Network Test 3 was the test that replaced random jitter with a 5% failure chance. This test saw the success rate fall to only 94.0315%, meaning that of the transmissions that didn't randomly fail, 98.9805% succeeded. However, the average suggested threshold distance plummeted to 57.3099 feet, while the optimum returned to the value it had in the first network test case, 91.9851 feet. This means that the program only got within 35.8536% of the optimal value, a far cry from its previous performance. This came about since the class was set up to seek a 95% success rate, but was run in a system where a 95% success rate is practically impossible to achieve. This further demonstrates how this

version of the algorithm has a set exchange between proximity to the optimal threshold value and success rate of transmissions. Where exactly the algorithm will fall on this scale is determined by the initial values passed to the class. Since this means certain values being less than optimal for the system the algorithm is employed in would cause the algorithm to function significantly less well than desired, I set about to adjust the second version of this algorithm such that it could detect scenarios where its values were notably subobtimal and attempt to recover from them at least somewhat. This is what led to the recovery system mentioned previously where the algorithm is capable of adjusting its own goal.

TCP congestion control did not handle this test case very well. The tests show that the average radius decreased sharply in this scenario, down to being only within 68.3415% of the optimal radius. This indicates that the random failures started being the dominating item for TCP congestion control rather than the point at which failures start to occur from network overload. As a result, it can be seen that Rad-Ap was more resilient in this test case than TCP congestion control was by a significant margin. However, as mentioned previously, I still seek to improve the algorithm for these sort of situations.

Vowpal Wabbit performed well in this test case. It didn't directly outperform Rad-Ap due to its lower success rate, but it had much better proximity to the optimum radius, being within 14.8143% of the optimum value. Its success rate was lower, but if we account for the 5% of messages that were randomly failed, it achieved a success rate of 96.7105% on the remaining messages. This success rate is still within our goal of a 95% success rate, so considering only how well the algorithm performs and not the cost of running the algorithm, we would prefer Vowpal Wabbit over Rad-Ap and TCP congestion control for this environment. It makes sense that VW performed well in this environment, as the algorithm is prepared to screen out noise values, like the random failure rate we introduced. Additionally, this test has no jitter value, which was the primary feature giving VW difficulty in previous test cases.

The fourth test case (Network Test 4), the test case that combined the increased jitter of the second and the failure rate of the third network tests, saw a mix of results. As in the case of the second network test, the average suggested threshold distance grew closer to the optimum value than it was in the third, with the average decreasing only to 52.0310 feet, while the optimum decreased to 76.9851 feet. This is still far from ideal, however, as this only brought the average value within 32.4142% of the optimal threshold distance, which is not a huge increase. Also, similar to the difference between the second and first network tests, the success rate fell only slightly to 93.9759%. As such, it can be seen that the algorithm is quite resilient to increases in the amount of random jitter, especially when compared to how much it suffered from having a high simulated packet drop chance in the third test.

Once again, TCP congestion control did not do a particularly good job of adapting to the increase in random failure rate. The increased jitter also made its success rate and average radius fall further, though admittedly not as far as Rad-Ap's average radius and success rate. This smaller decrease from the change in cutoff values provides further evidence to support my hypothesis that random failures are dictating the TCP congestion control radius in this scenario. If the cutoff value were still the primary decider for TCP congestion control, a decrease in average radius proportional to the change in the optimal cutoff value should have been seen. Since it was not, this indicates that TCP congestion control is only rarely reaching the point where its failures are caused by reaching the cutoff value. Thus, TCP congestion control must be mostly encountering only random failures, which means it is not growing to reach the cutoff value.

The increase in random jitter decreased Vowpal Wabbit's performance in this test case, lowering it to only be within 19.3361% of the optimum radius. Its success rate also fell slightly, causing 95.9014% of the messages that didn't fail randomly to be transmitted. As a result, this again brings it to the range where which is preferred between it and Rad-Ap will change based on the priorities of the application it is being used for.

Test Name	Algorithm Used	Average Threshold Distance	Optimum Threshold Distance	Success Rate
Scaling Test 1	RadAp	80.7687 ft	85.0000 ft	95.0788%
Scaling Test 1	TCP CC	61.5239 ft	85.0000 ft	98.0325%
Scaling Test 1	VW	67.7948 ft	85.0000 ft	96.0141%
Scaling Test 2	RadAp	86.4508 ft	91.1076 ft	95.0855%
Scaling Test 2	TCP CC	63.9906 ft	91.1076 ft	98.1272%
Scaling Test 2	VW	72.4181 ft	91.0762 ft	96.3102%
Scaling Test 3	RadAp	88.9243 ft	102.113 ft	95.1007%
Scaling Test 3	TCP CC	69.8315 ft	102.113 ft	98.3113%
Scaling Test 3	VW	78.3050 ft	102.062 ft	96.4491%

Table 4.4: Scaling Rate Test Results

4.2.3 Scaling Rate Test Results

The last battery of tests this dissertation will go over are the tests that were designed to determine how different scaling rates could affect the performance of the class. In order to isolate this variable, the only variables altered were the scaling rates of the densities, and otherwise these tests matched the natural tests. Once again, the optimum value for threshold distance varied from test to test, so optimum values were re-evaluated for each test. The results of the Scaling Tests can be found in Table 4.4.

Scaling Test 1, which had no scaling, performed comparably to how the class ran with the square root of density scaling in the three natural tests. It achieved an average threshold distance within 4.9878% of the optimum value, and had a success rate at 95.0788%, which was 0.0025% lower than that achieved in Natural Test 1. This slight increase in average suggested radius in exchange for a slight decrease in success rate is comparable to the other tests run so far. The increase in average value might be slightly better than the decrease in the success rate compared to some of the other exchanges the algorithm has had, which would indicate that it had an easier time running in this environment, but it is difficult to tell conclusively.

TCP congestion control also performed comparably in Scaling Test 1 to how it did in Natural Test 1, with a slight decrease in average radius loosely corresponding to the de-

crease in optimum radius, though slightly closer to it, now within 27.6189% of the optimum value. Its success rate did decrease in correlation to its increase in average radius, but only by 0.0301%.

Vowpal Wabbit performed similarly to how it did in the Natural Tests, with a distance of 20.2414% from the optimum value. This puts it marginally closer to the optimum than it was in the Natural Tests, while also having a marginally higher success rate.

The second test case (Scaling Test 2) had the average suggested distance within 5.1113% of the optimum value, only slightly further from the optimum value than Scaling Test 1 and Natural Test 1. Its success rate also rose slightly to 95.0855%. This difference is attributed primarily to the increased period of time where the suggested distance was lower than the optimum value, as the linear scaling raised the typical cutoff value. Based on these values, it was determined that the class could run comparably with scaling proportional to the density when compared to tests that ran under the assumption of square root of density scaling.

TCP congestion control performed slightly less well in linear scaling, with its distance to the optimum value falling to 29.7637%, the furthest it has been from the optimum value in any test case so far besides Network Test 3 and 4. This is the case because TCP congestion control does not handle different densities differently, so increasing the spread of possible values by increasing the scaling effectively is the same as increasing the jitter to it. As a result, the importance of differentiating the densities, as Rad-Ap does, can be seen. The success rate did increase slightly in this case, but this is again just the result of the algorithm spending more time beneath the optimum value.

Vowpal Wabbit still performed as normal with linear scaling, reaching within 20.4863% of the optimum radius, marginally further than it was in the Natural Tests. Its average radius instead rose slightly in this case. This is reasonable, since the increase in scaling can allow the algorithm to more easily underestimate the radius, increasing the success rate while decreasing the proximity to the optimum.

The last of these tests (Scaling Test 3) had more issues with reaching the optimal value. In this test, the average suggested threshold distance only got within 12.9142% of the optimum value, with the average value at 88.9243 and the optimum at 102.113. It would seem that in a density squared scaling scenario, rare high density values can result in very high goal values, which the algorithm struggles to reach due to the rarity of these values. Because of this, the average cutoff distance increases from these values, while the suggested radii are left behind for a significant portion of the time the algorithm is running. One positive result of this, however, is that the amount of time spent under the optimum value increased, the success rate rose slightly to 95.1007%. Additionally, since our assumptions about how the radius affects the number of cars included has changed, that means that in this case the difference in values only represents 6.6811% of vehicles that enter the intersection being excluded from the cluster.

TCP congestion control also encountered more difficulties with Scaling Test 3, with its average radius also falling even further from the optimal radius, now only within 31.6135% of the optimum value. I suspect that the reason why TCP congestion control struggled more with linear scaling and Rad-Ap struggled more with squared scaling is due to a combination of reasons: First, only maintaining a separate instance for different densities causes TCP congestion control to be less adaptive to the differences caused in optimal radius due to differing density values. Second, as I discussed previously, the squared scaled scheme creates some values that are significantly higher than other values around them in extremely rare instances. When these values occur, they will immediately start raising the average while the algorithm still has to slowly work upwards to determine where the cutoff is. Thus, the algorithm will spend this time in a similar range to where it previously was despite the change in the optimal value. This problem is exacerbated for TCP congestion control, since it doesn't have any way to track density. This causes it to have no way to tell when the optimum value might have suddenly spiked, and as a result cannot approach the optimum radius for these values.

Vowpal Wabbit also got further from the optimum radius in this case, now only being within 23.2770% of the optimum value. It suffered less than Rad-Ap in this case, presumably because it has the ability to guess at values outside of the scope of what it has seen by extending its projection. However, this is not particularly accurate. Basically any machine learning algorithm needs data to learn the patterns, and for the rare high values, there was not a considerable amount of data. In exchange however, the success rate did increase slightly

4.3 Version 2

One initial testing was completed, we used these results to fuel future development on the algorithms. I introduced the ability to increase precision over time to create an algorithm that would be able to achieve high final success rates while not compromising on the initial growth rate of the algorithm. Additionally, measures were added to detect when the current goal was unfit for the network environment the algorithm was running in, allowing it to adjust the goal until it was more reasonable.

I ran tests on this version of the algorithm twice. On the first run, the goal was set similarly to how it had been in the previous test cases, allowing for a more even comparison between the new version and the initial version. On the second run, the goal was set significantly higher, as the new system goal allows for achieving higher success rates without compromising on initial growths. As a result, the higher goal scenario should be the one for which the new algorithm was most fit.

4.3.1 Natural Test Results for Updated Algorithm

In Table 4.5 there are the results of the updated algorithm being run on the same natural tests as the initial version of the algorithm with its two different goal values. The goal number represents the number of transmissions the operator wants the network to be able to transmit in a row, and thus correspond loosely to a success rate. Since the algorithm will attempt to successfully complete 20 transmissions in a row with a goal value of 20, the

success rate should be at least slightly higher than 95%. Due to statistical variance, it will be a bit higher than that in practice. In the results, it can be seen that the goal value of 20 seems to correspond with a success rate of around 96.6%, and a goal value of 50 corresponds with a success rate of around 98.5%.

Comparing these results to those that were performed by the initial version of the algorithm, it can be seen that the threshold distance has decreased by a fairly notable amount, going down to only being within 5.5013% of the optimum value from the 5.0243% of the previous version of the algorithm. However, the success rate increased, now successfully completing an additional 1.5214% of the transmissions. Since I consider success rate to be more important than the distance to the optimal radius, this exchange seems to be a quite favorable difference. However, since the previous version of the algorithm was also shown to be able to exchange between radius and success rate, this mode of operation doesn't show any particularly noteworthy changes from how the previous version of the algorithm performed, though the exchange rate seems to be slightly improved. The instance run with the higher goal set shows its ability to reach even higher success rates, with a success rate at 98.5251%, almost two percent higher than the version run with the goal set to 20. It is worth noting, however, that this two percent difference is representative of a reduction in the number of failed transmissions by 56.8805% from the version run with a goal of 20 and a reduction of 70.2177% from the initial version of the algorithm. This means that the total number of failed transmissions was less than half that of the version run with a goal of 20 and less than a third of the failures experienced by the initial version of the algorithm. While the average radius once again declined for this, the exchange seems to be quite worthwhile for most situations.

Natural Test 1, with the *Goal* of the new algorithm set to 50, also affords the opportunity to directly compare Rad-Ap with TCP congestion control as the success rate of Rad-Ap is now much cluster to that of TCP congestion control's. As a result, in can be seen in this scenario that Rad-Ap achieves a higher success rate than TCP congestion control

does, by an additional 0.4725%. While it achieves a higher success rate, it also manages to achieve a much higher average radius, being within 6.7459% of the optimum instead of TCP congestion control's distance of 28.3576%. As a result, it can be seen that Rad-Ap has a much better tradeoff between average radius and success rate than TCP congestion control. This should cause Rad-Ap to comprehensively outperform TCP congestion control when the success rate is set to be similar to or higher than that of TCP congestion control.

Regardless of whether *Goal* was set to 20 or 50, both versions of Rad-Ap outperformed Vowpal Wabbit directly in this case. Ras-Ap was better at handling the random jitter value than Vowpal Wabbit was, which lead to its better performance in this case. While it is likely that we could still train Vowpal Wabbit to outperform Rad-Ap in this case, recall that we are looking to prove Rad-Ap is good initial solution to this problem. While more data and passes over the data could improve Vowpal Wabbit's performance, it isn't going to have access to such things in a real world environment unless a system like Rad-Ap is deployed in the real world first for a reasonable time period. As such, we will not consider the solution of training Vowpal Wabbit to perform better in this dissertation.

The other two natural tests produced similar results with the same sort of mild variations that had been seen when running these tests on the initial version of the algorithm. Thus far, it appears that the new version is outperforming the previous version. By training on the same number of tests, the new version was able to achieve high success rate values while only slightly decreasing the average threshold. This is noteworthy, as the previous version of the algorithm would take longer to train as the desired success rate increased. If it were trained on the same number of test scenarios, the total gap between the optimal distance and a perfect success rate would widen as the accuracy increased because the algorithm would not have yet had time to train to the same levels of accuracy.

Test Name	Algorithm Used	Goal Value	Average Threshold Distance	Optimum Threshold Distance	Success Rate
Natural Test 1	RadAp v2	20	82.1998 ft	86.9851 ft	96.6027%
Natural Test 1	RadAp v2	50	81.1179 ft	86.9851 ft	98.5351%
Natural Test 2	RadAp v2	20	82.1944 ft	86.9851 ft	96.6018%
Natural Test 2	RadAp v2	50	81.1172 ft	86.9851 ft	98.5370%
Natural Test 3	RadAp v2	20	84.5320 ft	86.9851 ft	96.6009%
Natural Test 3	RadAp v2	50	83.8793 ft	86.9851 ft	98.5330%
Natural Tests	TCP CC	-	62.3182 ft	86.9851 ft	98.0626%
Natural Tests	VW	-	69.2725 ft	86.9541 ft	96.0094%

Table 4.5: Natural Test Results with Updated Algorithm

Test Name	Algorithm Used	Goal Value	Average Threshold Distance	Optimum Threshold Distance	Success Rate
Network Test 1	RadAp v2	20	84.9224 ft	91.9851 ft	95.1922%
Network Test 1	RadAp v2	50	84.3478 ft	91.9851 ft	97.9483%
Network Test 1	TCP CC	-	68.1645 ft	91.9851 ft	97.8747%
Network Test 1	VW	-	86.2005 ft	91.9541 ft	95.8563%
Network Test 2	RadAp v2	20	74.6379 ft	76.9851 ft	96.2606%
Network Test 2	RadAp v2	50	73.0487 ft	76.9851 ft	98.5020%
Network Test 2	TCP CC	-	57.2289 ft	76.9851 ft	97.8915%
Network Test 2	VW	-	65.1075 ft	76.9541 ft	95.2500%
Network Test 3	RadAp v2	20	56.0157 ft	86.9851 ft	93.9385%
Network Test 3	RadAp v2	50	70.3349 ft	86.9851 ft	92.3588%
Network Test 3	TCP CC	-	29.1211 ft	91.9851 ft	94.8940%
Network Test 3	VW	-	78.3317 ft	91.9541 ft	91.8750%
Network Test 4	RadAp v2	20	51.3073 ft	76.9851 ft	93.8458%
Network Test 4	RadAp v2	50	69.5401 ft	76.9851 ft	89.9492%
Network Test 4	TCP CC	-	28.2624 ft	76.9851 ft	94.8287%
Network Test 4	VW	-	62.0742 ft	76.9541 ft	91.1063%

Table 4.6: Network Condition Test Results with Updated Algorithm

4.3.2 Network Test Results for Updated Algorithm

Network Test 1, the test with no random jitter, had a notable decrease in success rate from what was seen in the Natural Test results. This is fairly interesting because the previous version of the algorithm saw an increase of performance on this case. The reason behind this is probably due to how the goal works. As mentioned previously, the goal attempts to get at least that number of successes in a row. In an environment with no randomness to the failure point, it can directly hone in on this goal, and does not need to add the previous buffer space to prevent statistical variance from causing two failures to occur without that many successes inbetween as frequently. It is worth noting, however, that this change in performance provides the opportunity to directly compare the two algorithms. Since the success rate was lowered and the threshold distance was increased, when comparing this test to the version run with the initial algorithm, it can be seen that both the average threshold distance and the success rate are higher in the new algorithm than they were with the initial version. This shows that Rad-Ap is not just on a different part of the threshold distance and success rate trade-off than the previous version but is, in fact, directly outperforming it. Since the second version is also more flexible and faster to train than the previous version, it can be seen how this update is superior to the initial version.

Because of the decrease in success rate in this particular case, TCP congestion control manages to get closer to the success rate of Rad-Ap with *Goal*'s value set to 50, now being only 0.0736% away from the success rate of Rad-Ap. As ever, TCP congestion control's average radius is still very far from Rad-Ap's, however.

This test was the test that Vowpal Wabbit performed best in originally, and as a result it still directly outperforms Version 2 when *Goal* is set to 20. When *Goal* is set to 50, Rad-Ap has a higher success rate, while Vowpal Wabbit has a closer proximity to the optimum radius. Since the gap between their proximities to the average radius is basically the same as the difference in their success rates, which one is preferable is going to be

application-specific. However, since we said that success rate was more important to us, we will say that we have a slight preference towards Rad-Ap for this test case.

In Network Test 2, the test with triple the random jitter, it can be seen how the algorithm again decreases slightly in terms of success rate from how it was performing in the natural tests to get significantly closer to the optimal value, just as the initial version of the algorithm did on this test. Interestingly, of all three versions, the new version with the goal set to 50 has the smallest decrease in terms of success rate among them. Once again Rad-Ap directly outperformed Vowpal Wabbit in this test, most likely due to the reintroduction and amplification of the random jitter value.

Network Test 3 is a particularly important test for the new algorithm because achieving the goal value of 50 is completely impossible when transmissions have a random failure chance. As such, Rad-Ap will have to be able to adapt its goal value to be able to perform well on this test. Ideally, the version with a goal of 20 would also adapt its value slightly, but because it is so close to the ideal goal in this environment it will be more difficult for the algorithm to realize this.

When looking at the results for Network Test 3, it can be seen that the algorithm did, in fact, detect the incompatible goal value and adjust itself when the goal was 50. The particular value it adjusted to was 13. The version of the algorithm with the goal value set to 20 seems to have not detected the difficulty of the environment, and thus not adjusted itself. As a result, the new algorithm with the goal value set to 50 had the best performance in this test case, as it achieved a success rate of 97.2198% on transmissions that did not randomly fail, which is well within the acceptable range. It also had an average threshold distance significantly closer to the average value than any of the other algorithms, with a distance of only 19.1414% to the optimum value. This is still far from ideal, but understandable when there is no easy way to distinguish between random failures and threshold failures. Future versions of the algorithm may look at increasing the sensitivity of the goal adaptation so that it would also change itself when the goal is set to 20. Other

future revisions may also consider collecting failure values and attempting to classify the failures as random failures or threshold failures based off their distribution pattern. However, due to the difficulties in distinguishing between these in a network environment, this dissertation will not investigate this issue.

Version 2 was unable to directly outperform Vowpal Wabbit in this scenario. While it managed to achieve a much higher average radius than Version 1 when *Goal* was set to 50, it was still not as high as Vowpal Wabbit's. It is, however, much closer to Vowpal Wabbit's results than before, and has a higher success rate than Vowpal Wabbit. As a result, which one is preferable in such an environment is going to be application-specific.

In Network Test 4, some of the same patterns that emerged in Network Test 3 can be seen. However, it is worth noting that in this test case the version of the new algorithm with the goal set to 50 has had only a very small decrease in average threshold distance, bringing it within 9.6707% of the optimum threshold distance. However, in exchange for this, the success rate has fallen even further, dropping out of my goal of only an additional 5% failure rate to an additional failure rate of 5.3166%. While this is not too far from my goal, it shows that I should consider revising the suboptimal-goal adjusting system to prevent it from being able to over-adjust as easily, as this shows that there are theoretical situations where it would over-adjust to the extent of hitting an extremely suboptimal goal value. While these could be mitigated with better initial declarations of the variables, I want the system to be as insensitive to suboptimal initial variable declarations as possible to assist with the initial deployment process that will have to occur in the future.

In a reversal from the previous test, Rad-Ap now has a higher optimum radius than Vowpal Wabbit but a lower success rate for this test. Both algorithms struggled here, as Rad-Ap has difficulties with random failures, while Vowpal Wabbit struggles with the tripled random jitter. Once again, which algorithm is preferable in this case is going to be application-specific.

Test Name	Algorithm Used	Goal Value	Average Threshold Distance	Optimum Threshold Distance	Success Rate
Scaling Test 1	RadAp v2	20	80.3571 ft	85.0000 ft	96.6009%
Scaling Test 1	RadAp v2	50	79.2885 ft	85.0000 ft	98.5385%
Scaling Test 1	TCP CC	-	61.5239 ft	85.0000 ft	98.0325%
Scaling Test 1	VW	-	67.7948 ft	85.0000 ft	96.0141%
Scaling Test 2	RadAp v2	20	85.8982 ft	90.9567 ft	96.6072%
Scaling Test 2	RadAp v2	50	84.9236 ft	90.9567 ft	98.5273%
Scaling Test 2	TCP CC	-	63.9906 ft	91.1076 ft	98.1272%
Scaling Test 2	VW	-	72.4181 ft	91.0762 ft	96.3102%
Scaling Test 3	RadAp v2	20	75.6804 ft	102.113 ft	96.5940%
Scaling Test 3	RadAp v2	50	74.8534 ft	102.113 ft	98.4312%
Scaling Test 3	TCP CC	-	69.8315 ft	102.113 ft	98.3113%
Scaling Test 3	VW	-	78.3050 ft	102.062 ft	96.4491%

Table 4.7: Scaling Rate Test Results with Updated Algorithm

4.3.3 Scaling Rate Tests for Updated Algorithm

For both of the two configurations of Rad-Ap tested, the results on Scaling Test 1 were exactly what one would expect based on the previous results of these algorithms on the natural tests and the relative performance of the original version of the algorithm. As a result, the test show the same pattern of the new algorithm with the goal set to 20 having a slightly lower average radius than the original version in exchange for a notably higher success rate. The version with the goal set to 50 has an even lower average radius but a much higher success rate. The results of Scaling Test 2 are also very similar to those of Scaling Test 1. Additionally, Rad-Ap directly outperformed Vowpal Wabbit directly in these case, much as it did in the Natural Tests. As a result, it can be seen that the updated version of the algorithm works similarly well when the scaling rate is set to 1, square root of n, or n.

Interestingly, the updated version of the algorithm appears to have performed significantly less well when it comes to density squared scaling. The average threshold distances are significantly lower than they were for the original version of the algorithm in this test case. Unlike the original version where the success rate saw a small increase over the re-

sults of Scaling Tests 1 and 2, it declined slightly. The exact reasons for this are currently unknown, and investigations are ongoing. Due to these changes, the original version appears to perform better in a density squared scaling case, but the new algorithm still appears to outperform TCP congestion control. However, it loses its direct advantage over Vowpal Wabbit, and once again presents a tradeoff between a higher optimum radius and a higher success rate.

4.4 Version 3

For Version 3, my primary objective was to compare its results to those of Version 2 at their best, as if Version 3 performed worse than Version 2, there would be no point in implementing it. As a result, I elected to test Version 3 at only the *Goal* value of 50, as this is the condition in which Version 2 performed the best. The purpose of the *Goal* 20 tests was to allow for direct comparisons between Version 2 and Version 1, so now that I have established the superiority of Version 2, we can compare it directly to Version 3 at the *Goal* 50 level.

Version 3 introduced the concept of switching to an additive increase mode once the model had been sufficiently trained. As such, for Version 3 to be considered a success we need to see higher average threshold distances for the same success rate. When we see this, it will indicate that the switch to an additive mode is helpful to the algorithm.

4.4.1 Natural Test Results for Version 3

Table 4.8 shows the results of running the three natural tests on Version 3 of my algorithm. We can see for these cases that the algorithm got closer to the optimum value in all of the test cases. The first test case now got within 4.1455% of the optimum value as opposed to the 6.7451% difference that Version 2 saw when its *Goal* value was also set to 50. This trend continues in the next two cases, with Natural Test 2 getting within 4.1584% of the optimum value and Natural Test 3 getting within 0.7088% of the optimum value, while Version 2 only got within 6.7459% and 3.5705%, respectively. As in previous cases, Natu-

Test Name	Algorithm Used	Average Threshold Distance	Optimum Threshold Distance	Success Rate
Natural Test 1	RadAp v3	83.4536 ft	87.0628 ft	98.2556%
Natural Test 2	RadAp v3	83.4423 ft	87.0628 ft	98.2559%
Natural Test 3	RadAp v3	86.4447 ft	87.0628 ft	98.2536%
Natural Tests	TCP CC	62.3182 ft	86.9851 ft	98.0626%
Natural Tests	VW	69.2725 ft	86.9541 ft	96.0094%

Table 4.8: Version 3 Natural Test Results

ral Test 3's results may be somewhat misleading due to large initial number throwing off the average. While Version 3's average threshold distance has improved from that of Version 2's, it was not without any trade-offs. When comparing the two, we can see that the success rate has decreased slightly, from the range of 98.5330% - 98.5370% to the range of 98.2536% - 98.2559%. While this would seem to be a disadvantage, the Version 3 tests reached much greater proximity to the optimum distance than Version 2 to the extent that all tests got closer to the average threshold distance than even the *Goal* 20 tests of Version 2. This would indicate that this system is capable of achieving a higher success rate and average threshold distance than Version 2 with some slight tuning. In this case, this could be achieved by setting the *Goal* to 51 or 52 for the Version 3 tests. However, for the sake of fairness I elected not to adjust the value *Goal* for these tests. Due to these factors, I concluded that Version 3 would outperform Version 2 with a better tradeoff between success rate and average radius for these tests.

Despite the minor tradeoff, Version 3 still directly outperformed TCP congestion control and Vowpal Wabbit in all of these test cases.

4.4.2 Network Test Results for Version 3

Network Test 1, the test with no random jitter, shows a notable increase in average threshold distance between Version 2 and Version 3. The difference between the average value and the optimum value decreases from the 8.3028% we saw in Version 2 to the much lower 5.3959% of Version 3. This result is not unexpected, as we saw in the natural tests

Test Name	Algorithm Used	Average Threshold Distance	Optimum Threshold Distance	Success Rate
Network Test 1	RadAp v3	87.0942 ft	92.0628 ft	98.2547%
Network Test 1	TCP CC	68.1645 ft	91.9851 ft	97.8747%
Network Test 1	VW	86.2005 ft	91.9541 ft	95.8563%
Network Test 2	RadAp v3	74.9444 ft	77.0628 ft	98.2582%
Network Test 2	TCP CC	57.2289 ft	76.9851 ft	97.8915%
Network Test 2	VW	65.1075 ft	76.9541 ft	95.2500%
Network Test 3	RadAp v3	77.3983 ft	92.0628 ft	91.4085%
Network Test 3	TCP CC	29.1211 ft	91.9851 ft	94.8940%
Network Test 3	VW	78.3317 ft	91.9541 ft	91.8750%
Network Test 4	RadAp v3	61.5357 ft	77.0628 ft	93.6090%
Network Test 4	TCP CC	28.2624 ft	76.9851 ft	94.8287%
Network Test 4	VW	62.0742 ft	76.9541 ft	91.1063%

Table 4.9: Version 3 Network Test Results

that Version 3 can achieve higher average radiuses than Version 2 could. What is unexpected is that the success rate is now higher for Version 3, at 98.2546% as opposed to Version 2's 97.9483%. In fact, the success rate is basically unchanged from the natural test's success rate. This makes some sense, however, as when the system is operating in additive mode, the random jitter would not effect the success rate, as the algorithm is now growing linearly towards whatever value it got cut off at before. With the random jitter on, the number of successes in a row would average out to be the same as if the cutoff value were a constant. As a result, Version 3 appears to be slightly more resilient than Version 2. I hypothesize that this will continue to be true as when Version 3 encounters difficulties it reverts back to have the same performance as Version 2, so it should not be any less resilient than Version 2.

This increase in performance on Network Test 1 means that for the first time of any of the versions of Rad-Ap we tested, Version 3 will directly outperform Vowpal Wabbit in this test. This is very encouraging, as we previously noted how this test case was particularly well suited for Vowpal Wabbit, and is where Vowpal Wabbit had its best performance. Being able to outperform Vowpal Wabbit in this case means that our algorithm

can outperform Vowpal Wabbit at its best, and not just in cases that are not suited for Vowpal Wabbit.

Network Test 2, the test with triple random jitter, sees a return to form with results much similar to what we saw in the Natural Tests. In this test, Version 3 has a higher average threshold distance, being within 2.7489% of the optimum threshold distance, while Version 2 was only within 5.1132% of the optimum threshold distance. Both results are closer to the optimum threshold distance they were for the Natural Tests, and seem to have improved a fairly similar amount. As a result, it would seem that from an average threshold distance perspective, both versions handled this similarly well, with the previous performance edge of Version 3 being maintained. Interestingly, Version 2's success rate dipped slightly here, from its range of 98.5330% - 98.5370% on the natural tests to 98.5020%, while Version 3's increased slightly, from its range of 98.2536% - 98.2559% in the natural tests to 98.2582%. However, even though the gap in success rates between the two algorithms is small, this change was not enough to make a notable difference. As such, I draw the same conclusion for this test case as I did for the Natural Tests: Version 3 performs better overall in this scenario.

In Network Test 3, the test with a 5% random failure chance, we see that Version 3 has a much higher average threshold distance, being within 11.1006% of the optimum threshold distance as opposed to Version 2's 19.1414% proximity to the optimum threshold distance. However, this large increase in proximity to the optimum threshold distance comes at a cost: Version 3 has a significantly lower success rate in this test, falling from Version 2's 92.3588% to 91.4085%. While this difference is only slightly less than 1%, we have to keep in mind that 95% is the maximum possible success rate in this test with a 5% random failure chance, so this actually represents a 12.4362% increase in the number of failures introduced by my algorithm. This means that Version 3 offers a 12.4362% increase in failure rate in exchange for a decrease in the distance from the optimum threshold distance of 42.0074%, which is equivalent to a 9.9434% increase in average threshold distance. As

such, it would appear that which of Version 2 and Version 3 performs better in this case will largely be application specific. Applications that require a high success rate would favor Version 2, while applications that require a large amount of data to be collected and transmitted would favor Version 3.

Version 3 is actually directly outperformed by Vowpal Wabbit in this case, by very small margins on both the average radius and the success rate. As a result, we can see that in the case of an environment with a 5% random failure rate, Vowpal Wabbit has a slightly better tradeoff between average radius and success rate than Rad-Ap does.

Network Test 4, the test with triple random jitter and a 5% random failure rate sees a strange reversal in performance. In this test case, Version 3 had a much higher success rate, at 93.6090%, while Version 2 had a mere 89.9494% success rate. In exchanges, Version 3 had a lower average threshold distance, being 20.1486% from the optimal value, while Version 2 was only 9.6707% away from the optimum distance. Since this test and the previous one were designed to test the *Goal*-decreasing mechanism of the algorithm, it would appear that the expected performance after this mechanism is used is not particularly consistent. While both algorithms achieved acceptable results on this test and the previous one, whether they favored a high success rate or a high average threshold distance seems to be difficult to predict. As such, I must conclude that while the *Goal*-decreasing mechanism is functional, it might be wise to re-configure the algorithm if this mechanism kicks in so that you can better customize the balance between success rate and threshold distance for your particular algorithm.

Comparing against Vowpal Wabbit for Network Test 4, we see that there is still a tradeoff between Version 3 and Vowpal Wabbit, but now the numbers are more stacked in favor of Version 3. Vowpal Wabbit is now only 0.8126% closer to the optimum radius, while Rad-Ap has a 2.5027% higher success rate. Since Rad-Ap offers a significantly higher success rate for only a moderate tradeoff, and we consider success rate to be more important, we will declare Rad-Ap the preferred algorithm for this test case.

In general, Version 3 seems to have done as well as or better on the Network Tests than Version 2. While there are some inconsistencies on what level of tradeoff they settled on for Network Test 3 and 4, the actual tradeoff between the numbers seems to have maintained their previous balance. Version 3 can be seen to have a better total mix of average threshold distance and success rate on the Natural Tests and Network Tests 1 and 2. Thus, Version 3 seems to be outperforming Version 2 overall on these tests.

4.4.3 Scaling Test Results for Version 3

The Scaling Tests in general had results much in line with what we would expect from the Natural Tests. In Scaling Test 1, the test with no scaling, we saw Version 3 get within 4.1332% of the optimum threshold distance, a notable improvement over Version 2's 6.7194% difference. Meanwhile, Version 3's success rate saw the same small decline from Version 2's 98.5385% to 98.2556%. Scaling Test 2 had similar results, with version 3 getting within 4.1685% of the optimum threshold distance with a 98.2558% success rate, while Version 2 got within 6.5815% of the optimum threshold distance with a 98.5273% success rate. Regardless of these small changes, Version 3 still directly outperformed TCP congestion control and Vowpal Wabbit in both of these test cases.

Scaling Test 3, the test with n squared scaling, had slightly more interesting results. The difference in distance to the optimum threshold distance declined marginally, with Version 3 being within 24.8851% of the optimum threshold distance while Version 2 was within 26.6955%. Additionally, Version 2's success rate fell in this case, dipping to 98.4312%, while Version 3's remained solidly at 98.2557%. This may be indicative of Version 3 being more stable in such a case, though it is hard to say conclusively since Version 3's success rate is still slightly lesser than Version 2's.

Version 3 performed better compared to Vowpal Wabbit in this test case, having a proximity to the optimum radius only 1.6080% lower than that of Vowpal Wabbit's while its success rate was 1.8066% higher. While which of these is preferred is going to be based on the application, since it was stated that I consider success rate to be more important, I

Test Name	Algorithm Used	Average Threshold Distance	Optimum Threshold Distance	Success Rate
Scaling Test 1	RadAp v3	81.4868 ft	85.0000 ft	98.2556%
Scaling Test 1	TCP CC	61.5239 ft	85.0000 ft	98.0325%
Scaling Test 1	VW	67.7948 ft	85.0000 ft	96.0141%
Scaling Test 2	RadAp v3	87.3413 ft	91.1414 ft	98.2558%
Scaling Test 2	TCP CC	63.9906 ft	91.1076 ft	98.1272%
Scaling Test 2	VW	72.4181 ft	91.0762 ft	96.3102%
Scaling Test 3	RadAp v3	76.7164 ft	102.132 ft	98.2557%
Scaling Test 3	TCP CC	69.8315 ft	102.113 ft	98.3113%
Scaling Test 3	VW	78.3050 ft	102.062 ft	96.4491%

Table 4.10: Version 3 Scaling Test Results

will declare Version 3 the preferred algorithm for my environment.

Now that I have conducted all of the tests on all of the algorithms, it can be seen that the most recent version of Rad-Ap is capable of outperforming TCP congestion control in all of the cases tested. Additionally, it outperformed Vowpal Wabbit despite Vowpal Wabbit having higher requirements on the device's computational power. Version 3 achieved a performance that was at least as good as that of Vowpal Wabbit's in nine of the ten test cases, and was often much better. Additionally, in the one test case it was outperformed by Vowpal Wabbit, the difference was extremely small, to the point that Rad-Ap Version 3 would often be preferred over Vowpal Wabbit anyways due to its lower requirements on computational power. As a result, I conclude that Version 3 is the best of the five algorithms tested for fulfilling the demands of this testing suite.

4.5 ns-3 Results

Version 3 was also tested by being run in a network environment simulated by ns-3. For this test, I had the radius Rad-Ap suggested represent the distance from the vehicle to the center of the intersection. While this may be slightly different from how it will actually be implemented, it should be similar enough to give a good idea of how this algorithm will perform in a real-world scenario. Since this is a more complex simulation, the idea of

an optimal radius is no longer as straightforward to determine. As such, for this analysis I will be looking at the number of packets that were successfully sent in my system and comparing it to the number of packets that failed to send when my system recommended that they do so. I will also compare this to the number of packets that were not sent due to my system recommending that they avoid sending them when they were within normal communication range. Note that this metric does not necessarily indicate a packet that would have been successfully sent if Rad-Ap had have allowed it, as a packet that would have been sent could have collided with other transmissions and not been successfully delivered regardless. However, this number may give us some sort of clue as the number of packets that could have been successfully received that this network missed out on the opportunity to send. For each of these metrics I will also briefly discuss the standard deviation across the 10 runs to ensure that the number the test is providing is actually meaningful.

In this test scenario, Rad-Ap achieved an average of 4,714,235 packets delivered. Across the ten runs, I observed a standard deviation of 95.5476 packets, meaning that the true average likely likes close enough to the observed average to disregard a large amount of variance. While my system successfully delivered 4,714,235 packets on average, it would have 53,650 packets that failed to be delivered in the average case. This number is slightly less consistent than the average number of messages delivered, with a standard variance of 186.0654. If we allow for the true mean being up to three standard variances out, which is extremely likely to be the case, then this means that we would expect a number of failures between 53,091 and 54,208 in a real world scenario with a similar number of packets sent. This correlates to a failure rate between 1.1262% and 1.1499%, which is completely within acceptable bounds. As such, it appears that even in a more realistic test environment, Rad-Ap does a good job of keeping the failure rate to a relatively low percentage.

Failure rate does not tell the whole story of my algorithm's performance. We must also look at the number of packets that my system prevented from being sent to see it Rad-Ap

was limiting the throughput of the system too much. Across ten runs, I observed Rad-Ap excluding an average of 22,734 packets, with a standard deviation of 100.9246 packets. With the same error bounds of three standard deviations, this correlates to a 0.4758% to 0.4887% decrease in packets sent. Since, as I mentioned previously, packets not sent do not necessarily correlate to a loss in potential traffic in a one to one ration, this amount of packets withheld is also completely within acceptable bounds. As such, I conclude that Rad-Ap performed well in my more realistic simulated environment and will likely be able to perform well in the real world.

4.6 Algorithm Efficiency

In this section, I will discuss the efficiency of all parts of Rad-Ap from a time and space complexity perspective. First, I will investigate the complexity of the beaconing system. In the algorithm, the only nodes that beacon are the cluster heads. They use the beaconing method set forth in 802.11p's WBSS system. Since each cluster head will be beaconing, the network capacity consumed by this part of the algorithm will be proportional to either the square root of n or directly proportional to n, depending on the network environment. When the network is not particularly crowded and the biggest limiter to vehicular communications is distance, the number of cluster heads will be proportional to the square root of the number of vehicles present. This should be the case since each cluster head can serve an area proportional to their broadcasting range squared. This value is squared rather than cubed or directly proportional as vehicles at an intersection will most often occupy a two dimensional area around the cluster head. However, there may be some differences for special cases.

As long as broadcast range is the limiting factor, increasing the number of nodes will also increase the number of nodes that can fit within a cluster head's broadcasting area; therefore, the number of cluster heads will increase primarily with the area that the cars occupy. However, once the number of vehicles reaches the threshold where adding more ve-

hicles to a cluster is determined to be harmful to the network's performance, clusters will not grow past this point. Thus, the system will require a number of cluster heads directly proportional to the number of vehicles in the worst case. As such, beaconing will consume a portion of the network's bandwidth of $O(n)$ space in the worst case, but there will be many cases where it only requires $O(\sqrt{n})$ space. Also note that these operations are performed on a separate channel than the data channel for cluster communications.

Next I will look at the task of joining a cluster. When a node joins a cluster, it needs to send a message to the cluster head, either a message solely for the purpose of claiming that the node is joining the cluster or a data packet with an embedded message identifying that it is joining the cluster, based on the mode of operation. In the second mode of operation the increase to the load of the network is either zero or negligible; however, since it requires $O(n)$ traffic to be passed, this is a $O(n)$ operation regardless of which mode of operation is used. When a node is leaving a cluster, it does not have to send a notification to the cluster head; therefore, leaving a cluster is a $O(1)$ operation. Since node backoff messages are sent in the beacons, nodes leaving for this reason will have the same total running time as the beaconing, but it will not require any additional communications besides the beacons. However, once again, note that these messages will be passed as parts of other messages if the system is operating as intended.

Cluster heads are selected using two different methods in this system based on whether they are an initial declaration or a cluster head handoff that occurs later. When the initial cluster head selection is performed, the system uses a first declaration wins scheme. Since this involves no collaboration and only a single broadcasted declaration, this operation occurs in $O(1)$ time. After this, the nodes must engage in $O(n)$ communication where they share their attributes. Like the join messages, this conversation is intended to be piggybacked on other messages, and those should not significantly increase the utilization of the channel. Once the nodes have finished their $O(n)$ communication of node attributes, the nodes may undergo cluster head selection again if the cluster head does not stop them

from doing so. Regardless of whether this process is stopped or not, this process is $O(n)$ because the nodes have already exchanged their information, only rely on a timer to declare, and require only a single declaration or stop message.

Based off the previous discussion, it can be seen that this system is largely a $O(n)$ system. In the case that the clusters are already exchanging information on at least the magnitude of $O(n)$, the algorithm will have a very minimal impact on the data channel. Meanwhile, the management channel will also consume resources up to the order of $O(n)$, but will often consume only $O(\sqrt{n})$ resources. Additionally, even in the worst case where it consumes resources of $O(n)$ just like an active system would, it is worth noting that it significantly lowers the number of vehicles using the control channel compared to a system that has all nodes beacon. This makes it much more unlikely for the algorithm to exceed the 150 vehicle limit of the control channel mentioned in Section 2.3.5.

Since this algorithm is intended to be run for a significant period of time, the algorithm must ensure that it will not store too much data or require increasingly complex computations over time. Either of these could overtax a more limited computational device connected to a VANET. The system stores two different types of values: timestamp values that correspond to the entry of vehicles into the cluster and the current mapped values of densities to cluster sizes. The number of timestamp values stored at any particular time is limited to a constant 168 per element, which means that the amount of data to be processed will not increase as the cluster head processes an increasing number of unique vehicle entries. Additionally, a queue structure is used to manage the data, which allows the algorithm to insert and remove elements from the data in constant time. Since the algorithm is only concerned with the density of the traffic, only the front, back, and current size of the queue are needed. This allows density to be calculated in constant time as well [69].

Unlike the collected traffic data, the mapping of the densities to radius values is stored in a red-black tree, and accessing and updating the elements of the map takes $O(log(n))$

time. However, in this case n will grow with the number of new densities seen. In the typical case, it would not grow linearly to the number of entries processed, as the distribution of densities forms a curve somewhat similar to a normal distribution. In an average case, map access would be expected to take $O(log(log(n)))$ time. As a result, the algorithm scales well with large amounts of data. The algorithm does slightly less well from a space complexity standpoint. Most of the data collected is maintained within a bounds of 168 entries each in 48 queues, but the mapping of densities to radii will grow at approximately a $O(log(n))$ rate, as previously discussed. Therefore, additional precautions will have to be taken to ensure that the device running the task does not run out of memory resources. If future investigations reveal this to be a problem, a pruning operation may be implemented to maintain a constant sized map. That would reduce the time and space complexity of this algorithm to $O(1)$. However, this growth rate is not very large, and may not pose a problem to the devices on which the application is being run [69].

4.7 Summary

Now that all of the test have been run and I have discussed how the results compare to each other, I will determine the overall bearing they have on my algorithm. In my initial version of the algorithm, I compared it against a similar algorithm that is often used in a related field, the TCP congestion control algorithm. I found that my algorithm outperformed the TCP congestion control algorithm in the testing environment I set up. I then introduced and tested two more versions of the algorithm, each of which performed better than the last due to the addition of new features. Since these algorithms outperformed their previous versions, and the first version outperformed the TCP congestion control algorithm, I concluded that my algorithm would outperform the TCP congestion control algorithm based on my test cases. In fact, the algorithm easily met some very stringent requirements that I set up before running the tests.

I also investigated whether my algorithm could outperform a more computationally

complex algorithm that was part of the Vowpal Wabbit package. I found that my algorithm was able to directly outperform Vowpal Wabbit in 7/10 test cases, and achieve a superior performance overall in 9/10 test cases. Since my algorithm was also lighter weight than Vowpal Wabbit and deployed without any advance training, I determined that my algorithm was better suited to use as the initial solution in a VANET environment than Vowpal Wabbit's logistic regression algorithm.

However, I could not say at this point that the algorithm was certain to work in the field, as the test cases I set up were performing very simplistic simulations of what a network environment may actually be like. To help address these concerns, I performed additional testing in a more complex network simulation environment that is often used in the field, namely ns-3. Since ns-3 is commonly used and accepted as an effective network simulator, getting good results in a test case where the network is simulated by ns-3 would provide additional credibility to the effectiveness of my algorithm. In these tests, I observed results very similar to those I observed in the initial battery of tests. Since the results were so similar, it helps inspire confidence that the Rad-Ap algorithm will be effective in the real world. This also indicated that the simplistic network environment I set up for my tests was also a reasonable approximation of real world conditions, making those results more meaningful as well.

Once I had finished running extremely large practical test batteries with different levels of simulators, I also wanted to take a look at my algorithm from a theoretical perspective to ensure that there weren't any running-time based issues that would cause Rad-Ap to suffer in the long term. In these investigations I concluded that the running time complexity and the space complexity of Rad-Ap were sufficiently low to not be a concern for the intended method of deployment.

Now that I have thoroughly investigated the algorithm from both a practical and theoretical perspective and found no particular areas of concern, I would like to conclude that Rad-AP is an effective first solution to this problem and can be used to help create

VANET clusters at intersections in the future.

CHAPTER 5

CONCLUSION

In this dissertation, I discussed the current state of VANET technologies. I started by discussing some of the available network-layer technologies for vehicular communications, primarily in the form of 802.11p. I discussed some of the features this provides to assist in the formation and advertising of VANET clusters. Primary among these was the WBSS, a new feature that allows interested vehicles to receive and evaluate nearby VANET clusters. I then discussed two of the main philosophies to clustering algorithm designs, active and passive techniques. Active techniques allow the network to create more accurate and reliable clusters, but can consume a large portion or even all of the available network resources. Passive techniques consume an incredibly small portion of the available network resources, but have lower knowledge of the network and can only perform effectively when network traffic is high. I also looked at a hybrid method that switches between the two schemes based on the current level of network traffic. However, because this method required accurate network information to make its decision on when to switch, it consumed a significantly larger portion of network resources than a passive scheme even when operating in passive mode.

Because I saw these shortcomings in current passive and hybrid active-passive techniques, I introduced a new scheme that attempts to use low-level, lightweight, machine-learning techniques to determine what the optimal network condition would be based on the reduced information available in a passive system. This technique uses online machine learning to adapt its answers based on the current condition of the network. It is

intended to be deployed initially at intersections since the greater stability and higher network traffic makes intersections more suitable to a passive technique than an active one. This technique was developed with a number of key characteristics in mind, including being lightweight, unit agnostic, affected less by initial parameter selection, able to record and distribute training data, highly accurate, and secure.

The technique being lightweight is essential due to the potential low processing power of devices connected to a VANET. If the algorithm consumes too many resources, weaker devices on the network may not be able to run it, lowering its utility. Being unit agnostic helps the algorithm be more universally deployable, as it can be used regardless of the metric the system it is in uses. This also allows it to be prepared for a future where new, superior metrics are discovered and used, and allows it to not be permanently linked to the field's current models. The algorithm being affected less by parameter selection allows it to be used with extremely little foreknowledge of the environment, and reduces the amount of time and data needed to configure it. This makes it particularly suited for a first attempt at this issue, which is the intended role of this algorithm. The algorithm being capable of recording and distributing training data also assists this effort, as it allows the algorithm to spread faster by having other instances of it be pre-trained on previous instances' collected data. Additionally, this data can be used by future algorithms in this field to help train more advanced models. However, it is key that while I address these concerns, I also keep the algorithm highly accurate. If the algorithm is not accurate enough, then it serves no practical use even with all these features accounted for. As such, it must be able to get within a reasonable range of the optimal values. Finally, the algorithm being secure allows it to be deployed without any concerns over it introducing vulnerabilities to the system it is employed in. This is necessary since it is intended to be deployed in a real world environment where it could be targeted by attackers.

With these design parameters in mind, I introduced my algorithm, called Rad-Ap. Rad-Ap uses a multi-faceted Multiplicative Increase, Multiplicative Decrease algorithm

to quickly hone in on the optimal value regardless of its proximity to it. Once it gets close enough to this value, it switches to an Additive Increase, Multiplicative Decrease mode to give it an increased amount of stability. It has a number of mechanics to assist with the process of seeking the optimal values, as well as a number of preventative systems to avoid some potential failure states. It complies to all of the previously discussed features that my envisioned algorithm would need to fulfill to achieve its purpose.

Once I finished designing this algorithm, I ran it through an exhaustive suite of tests to ensure that it could work in all envisioned scenarios. I also compared it to another algorithm that was intended to fulfill a similar role, the TCP congestion control algorithm. These tests found that my algorithm performed very stably across a large number of possible scenarios, and more recent versions of the algorithm were capable of outperforming TCP congestion control in all observed metrics for the majority of the test cases. In cases where TCP congestion control outperformed my algorithm in one of the metrics, it did so at the cost of being vastly outperformed in another metric, causing my algorithm to have the better overall performance. As such, I concluded that my algorithm would outperform TCP congestion control when deployed in these circumstances.

Furthermore, I compared it to a more advanced and processor-consuming machine learning algorithm, Vowpal Wabbit's logistic regression algorithm. I found that my algorithm was capable of achieving directly superior performance in the majority of the cases tested, and found it to have better overall performance in all but one of the test cases. Since my algorithm also has less demands on the system it is being run on than Vowpal Wabbit, I concluded that my algorithm was better suited to this environment than Vowpal Wabbit.

I then ran tests in a more accurate simulation environment, using the often-employed network simulator, ns-3. The algorithm performed similarly in this environment to how it did in the previous batch of tests, proving that the algorithm should be able to function well in the real world, while also providing additional credence to the previous tests that

showed the superiority of the algorithm to other algorithms in the field.

After performing practical tests on this algorithm, I also analysed the theoretical time complexity and space complexity of this technique and algorithm I introduced. In doing so, I determined that the clustering technique would not increase the time complexity that modern clustering techniques already have. As such, I concluded that this system was sufficient from a time complexity standpoint. I then analysed my algorithm to determine its time and space complexity. I found that it had good time and space complexities, at a $O(log(log(n)))$ running time and $O(log(n))$ space used. I determined that this was likely sufficient for most modes of operation, but introduced a variant that would have constant running time and space complexity if my initial version was found to have running time issues in the long term.

In summary, I evaluated the current state of the field and found an area where modern methods are lacking. Then I identified the characteristics that an algorithm would need to have to fulfill this role, and designed an algorithm to fit them. Then I tested and analysed the algorithm and determined that it was an effective solution to this problem. Therefore, I conclude that my passive system and the Radius Approximator tool would be well suited to an initial approach at solving issues with VANET communications at high density locations like intersections.

CHAPTER 6

FUTURE WORKS

While I have discussed many tasks that I have accomplished so far in the production and testing of the algorithm, there are many future directions that I am still working on in the development of this algorithm. I will now begin to discuss some of my plans for the future as well as future avenues of study on this project. I will then discuss some of the reasoning behind these future direction and potential advantages of pursuing such goals.

In the current state of the algorithm, weekdays and weekends are handled separately. This is the case because typical traffic patterns differ across these two types of days. These differences in traffic patterns could make it much more difficult for the algorithm to successfully serve the network at its maximum capacity if it did not handle them separately. However, there is another category of day that will cause different traffic behaviors that the algorithm does not account for: Holidays. These days have very different traffic patterns than normal days as they remove the flow of traffic related to people going to work that normally occurs on weekdays, and certain holidays may also cause an increase in traffic that is travelling long distances. As such, it would be appropriate if the program had a separate handler for them. Thus, for future work I would like to add on a section of the algorithm that considers the case of holidays and either handles them separately or categorizes them as weekend days regardless of which day of the week it is. I hope to run experiments to determine which of these courses of action will produce better results for the algorithm as well.

Currently, my results are only focused on the performance of the approximation algo-

rithm to assist in the process of cluster selection for my introduced passive scheme. While I discuss some of the theoretical performances of the full clustering scheme, I do not provide any experimental results showing how effective this clustering scheme is. As such, I hope to one day run a full simulation that accounts for all aspects of the clustering scheme and compares it to other suggested clustering schemes. I hope to compare its performance in terms of success of the cluster selection, average throughput of the clusters, and required overhead network traffic. I hope to compare it both against active clustering algorithms and passive clustering algorithms. In the case of active clustering algorithms, I hope that the algorithm will be able to achieve a fairly comparable level of success in terms of cluster selection and throughput while offering a greatly reduced network overhead. This would allow the algorithm to at least be a sufficient alternative to active clustering schemes when the network becomes overcrowded, as it likely would at intersections. When comparing it against passive schemes, I would mostly be looking to see if the accuracy of the algorithm's clustering scheme could outperform them, as the algorithm is not offering any additional services over them in terms of reducing network overhead. If the algorithm could provide these advantages, then that would mean my efforts were fully successful.

Once I develop and test the full clustering algorithm using my Rad-Ap approximator, my next step would be to test it in increasingly realistic conditions, culminating in a actual real-world test. Since VANET devices are starting to be deployed in both the United States of America and across Europe, it may be possible to find some opportunity to run tests of clustering algorithms in a testbed that uses real vehicles with VANET-capable devices. Given such a chance, I would be very interested to see what types of results this algorithm can achieve and how it would perform against other current state-of-the-art algorithms.

REFERENCES

[1] Ameer Ahmed Abbasi and Mohamed Younis. A survey on clustering algorithms for wireless sensor networks. *Computer communications*, 30(14-15):2826–2841, 2007.

[2] Abdeldime MS Abdelgader and Wu Lenan. The physical layer of the ieee 802.11 p wave communication standard: the specifications and challenges. In *Proceedings of the world congress on engineering and computer science*, volume 2, pages 22–24, 2014.

[3] Ahmad Abuashour and Michel Kadoch. An intersection dynamic vanet routing protocol for a grid scenario. In *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 25–31. IEEE, 2017.

[4] Ahmed Ahizoune and Abdelhakim Hafid. A new stability based clustering algorithm (sbca) for vanets. In *37th Annual IEEE Conference on Local Computer Networks-Workshops*, pages 843–847. IEEE, 2012.

[5] Waleed Ahsan, Muhammad Fahad Khan, Farhan Aadil, Muazzam Maqsood, Staish Ashraf, Yunyoung Nam, and Seungmin Rho. Optimized node clustering in vanets by using meta-heuristic algorithms. *Electronics*, 9(3):394, 2020.

[6] Mohammad S Almalag, Stephan Olariu, and Michele C Weigle. Tdma cluster-based mac for vanets (tc-mac). In *2012 IEEE international symposium on a world of wireless, mobile and multimedia networks (WoWMoM)*, pages 1–6. IEEE, 2012.

[7] Saleha Mubarak AlMheiri and Hend Saeed AlQamzi. Manets and vanets clustering algorithms: A survey. In *2015 IEEE 8th GCC Conference & Exhibition*, pages 1–6. IEEE, 2015.

[8] Eitan Altman, KE Avrachenkov, and BJ Prabhu. Fairness in mimd congestion control algorithms. *Telecommunication Systems*, 30(4):387–415, 2005.

[9] Marwane Ayaida, Mohtadi Barhoumi, Hacène Fouchal, Yacine Ghamri-Doudane, and Lissan Afilal. Phrhls: A movement-prediction-based joint routing and hierarchical location service for vanets. In *2013 IEEE international conference on communications (ICC)*, pages 1424–1428. IEEE, 2013.

[10] Xu Bao, Haijian Li, Guoqiang Zhao, Lv Chang, Jun Zhou, and Yun Li. Efficient clustering v2v routing based on pso in vanets. *Measurement*, 152:107306, 2020.

[11] Stefano Basagni. Distributed clustering for ad hoc networks. In *ispan*, page 310. IEEE, 1999.

[12] Abderrahim Benslimane, Tarik Taleb, and Rajarajan Sivaraj. Dynamic clustering-based adaptive mobile gateway management in integrated vanet—3g heterogeneous wireless networks. *IEEE Journal on Selected Areas in Communications*, 29(3):559–570, 2011.

[13] Giuseppe Bianchi and Ilenia Tinnirello. Analysis of priority mechanisms based on differentiated inter frame spacing in csma-ca. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No. 03CH37484)*, volume 3, pages 1401–1405. IEEE, 2003.

[14] Luciano Bononi and Marco Di Felice. A cross layered mac and clustering scheme for efficient broadcast in vanets. In *2007 IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 1–8. IEEE, 2007.

[15] Azzedine Boukerche, Zhenxia Zhang, and Xin Fei. Reducing handoff latency for nemo-based vehicular ad hoc networks. In *2011 IEEE Global Telecommunications Conference-GLOBECOM 2011*, pages 1–5. IEEE, 2011.

[16] Neelakantan Pattathil Chandrasekharamenon and Babu AnchareV. Connectivity analysis of one-dimensional vehicular ad hoc networks in fading channels. *EURASIP Journal on Wireless Communications and Networking*, 2012(1):1, 2012.

[17] Mainak Chatterjee, Sajal K Das, and Damla Turgut. Wca: A weighted clustering algorithm for mobile ad hoc networks. *Cluster computing*, 5(2):193–204, 2002.

[18] Nevadita Chatterjee, Anupama Potluri, and Atul Negi. A self-organizing approach to manet clustering. In *High Performance Computing*, 2006.

[19] Yuzhong Chen, Mingyue Fang, Song Shi, Wenzhong Guo, and Xianghan Zheng. Distributed multi-hop clustering algorithm for vanets based on neighborhood follow. *Eurasip journal on Wireless communications and networking*, 2015(1):98, 2015.

[20] Jiu Jun Cheng, Gui Yuan Yuan, Meng Chu Zhou, Shang Ce Gao, Zhen Hua Huang, and Cong Liu. A connectivity prediction-based dynamic clustering model for vanet in an urban scene. *IEEE Internet of Things Journal*, 2020.

[21] Arslane Hamza Cherif, Khaled Boussetta, Gladys Diaz, and Didi Fedoua. Improving the performances of geographic vdtn routing protocols. In *2017 16th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pages 1–5. IEEE, 2017.

[22] Ching-Chuan Chiang, Hsiao-Kuang Wu, Winston Liu, and Mario Gerla. Routing in clustered multihop, mobile wireless networks with fading channel. In *proceedings of IEEE SICON*, volume 97, pages 197–211, 1997.

[23] Francesco Chiti, Romano Fantacci, and Giovanni Rigazzi. A mobility driven joint clustering and relay selection for ieee 802.11 p/wave vehicular networks. In *2014 IEEE International Conference on Communications (ICC)*, pages 348–353. IEEE, 2014.

[24] CISCO. 802.11ac: The fifth generation of wi-fi technical white paper. 2018.

[25] David Clark. The design philosophy of the darpa internet protocols. In *Symposium proceedings on Communications architectures and protocols*, pages 106–114, 1988.

[26] Craig Cooper, Daniel Franklin, Montserrat Ros, Farzad Safaei, and Mehran Abolhasan. A comparative survey of vanet clustering techniques. *IEEE Communications Surveys & Tutorials*, 19(1):657–681, 2016.

[27] Craig Cooper, Daniel Franklin, Montserrat Ros, Farzad Safaei, and Mehran Abolhasan. A comparative survey of vanet clustering techniques. *IEEE Communications Surveys & Tutorials*, 19(1):657–681, 2017.

[28] Ameneh Daeinabi, Akbar Ghaffar Pour Rahbar, and Ahmad Khademzadeh. Vwca: An efficient clustering algorithm in vehicular ad hoc networks. *Journal of Network and Computer Applications*, 34(1):207–222, 2011.

[29] Efi Dror, Chen Avin, and Zvi Lotker. Fast randomized algorithm for hierarchical clustering in vehicular ad-hoc networks. In *2011 The 10th IFIP Annual Mediterranean Ad Hoc Networking Workshop*, pages 1–8. IEEE, 2011.

[30] Efi Dror, Chen Avin, and Zvi Lotker. Fast randomized algorithm for 2-hops clustering in vehicular ad-hoc networks. *Ad Hoc Networks*, 11(7):2002–2015, 2013.

[31] Stephan Eichler. Performance evaluation of the ieee 802.11 p wave communication standard. In *2007 IEEE 66th Vehicular Technology Conference*, pages 2199–2203. IEEE, 2007.

[32] Ghayet El Mouna Zhioua, Nabil Tabbane, Houda Labiod, and Sami Tabbane. A fuzzy multi-metric qos-balancing gateway selection algorithm in a clustered vanet to lte advanced hybrid cellular network. *IEEE Transactions on Vehicular Technology*, 64(2):804–817, 2015.

[33] A. Ephremides, J. E. Wieselthier, and D. J. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of the IEEE*, 75(1):56–73, 1987.

[34] Peng Fan. Improving broadcasting performance by clustering with stability for inter-vehicle communication. In *2007 IEEE 65th Vehicular Technology Conference-VTC2007-Spring*, pages 2491–2495. IEEE, 2007.

[35] Peng Fan, Prasad Sistla, and Peter Nelson. Theoretical analysis of a directional stability-based clustering algorithm for vanets. In *Proceedings of the fifth ACM international workshop on VehiculAr Inter-NETworking*, pages 80–81, 2008.

[36] Mohammed El Amine Fekair, Abderrahmane Lakas, and Ahmed Korichi. Cbqos-vanet: Cluster-based artificial bee colony algorithm for qos routing protocol in vanet. In *2016 International conference on selected topics in mobile & wireless networking (MoWNeT)*, pages 1–8. IEEE, 2016.

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

[38] Chinmoy Ghorai and Indrajit Banerjee. A novel priority based exigent data diffusion approach for urban vanets. In *Proceedings of the 18th International Conference on Distributed Computing and Networking*, pages 1–10, 2017.

[39] Rituparna Ghosh and Stefano Basagni. Mitigating the impact of node mobility on ad hoc clustering. *Wireless Communications and Mobile Computing*, 8(3):295–308, 2008.

[40] RT Goonewardene, FH Ali, and ELIAS Stipidis. Robust mobility adaptive clustering scheme with support for geographic routing for vehicular ad hoc networks. *IET Intelligent Transport Systems*, 3(2):148–158, 2009.

[41] Sergey Gorinsky, A Kantawala, and J Turner. Feedback modeling in internet congestion control. *Proceedings of the NEW2AN*, 4, 2004.

[42] Forough Goudarzi, Hamid Asgari, and Hamed S Al-Raweshidy. Traffic-aware vanet routing for city environments—a protocol based on ant colony optimization. *IEEE Systems Journal*, (99):1–11, 2018.

[43] Phillip G Gould, Anne B Koehler, J Keith Ord, Ralph D Snyder, Rob J Hyndman, and Farshid Vahid-Araghi. Forecasting time series with multiple seasonal patterns. *European Journal of Operational Research*, 191(1):207–222, 2008.

[44] Jyoti Grover, Nitesh Kumar Prajapati, Vijay Laxmi, and Manoj Singh Gaur. Machine learning approach for multiple misbehavior detection in vanet. In *International Conference on Advances in Computing and Communications*, pages 644–653. Springer, 2011.

[45] Yvonne Guenter, Bernhard Wiegel, and Hans Peter Großmann. Medium access concept for vanets based on clustering. In *2007 IEEE 66th vehicular technology conference*, pages 2189–2193. IEEE, 2007.

[46] Yvonne Gunter, Bernhard Wiegel, and Hans Peter Grossmann. Cluster-based medium access scheme for vanets. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 343–348. IEEE, 2007.

[47] Mohamed Hadded, Rachid Zagrouba, Anis Laouiti, Paul Muhlethaler, and Leila Azouz Saidane. A multi-objective genetic algorithm-based adaptive weighted clustering protocol in vanet. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 994–1002. IEEE, 2015.

[48] Khalid Abdel Hafeez, Alagan Anpalagan, and Lian Zhao. Optimizing the control channel interval of the dsrc for vehicular safety applications. *IEEE Transactions on Vehicular Technology*, 65(5):3377–3388, 2015.

[49] Khalid Abdel Hafeez, Lian Zhao, Zaiyi Liao, and Bobby Ngok-Wah Ma. Impact of mobility on vanets' safety applications. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–5. IEEE, 2010.

[50] A Shalom Hakkert and David Mahalel. Estimating the number of accidents at intersections from a knowledge of the traffic flows on the approaches. *Accident Analysis & Prevention*, 10(1):69–79, 1978.

[51] Jérôme Härri, Fethi Filali, Christian Bonnet, and Marco Fiore. Vanetmobisim: generating realistic mobility patterns for vanets. In *Proceedings of the 3rd international workshop on Vehicular ad hoc networks*, pages 96–97, 2006.

[52] Mohammad Reza Jabbarpour, Rafidah Md Noor, Rashid Hafeez Khokhar, and Chih-Heng Ke. Cross-layer congestion control model for urban vehicular environments. *Journal of Network and Computer Applications*, 44:1–16, 2014.

[53] Daniel Jiang and Luca Delgrossi. Ieee 802.11 p: Towards an international standard for wireless access in vehicular environments. In *VTC Spring 2008-IEEE Vehicular Technology Conference*, pages 2036–2040. IEEE, 2008.

[54] Xin Jin, Weijie Su, and Wei Yan. Quantitative analysis of the vanet connectivity: Theory and application. In *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2011.

[55] John Jubin and Janet D Tornow. The darpa packet radio network protocols. *Proceedings of the IEEE*, 75(1):21–32, 1987.

[56] Mohamed Kafsi, Panos Papadimitratos, Olivier Dousse, Tansu Alpcan, and J-P Hubaux. Vanet connectivity analysis. *arXiv preprint arXiv:0912.5527*, 2009.

[57] Holger Karl and Andreas Willig. *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2007.

[58] Abhay Katiyar, Dinesh Singh, and Rama Shankar Yadav. State-of-the-art approach to clustering protocols in vanet: a survey. *Wireless Networks*, 26(7):5307–5336, 2020.

[59] Omer Kayis and Tankut Acarman. Clustering formation for inter-vehicle communication. In *2007 IEEE Intelligent Transportation Systems Conference*, pages 636–641. IEEE, 2007.

[60] Grace Khayat, Constandinos X Mavromoustakis, George Mastorakis, Jordi Mongay Batalla, Hoda Maalouf, and Evangelos Pallis. Vanet clustering based on weighted trusted cluster head selection. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 623–628. IEEE, 2020.

[61] Takashi Koshimizu, Shi Gengtian, Huan Wang, Zhenni Pan, Jiang Liu, and Shigeru Shimamoto. Multi-dimensional affinity propagation clustering applying a machine learning in 5g-cellular v2x. *IEEE Access*, 8:94560–94574, 2020.

[62] Neeraj Kumar, Naveen Chilamkurti, and Jong Hyuk Park. Alca: agent learning—based clustering algorithm in vehicular ad hoc networks. *Personal and ubiquitous computing*, 17(8):1683–1692, 2013.

[63] Sushil Kumar and Anil Kumar Verma. Position based routing protocols in vanet: A survey. *Wireless Personal Communications*, 83(4):2747–2772, 2015.

[64] Vishal Kumar, Shailendra Mishra, Narottam Chand, et al. Applications of vanets: present & future. *Communications and Network*, 5(01):12, 2013.

[65] Taek Jin Kwon. Energy efficient clustering in ad hoc networks. 2001.

[66] Taek Jin Kwon, Mario Gerla, Vijay K Varma, Melbourne Barton, and T Russell Hsing. Efficient flooding with passive clustering-an overhead-free selective forward mechanism for ad hoc/sensor networks. *Proceedings of the IEEE*, 91(8):1210–1220, 2003.

[67] Peiyuan Lai, Xinhong Wang, Ning Lu, and Fuqiang Liu. A reliable broadcast routing scheme based on mobility prediction for vanet. In *2009 IEEE Intelligent Vehicles Symposium*, pages 1083–1087. IEEE, 2009.

[68] Michael Lee. Clustering technique for vc-vanet. 2018.

[69] Michael Lee. An intersection-based clustering technique for a vehicle crowd-based vanet. 2020.

[70] Michael Lee and Travis Atkison. Vanet applications: Past, present, and future. *Vehicular Communications*, 28:100310, 2021.

[71] Michael Lee, Beichen Yang, and Travis Atkison. 802.11 ac and p in a simulated vanet environment. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3797–3802. IEEE, 2019.

[72] Barry Leiner, Robert Cole, Jon Postel, and David Mills. The darpa internet protocol suite. *IEEE Communications Magazine*, 23(3):29–34, 1985.

[73] Ping Li, Anshumali Shrivastava, and Christian Konig. Training logistic regression and svm on 200gb data using b-bit minwise hashing and comparisons with vowpal wabbit (vw). *arXiv preprint arXiv:1108.3072*, 2011.

[74] Ping Li, Anshumali Shrivastava, Joshua Moore, and Arnd Christian Konig. Hashing algorithms for large-scale learning. *arXiv preprint arXiv:1106.0967*, 2011.

[75] Elnaz Limouchi and Imad Mahgoub. Beflab: Bandwidth efficient fuzzy logic-assisted broadcast for vanet. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

[76] Wei-Yen Lin, Mei-Wen Li, Kun-chan Lan, and Chung-Hsien Hsu. A comparison of 802.11 a and 802.11 p for v-to-i communication: A measurement study. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, pages 559–570. Springer, 2010.

[77] HQ Liu, LC Yang, Yao Zhang, and Lei Wu. A position sensitive clustering algorithm for vanet. *International Journal of Online Engineering*, 10(1), 2014.

[78] K. Liu and K. Niu. A hybrid relay node selection strategy for vanet routing. In *2017 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 1–6. IEEE, 2017.

[79] Suman Malik and Prasant Kumar Sahu. A comparative study on routing protocols for vanets, 2019.

[80] Francisco J Martinez, Chai-Keong Toh, Juan-Carlos Cano, Carlos T Calafate, and Pietro Manzoni. Emergency services in future intelligent transportation systems based on vehicular communication networks. *IEEE Intelligent Transportation Systems Magazine*, 2(2):6–20, 2010.

[81] Hamid Menouar, Massimiliano Lenardi, and Fethi Filali. An intelligent movement-based routing for vanets. In *ITS world congress*, pages 1–8, 2006.

[82] Hamid Menouar, Massimiliano Lenardi, and Fethi Filali. Movement prediction-based routing (mopr) concept for position-based routing in vehicular networks. In *2007 IEEE 66th vehicular technology conference*, pages 2101–2105. IEEE, 2007.

[83] S Almalag Mohammad and C Weigle Michele. Using traffic flow for cluster formation in vehicular ad-hoc networks. In *IEEE Local Computer Network Conference*, pages 631–636. IEEE, 2010.

[84] Garrett Lee Moore. A hybrid (active-passive) vanet clustering technique. 2019.

[85] Rajendra Tushar Moorti, Matthew J Fischer, and George Kondylis. Reduced inter-frame spacing in a wireless transmission system, June 5 2012. US Patent 8,194,626.

[86] Mildred M Caballeros Morales, Choong Seon Hong, and Young-Cheol Bang. An adaptable mobility-aware clustering algorithm in vehicular networks. In *2011 13th Asia-Pacific Network Operations and Management Symposium*, pages 1–6. IEEE, 2011.

[87] Mohammad Mukhtaruzzaman and Mohammed Atiquzzaman. Clustering in vanet: Algorithms and challenges. *arXiv preprint arXiv:2009.01964*, 2020.

[88] Eng Hwee Ong, Jarkko Kneckt, Olli Alanen, Zheng Chang, Toni Huovinen, and Timo Nihtilä. Ieee 802.11 ac: Enhancements for very high throughput wlans. In *2011 IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 849–853. IEEE, 2011.

[89] Charles E Perkins and Elizabeth M Royer. Ad-hoc on-demand distance vector routing. In *Proceedings WMCSA'99. Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100. IEEE, 1999.

[90] Kashif Naseer Qureshi, Abdul Hanan Abdullah, Omprakash Kaiwartya, Fasee Ullah, Saleem Iqbal, and Ayman Altameem. Weighted link quality and forward progress coupled with modified rts/cts for beaconless packet forwarding protocol (b-pfp) in vanets. *Telecommunication Systems*, pages 1–16, 2016.

[91] B Ramakrishnan. Performance analysis of aodv routing protocol in vehicular ad-hoc network service discovery architecture. *network*, 13(14):65–72, 2009.

[92] Sami Abduljabbar Rashid, Mustafa Maad Hamdi, Sameer Alani, et al. An overview on quality of service and data dissemination in vanets. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–5. IEEE, 2020.

[93] Zaydoun Y Rawashdeh and Syed Masud Mahmud. A novel algorithm to form stable clusters in vehicular ad hoc networks on highways. *EURASIP Journal on Wireless Communications and Networking*, 2012(1):15, 2012.

[94] RA Santos. Inter vehicular data exchange between fast moving road traffic using an ad-hoc cluster-based location routing algorithm and 802.11 b direct sequence spread spectrum radio. *Proc. PGNet, Liverpool, UK, 2003*, 2003.

[95] RA Santos, RM Edwards, and A Edwards. Cluster-based location routing algorithm for inter-vehicle communication. In *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, volume 2, pages 914–918. IEEE, 2004.

[96] Miguel Sepulcre, Jens Mittag, Paolo Santi, Hannes Hartenstein, and Javier Gozalvez. Congestion and awareness control in cooperative vehicular systems. *Proceedings of the IEEE*, 99(7):1260–1279, 2011.

[97] Christine Shea, Behnam Hassanabadi, and Shahrokh Valaee. Mobility-based clustering in vanets using affinity propagation. In *GLOBECOM 2009-2009 IEEE Global Telecommunications Conference*, pages 1–6. IEEE, 2009.

[98] Ajit Singh, Mukesh Kumar, Rahul Rishi, and DK Madan. A relative study of manet and vanet: Its applications, broadcasting approaches and challenging issues. In *International Conference on Computer Science and Information Technology*, pages 627–632. Springer, 2011.

[99] Michael Slavik and Imad Mahgoub. Applying machine learning to the design of multi-hop broadcast protocols for vanet. In *2011 7th International Wireless Communications and Mobile Computing Conference*, pages 1742–1747. IEEE, 2011.

[100] Manu Sood and Shivani Kanwar. Clustering in manet and vanet: A survey. In *2014 international conference on circuits, systems, communication and information technology applications (CSCITA)*, pages 375–380. IEEE, 2014.

[101] Evan R Sparks, Ameet Talwalkar, Virginia Smith, Jey Kottalam, Xinghao Pan, Joseph Gonzalez, Michael J Franklin, Michael I Jordan, and Tim Kraska. Mli: An api for distributed machine learning. In *2013 IEEE 13th International Conference on Data Mining*, pages 1187–1192. IEEE, 2013.

[102] John Sucec and Ivan Marsic. Location management handoff overhead in hierarchically organized mobile ad hoc networks. In *Proceedings 16th International Parallel and Distributed Processing Symposium*, pages 10–pp. IEEE, 2001.

[103] Poonam Thakur and Anita Ganpati. A comparative study of cluster-head selection algorithms in vanet. In *Proceedings of International Conference on IoT Inclusive Life (ICIIL 2019), NITTTR Chandigarh, India*, pages 143–157. Springer, 2020.

[104] Daxin Tian, Yunpeng Wang, Guangquan Lu, and Guizhen Yu. A vanets routing algorithm based on euclidean distance clustering. In *2010 2nd International Conference on Future Computer and Communication*, volume 1, pages V1–183. IEEE, 2010.

[105] Chai-Keong Toh. Future application scenarios for manet-based intelligent transportation systems. In *Future generation communication and networking (fgcn 2007)*, volume 2, pages 414–417. IEEE, 2007.

[106] Yasser Toor, Paul Muhlethaler, Anis Laouiti, and Arnaud De La Fortelle. Vehicle ad hoc networks: Applications and related technical issues. *IEEE communications surveys & tutorials*, 10(3):74–88, 2008.

[107] Yu-Chee Tseng, Sze-Yao Ni, Yuh-Shyan Chen, and Jang-Ping Sheu. The broadcast storm problem in a mobile ad hoc network. *Wireless networks*, 8(2-3):153–167, 2002.

[108] Seyhan Ucar, Sinem Coleri Ergen, and Oznur Ozkasap. Multihop-cluster-based ieee 802.11 p and lte hybrid architecture for vanet safety message dissemination. *IEEE Transactions on Vehicular Technology*, 65(4):2621–2636, 2015.

[109] Aravindhan Venkateswaran, Venkatesh Sarangan, Natarajan Gautam, and Raj Acharya. Impact of mobility prediction on the temporal stability of manet clustering algorithms. In *Proceedings of the 2nd ACM international workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks*, pages 144–151, 2005.

[110] Samo Vodopivec, Janez Bešter, and Andrej Kos. A multihoming clustering algorithm for vehicular ad hoc networks. *International Journal of Distributed Sensor Networks*, 10(3):107085, 2014.

[111] Ryuji Wakikawa, Kouji Okada, Rajeev Koodli, and Anders Nilsson. Design of vehicle network: mobile gateway for manet and nemo converged communication. In *Proceedings of the 2nd ACM international workshop on Vehicular ad hoc networks*, pages 81–82. ACM, 2005.

[112] Sheng-Shih Wang and Yi-Shiun Lin. Performance evaluation of passive clustering based techniques for inter-vehicle communications. In *The 19th Annual Wireless and Optical Communications Conference (WOCC 2010)*, pages 1–5. IEEE, 2010.

[113] Sheng-Shih Wang and Yi-Shiun Lin. Passcar: A passive clustering aided routing protocol for vehicular ad hoc networks. *Computer communications*, 36(2):170–179, 2013.

[114] Theodore L Willke, Patcharinee Tientrakool, and Nicholas F Maxemchuk. A survey of inter-vehicle communication protocols and their applications. *IEEE Communications Surveys & Tutorials*, 11(2):3–20, 2009.

[115] Fan Yang, Yuliang Tang, and Lianfen Huang. A multi-channel cooperative clustering-based mac protocol for vanets. In *2014 Wireless Telecommunications Symposium*, pages 1–5. IEEE, 2014.

[116] Yibo Yang, Hongling Li, and Qiong Huang. Mobility management in vanet. In *2013 22nd Wireless and Optical Communication Conference*, pages 298–303. IEEE, 2013.

[117] Yunpeng Zang, Lothar Stibor, Xi Cheng, Hans-Jürgen Reumerman, Arthur Paruzel, and Andre Barroso. Congestion control in wireless networks for vehicular safety applications. In *Proceedings of the 8th European Wireless Conference*, volume 7, page 1, 2007.

[118] Liren Zhang and Hesham El-Sayed. A novel cluster-based protocol for topology discovery in vehicular ad hoc network. *Procedia Computer Science*, 10:525–534, 2012.

[119] Yaping Zhang, Chuanyun Fu, and Liwei Hu. Yellow light dilemma zone researches: a review. *Journal of traffic and transportation engineering (English edition)*, 1(5):338–352, 2014.