2011

A RELIABILITY-BASED ROUTING PROTOCOL FOR VEHICULAR AD-HOC NETWORKS

James Bernsen
University of Kentucky, jrbtex@gmail.com

ABSTRACT OF THESIS

A RELIABILITY-BASED ROUTING PROTOCOL FOR VEHICULAR AD-HOC NETWORKS

Vehicular Ad hoc NETworks (VANETs), an emerging technology, would allow vehicles to form a self-organized network without the aid of a permanent infrastructure. As a prerequisite to communication in VANETs, an efficient route between communicating nodes in the network must be established, and the routing protocol must adapt to the rapidly changing topology of vehicles in motion. This is one of the goals of VANET routing protocols. In this thesis, we present an efficient routing protocol for VANETs, called the Reliable Inter-VEhicular Routing (RIVER) protocol. RIVER utilizes an undirected graph that represents the surrounding street layout where the vertices of the graph are points at which streets curve or intersect, and the graph edges represent the street segments between those vertices. Unlike existing protocols, RIVER performs real-time, active traffic monitoring and uses this data and other data gathered through passive mechanisms to assign a reliability rating to each street edge. The protocol then uses these reliability ratings to select the most reliable route. Control messages are used to identify a node's neighbors, determine the reliability of street edges, and to share street edge reliability information with other nodes.

KEYWORDS: VANETs, vehicular ad-hoc networks, routing protocols for VANETs, vehicle-to-vehicle communication, reliability

Author's signature: James Bernsen

Date: July 4, 2011

A RELIABILITY-BASED ROUTING PROTOCOL FOR VEHICULAR AD-HOC NETWORKS

By
James Bernsen

Director of Thesis: D. Manivannan

Director of Graduate Studies: Raphael Finkel

Date: July 4, 2011

RULES FOR THE USE OF THESES

Unpublished theses submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the usual scholarly acknowledgments.

Extensive copying or publication of the thesis in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this thesis for use by its patrons is expected to secure the signature of each user.

Name	Date

THESIS

James Bernsen

The Graduate School
University of Kentucky
2011

A RELIABILITY-BASED ROUTING PROTOCOL FOR VEHICULAR AD-HOC NETWORKS

THESIS

A thesis submitted in partial
fulfillment of the requirements for
the degree of Master of Science in
the College of Engineering at the
University of Kentucky

By
James Bernsen
Lexington, Kentucky

Director: Dr. D. Manivannan, Associate Professor of Computer Science
Lexington, Kentucky 2011

Dedicated to my lovely wife and daughter

ACKNOWLEDGMENTS

The following thesis, while an individual work, benefited from the insights and direction of several people. My thesis chair, Dr. Manivannan, provided valuable guidance, for which I am grateful. His teaching of the "Distributed and Mobile Computing" special topics course offered my first glimpse into the exciting research area of vehicular ad-hoc networks and challenged me to find a niche in it. The other members of my thesis committee, Dr. Zongming Fei and Dr. Mukesh Singhal, also receive my appreciation for their time and contribution to this effort.

Thanks also to Brad Karp and Francesco Giudici for making source code available for their respective routing protocols, GPSR and STAR. Their code afforded me insight into the inner workings of routing protocols and allowed me to make meaningful performance comparisons between RIVER and related works.

Finally, I owe thanks to my family, friends, and co-workers for their encouragement throughout. Special thanks belong to my wife for her patience and unwavering support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

This thesis proposes Reliable Inter-VEhicular Routing (RIVER), a position-based, greedy, vehicle-to-vehicle routing protocol for Vehicular Ad-hoc Networks (VANETs). VANETs provide the ability for vehicles to spontaneously and wirelessly network with other vehicles nearby for the purposes of providing travelers with new features and applications that have never been previously possible. Within this ever-changing network, messages must be passed from vehicle to vehicle in order to reach their intended destination. A VANET routing protocol must direct these message transfers in an efficient manner to ensure robust data communication.

1.1 What is a VANET?

A VANET is a wireless network that is formed between vehicles on an as-needed basis. To participate in a VANET, vehicles must be equipped with wireless transceivers and computerized control modules that allow them to act as network nodes. Each vehicle's wireless network range may be limited to a few hundred meters, so providing end-to-end communication across a larger distance requires messages to hop through several nodes. Network infrastructure is not required for a VANET to be created, although permanent network nodes may be used in the form of roadside units. These roadside units open up a wide variety of services for vehicular networks such as acting as a drop point for messages on sparsely populated roads, serving up geographically-relevant data, or serving as a gateway to the Internet.

VANETs are a special class of mobile ad hoc networks (MANETs) with their own unique characteristics. Most nodes in a VANET are mobile, but because vehicles are generally constrained to roadways, they have a distinct controlled mobility pattern that is subject to vehicular traffic regulations. In urban areas, gaps between roads are often occupied by buildings and other obstacles to radio communication, so communication along roads is sometimes necessary. Vehicles generally move at higher rates of speed than nodes in other kinds of MANETs. Speeds of vehicles moving in the same direction are generally similar, and they can therefore remain in radio contact with one another for much longer periods of time than with vehicles moving in opposite directions.

Even though several unicast routing protocols have been developed for MANETs [4], [5], [42],[6], [11], [12], [13], [14], [20], [22], [26], [36], [37], [39], [41], [43], [53], [54],

[75], [66], [67], [68], [72], [76], [77], [82], [86], because of the unique characteristics of VANETs, these protocols cannot be directly used in VANETs efficiently. Hence, because of the expected potential impact of VANETs, several researchers have developed unicast routing protocols that are suitable for VANETs. The effect of traffic patterns and congestion on VANETs are studied in [33], [70], [83], and [84]. Security issues in VANETs are discussed in [17], [29], [32], [35], and [71]. Simulation and testing of unicast routing protocols for VANETs presents its own challenges in reproducing realistic vehicular motion on a simulated environment that is representative of real cities and roadways. These simulation issues unique to VANETs are discussed in [8], [23], [62], and [73]. Fairness of bandwidth sharing is discussed in [80].

1.2 Popular MANET Routing Protocols Adapted to VANETS

The Dynamic Source Routing Protocol (DSR) [43] and the Optimized Link State Routing Protocol (OLSR) [78] are well-known unicast routing protocols for MANETs and have been successfully adapted to VANETs as well. DSR is an on-demand routing protocol which searches for a route only when needed. Each node maintains the known routes in its cache. A route consists of the full source route, containing all the intermediate nodes in the route. New routes are discovered by a source by flooding the network with route request message. When the destination receives a route request, it sends a route reply. The route reply sent by the destination accumulates all the nodes through which the route reply propagates. When the route reply reaches the source, it gets the source route to the destination from the reply. On the other hand, OLSR is a proactive routing protocol, which maintains routes between any two nodes in the network. HELLO messages are used for maintaining the routes. The main advantage of this is that each node always has a route to every other node in the network. This advantage comes as a result of large message overhead for maintaining the routes. DSR has low overhead and is suitable for networks in which not all nodes need a route to every other node in the network and the user traffic is low.

1.3 VANET Applications

Numerous applications await users of vehicular ad hoc networks, serving the interests of consumers, businesses, governments, law enforcement agencies, and emergency services. Many of these applications center on the idea of improving the safety of motor vehicles. Accident avoidance warnings could quickly notify drivers of numerous conditions that could cause a collision. When vehicles ahead brake quickly, an audio

warning could alert the driver or an onboard computer might automatically apply the brakes. If vehicles on the road ahead swerve to avoid a road obstruction, the driver may be advised to change lanes. In a scenario in which a driver fails to observe a traffic signal, putting it on a collision course with a cross-street vehicle, both drivers may be alerted or corrective action may be taken by the vehicles' onboard computers. In case an accident does occur, trajectory and velocity information exchanged between vehicles prior to collision may allow the accident to be reconstructed more easily. Rescue vehicles could instantly receive exact coordinates of the location of an accident which can help them to reach the scene of the emergency faster.

Several features that enhance the convenience of drivers could be offered through the use of VANETs. Today, GPS navigation systems can provide detour information based on reported accidents downloaded through cellular carriers for a monthly fee. It may not be possible to obtain such information in real time through cellular carriers. When VANETs are in widespread use, information about traffic and road hazards could be acquired in real-time and fed into vehicle navigation systems to provide alternate driving routes. In such situations, reliability and authenticity of the information disseminated need to be ensured. Existing research results on ensuring reliability and authenticity in the context of MANETS need to be explored and carefully adopted to VANETs. Current electronic toll road payments are typically made via RFID tags installed in vehicles, but there are incompatibilities in the hardware used in different regions. Toll roads could be wirelessly paid without the installation of additional hardware to a vehicle, and protocol standardization [65] could make the technology ubiquitous among various regions. By utilizing the positioning information provided by GPS, geographically-oriented retrieval of local shopping and service information could be provided in real-time, just as this information is preloaded in GPS systems today. Traffic signals equipped with communication equipment could more accurately control intersection traffic, and by feeding their data into centralized systems, similar traffic control efficiencies could be applied at a more macroscopic level such as an entire district or downtown area. Further into the future, VANETs will provide the communication network required by cooperative driving applications, which would allow vehicles to navigate without driver intervention by communicating with other vehicles about velocity, proximity, and other factors.

From radio to television and video players, the possibilities for in-vehicle entertainment have been constantly evolving, and several new applications in this area could be made possible with VANETs. One doesn't yet know what the killer application would be in this area. Internet access for passengers, communication with other

vehicles, multimedia entertainment, and cooperative games are just a few potential entertainment services that VANETs may provide.

1.4 Contribution of the Thesis

This thesis proposes a reliability-based approach to routing in VANETs. RIVER performs real-time traffic monitoring using both active and passive mechanisms to determine the reliability of transmitting a message along a particular route. Active traffic monitoring occurs by sending probe messages along streets to determine whether that street represents a connected edge. Our protocol also detects connected edges by passively monitoring the routes of messages that it receives from adjacent vertices. This reliability information is then shared with other nodes in a distributed manner. The protocol allows routes to be recalculated dynamically during message transmission, taking full advantage of local information at each forwarding node.

RIVER uses these features to address several problems associated with VANETs. Primarily, the protocol attempts to solve the general problem of efficiently discovering and sharing information about network nodes for the purpose of reliably routing messages in a VANET. Its traffic monitoring components and reliability metrics are focused on that issue. A more specific problem of route volatility in VANETs is addressed through the route recovery and recalculation features in RIVER.

1.5 Organization of the Thesis

The remainder of the thesis is organized as follows. We discuss design factors of routing protocols for VANETs and also present an analysis of existing unicast routing protocols for VANETs in Chapter 2 [10]. The RIVER protocol for VANETs is described in Chapter 3. We conclude with a discussion of future work in Chapter 4.

Chapter 2 Related Work

Less than a century since the automobile was made affordable enough for the general public, hundreds of millions of vehicles now travel along highways and streets around the world. Innovations in safety, comfort, and convenience have made vast improvements in automobiles during that time, and now new technologies promise to change the face of vehicular travel once again. One such new technology is the vehicular ad hoc network or VANET, which provides the ability for vehicles to spontaneously and wirelessly network with other vehicles nearby for the purposes of providing travelers with new features and applications that have never been previously possible.

2.1 Design Factors

There are many factors to consider when designing a VANET. Will the network be vehicle-to-vehicle only or could roadside units be used for communication? What forms of communication will be available? Which vehicular systems will be employed in the network? These and many other aspects will require analysis when determining the features and capabilities of a VANET.

2.1.1 Vehicle-to-Roadside Communication

New technologies that will allow vehicles to communicate with roadside units are in progress. The IEEE 802.11 working group continues to actively develop draft amendment 802.11p [2] in order to provide support for Intelligent Transportation System (ITS) applications. In the existing infrastructure and ad-hoc modes of the IEEE 802.11 wireless standard, the time required to authenticate and associate with a basic service set (BSS) is too long to be employed by VANETs. The 802.11p standard provides wireless devices with the ability to perform the short-duration exchanges necessary to communicate between a high-velocity vehicle and a stationary roadside unit. This mode of operation, called WAVE (wireless access in vehicle environments) operates in a 5.9 GHz band and support the Dedicated Short Range Communications (DSRC) standard [81] sponsored by the U.S. Department of Transportation. These standards support systems that communicate from vehicle-to-roadside, vehicle-to-vehicle, or both.

2.1.2 Vehicle-to-Vehicle Communication

Installing fixed infrastructure on roads incurs great expenses, so vehicle-to-vehicle (V2V) communication is necessary to extend the effective range of networked vehicles. VANETs require features not provided by cellular network-based systems such as low data transport times for emergency warnings and robustness due to the network's decentralized structure [56]. In an emergency situation, cellular base stations are often overwhelmed with calls, but distributed communications have the potential for load balancing traffic to avoid network congestion.

2.1.3 Communication Paradigms

Like other kinds of networks, different communication paradigms can be supported in VANETs. *Unicast* communication provides the ability for one node to communicate with a target node in the network. The target node may be in a precise known location or an approximate location within a specified range [85]. While unicast is a useful mode of communication in VANETs, many applications require dissemination of messages to many different nodes in the network.

Multicast communication allows messages to be sent to multiple destinations using the most efficient route possible. For instance, when a traffic jam occurs at a particular location on a roadway, it would be valuable to send messages to vehicles that are approaching that point so that they can take alternate routes. Sending a multicast message that reaches vehicles not affected by the traffic jam would waste valuable network bandwidth. Instead, it is desirable to only target affected vehicles, whose positions can be determined by analyzing a road map. If each vehicle is equipped with knowledge of its own global coordinates, then a specialized form of multicast communication called *geocast* is possible. In geocasting, a message is sent to all of the nodes in a particular geographic position, usually relative to the source of the message. A similar form of communication is *anycast*, where a node sends a message to any destination node in a group of nodes. Anycast provides a data acquisition feature that is the intuitive inverse of geocasting, where a node sends a message to a certain geographic area to request data from any node found there, called geographical anycast. Geographic anycasting has the ability to provide "distributed floating car data" [31]. Another communication paradigm called *scan* operates like a sonar echo, sending a message that traverses a certain region once [85]. Much research [3] [9] [44] [48] [49] [85] has focused on providing these various multicast services in VANETs.

2.1.4 Environmental Constraints

VANETs operate in a very different environment than most computing applications. The high velocities at which vehicles move sometimes reduces the amount of time available for message exchanges. Protocols need to take advantage of vehicles moving in the same direction at relatively similar speeds to maintain connections for longer periods of time [64]. Protocols must operate well in both city roads and highways. City streets pose a unique set of geographic constraints as buildings between streets often form obstacles for radio signals that must be routed around. Road characteristics such as traffic signals and stop signs affect the flow of traffic in urban areas, breaking any reliable streams of similar-velocity vehicles that may be found on highways. On highways, low vehicle density must be considered. Traffic density, often measured in the number of vehicles per unit distance, has a large influence on road capacity and vehicle velocity. In low traffic densities, vehicles tend to move at faster rates, but as traffic density increases, vehicles slow down. Very high traffic density (in the case of a serious road block, for example) also causes both relative speed and distance between vehicles to become stable [64].

2.2 Timeline and Classification

In this section, we present a timeline of unicast routing protocols for VANETs and a qualitative comparison of them. For the reader's reference, publication dates for each protocol surveyed were used to build the timeline shown in Figure 2.1. In addition, where a paper cited another protocol as a reference point for its own improvements, Figure 2.1 captures this dependency with an arrow from the predecessor protocol to the potentially influenced protocol.

In Figure 2.2, we give a qualitative comparison of the existing VANET unicast routing protocols. We have classified VANET unicast routing protocols based on three sets of criteria: objectives, characteristics, and assumptions. In the set of objectives criteria, we categorize based on the design goals of each protocol. Some protocols are aimed at providing *vehicle-to-vehicle* services, while others focus on vehicle-to-roadside communication. Other protocols intend to provide communication in *delay tolerant/sparse* networks. In contrast, there are *QoS* (Quality of Service) oriented protocols, some of which provide *Internet connectivity* to vehicles.

In the set of characteristics criteria, we categorize based on the various strategies used by each protocol to achieve its objectives. Nearly all of the protocols are *position-based*, using knowledge of vehicles' positions, and velocities to route mes-

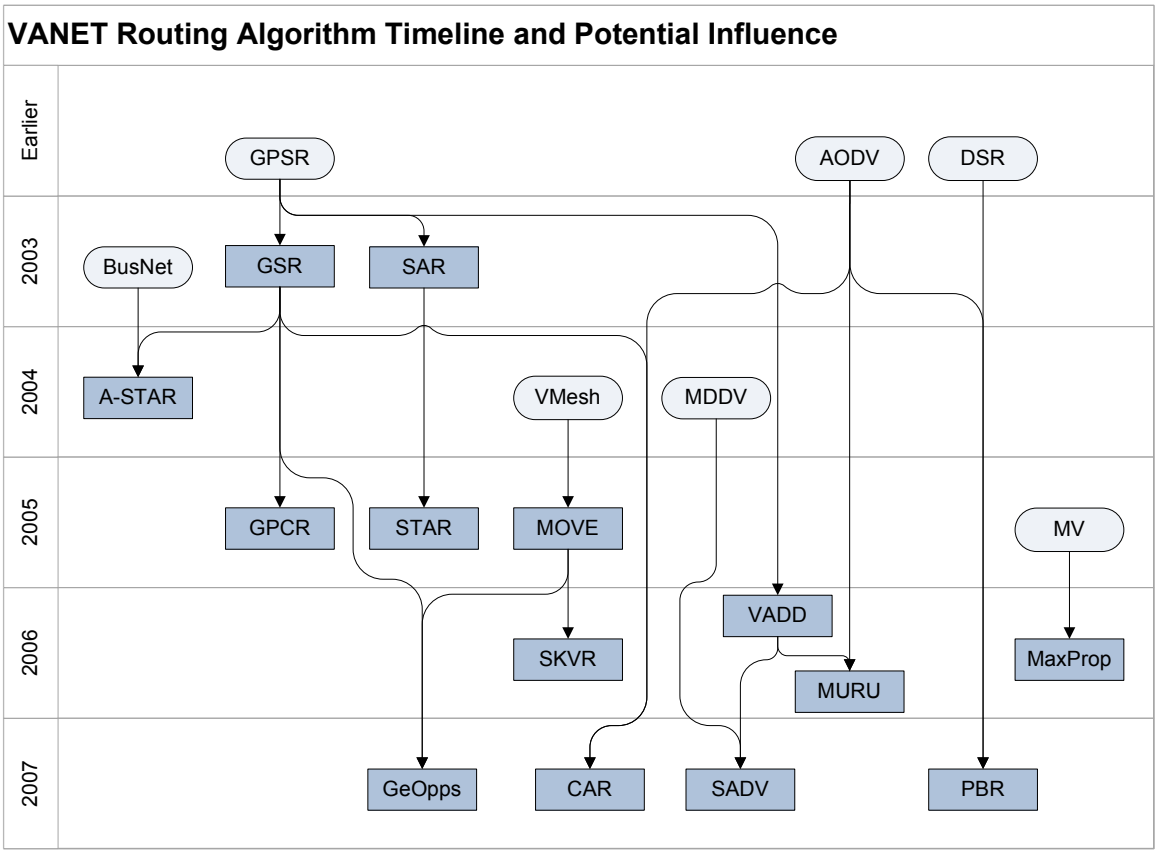


Figure 2.1: VANET Unicast Routing Protocols Timeline and Potential Influence

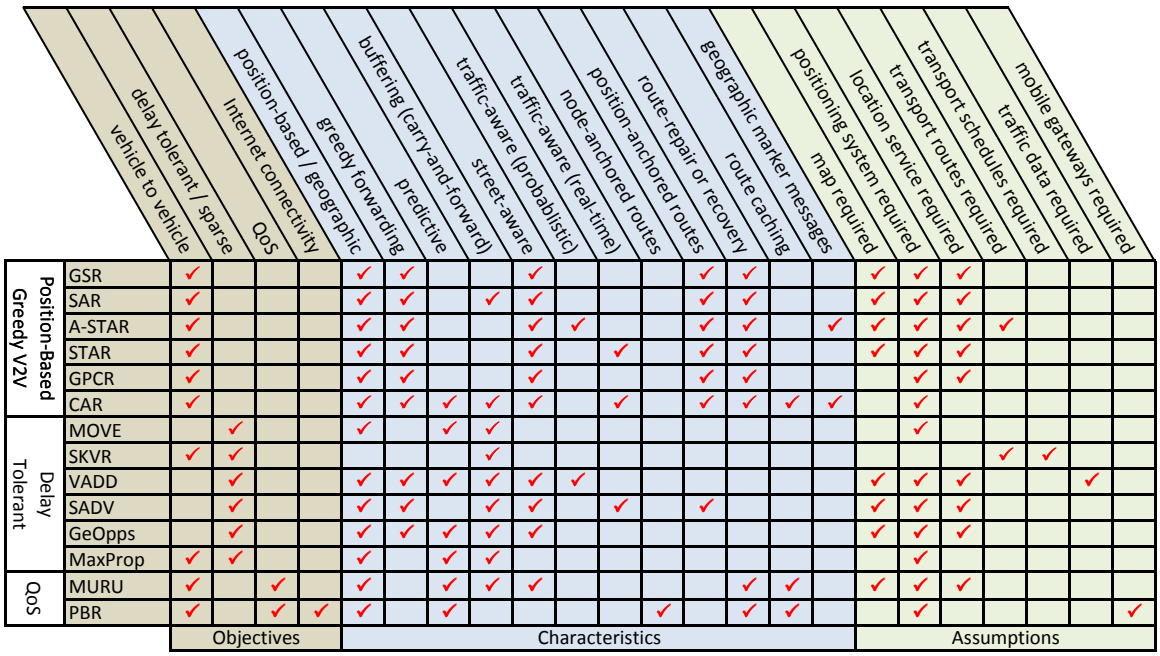


Figure 2.2: Qualitative Comparison of VANET Unicast Routing Protocols

sages. Many also utilize the *greedy forwarding* strategy for sending messages to the farthest neighbor in the intended direction. We also observe several *predictive* approaches where some speculation is made about characteristics of the nodes involved in a route. Some algorithms make predictions on the current locations of nodes based on the last known position, and velocity of the node. Other algorithms use this same information to make predictions about the stability or estimated lifetime of a route. To provide higher rates of delivery in sparse networks, a *buffering (carry-and-forward)* strategy is often used. In this strategy, a node may hold a packet in a local buffer until a forwarding opportunity is available, instead of simply dropping the packet.

We adopt the term *street-aware* to imply that the protocol builds route paths along streets, whether an external map is used to perform this function or not (this is noted independently in the assumptions criteria). We use a similar term, traffic-aware, to refer to a protocol's ability to utilize traffic information to select an efficient route. We divide the traffic-aware algorithms into two groups: 1) *traffic-aware (probabilistic)*, which includes those protocols that make probabilistic assumptions about traffic density by using static knowledge such as bus routes and lane information, and 2) *traffic-aware (real-time)*, which includes those that determine traffic density by real-time measurement. In establishing routes, we can categorize the unicast routing protocols based on whether hops on the route are *node-anchored* (by remembering the exact vehicles used in a route) or *position-anchored* (remembering approximate geographic positions used in the route).

The criterion *route-repair or recovery* refers to protocols which either use a strategy to recover from a greedy local optimum in a position-based route or have a mechanism for repairing broken routes. We also denote *route caching* protocols which remember routes that have been previously established. We observe that a small number of protocols include the use of what we call *geographic marker messages*. These are messages that are passed from one vehicle to another to proliferate previously discovered information about a geographic location. They may indicate a location that is disconnected [74] or they may provide forwarding information for a relocated node like the "guards" in [63].

In the final set of criteria, assumptions, we identify the external sources of information that each protocol depends upon for its operation. These sources of information include external maps, positioning systems (such as GPS), and location services. Some protocols take advantage of public transport routes and schedules to approximate the positions of those vehicles. Traffic data from external sources may be utilized, and one protocol [61] requires the presence of "external gateways", which

are vehicles that have a wireless WAN connection to the Internet.

2.3 Position-based Greedy V2V Protocols

A significant number of unicast routing protocols use a position-based, greedy approach to provide vehicle-to-vehicle communication. Position-based approaches use information about the geographic coordinates or relative positions of nodes to generate an efficient route through the network. Like most VANET unicast routing approaches, position-based routing allows for unicast communication, but it also allows for delivery of messages to all nodes in a geographic area, called geocasting.

When feasible, position based routing is beneficial in that no global route from a source node to a destination node needs to be created and maintained: each node decides where to forward each packet it receives. It has been shown that routing approaches using position information can adapt to the high mobility of nodes found in highway situations. [33]

In the greedy forwarding strategy, an intermediate node in a route forwards messages to the farthest neighbor in the direction of the next anchor or the destination. This approach requires the intermediate node to have three important data points: the position of itself, the position of its neighbors, and the position of the destination. Typically, a node's own position is acquired through GPS. (All of the protocols in this category assume a positioning system of some kind.) The node's neighbors' positions are obtained through message exchanges, and the position of the destination node is usually found through the use of a location service [57]. Location servers may be periodically placed external to the system, but this offers no guarantee that such a server is within range. To alleviate this problem, quorum-based location services may be built into nodes, or fully-distributed location services may be utilized [59].

2.3.1 Geographic Source Routing

The geographic source routing (GSR) algorithm [56] tries to overcome the disadvantages of position-based routing approaches designed for MANETs when applied to VANETs in urban scenarios. For example, consider the position-based MANET algorithm GPSR [45]. GPSR utilizes the strategy of greedy forwarding of messages toward a known destination. If, at any hop, there are no nodes in the direction of the destination, then GPSR has a recovery strategy called perimeter mode that routes around this void. The perimeter mode has two components. The first component is a distributed planarization algorithm that makes a local conversion of the connectivity

graph into a planar graph by eliminating redundant edges. The second component is an online routing algorithm that operates on planar graphs.

In urban areas, buildings and other radio obstructions restrict effective network routes to run along streets, so in VANET scenarios, the perimeter mode of GPSR would be frequently required. In GPSR, if a radio obstruction or void causes the algorithm to enter perimeter mode, then the planar graph routing algorithm begins operating. While attempting to route messages around the obstruction, the planarized connectivity graph causes messages to be sent to immediate neighbors instead of sending messages to the farthest node on the street in the perimeter route. Because this routing method uses immediate neighbors instead of the farthest reachable nodes, more nodes carry the message, causing increased delays and greater hop counts. In addition, with the rapid movement of vehicles inherent to VANETs, routing loops can be introduced while in the perimeter mode of GPSR, causing further inefficiencies. Even worse, these routing loops can cause messages to be transmitted in the wrong direction by persistently following the right hand rule. For example, consider five nodes a, b, c, d, and e arranged clockwise in a semicircular pattern. Node a attempts to send a message to e, but an obstruction blocks the path. Perimeter mode is entered, following the right-hand rule, and the message travels $a \rightarrow b \rightarrow c \rightarrow d$, but during this sequence, d moves within range of b. When link (d, b) forms, b becomes the next destination by the right-hand rule, and a loop is formed.

Geographic source routing [56] uses a map of the urban area to avoid these problems. Using a static street map and location information about each node, GSR computes a route to a destination by forwarding messages along streets. The sender of a message computes a sequence of intersections that must be traversed in order to reach the destination. This sequence of intersections can be placed in the packet header or they can be decided by each forwarding node. The path between the source and destination is computed using Dijkstra's shortest-path algorithm.

GSR is a seminal algorithm for position-based VANET unicast routing. However, the vehicle density on a chosen street is not considered, and the authors acknowledge that there is potential to improve the algorithm by considering this information. The algorithm also requires externally-provided static street maps for operation.

2.3.2 Spatially Aware Packet Routing

Similar to GSR, the Spatially-Aware Packet Routing (SAR) protocol [79], attempts to overcome some of the weaknesses of the recovery strategy used by GPSR. In particular, when GPSR reaches a local optimum at a network void, its stateless recovery

strategy causes each packet reaching the void to engage in its recovery strategy as well. If the network void is a permanent one, then the frequent use of the recovery strategy degrades the algorithm's performance. SAR is a position-based unicast routing protocol that predicts and avoids route recovery caused by permanent network voids. SAR relies upon the extraction of a static street map from an external service such as GIS (Geographic Information Systems) to construct a "spatial model" for unicast routing.

In SAR, a node determines its location on the "spatial model" and uses the street information to calculate a shortest path to a packet's destination. When this path is determined, the set of geographic locations to be traveled is embedded into the header of the packet. Note that no intermediate nodes are included in the source route; it is position-anchored: a route based on immobile physical locations. When a node needs to forward a packet, it inspects the packet header for the next geographic location in the route. Rather than utilizing a strictly greedy strategy toward the destination, a neighbor that is located along the route listed in the packet is chosen.

When a forwarding node cannot find a node along the predetermined routing path in SAR [79] it is suggested that one of the following strategies could be employed. First, the node can choose to place the packet in a "suspension buffer" where it remains until a suitable node is located along the routing path. Second, the node can attempt to greedily forward a packet towards its destination. Finally, the node may recompute the unicast routing path stored in the packet header. However, none of the suggested recovery strategies are recommended, nor is it specified how the node might select one of these strategies. The authors compared SAR with no recovery strategy and SAR using the suspension buffer approach (SARB) in their simulations and noted that SARB provides a higher packet delivery ratio at the expense of message delay in the presence of sparse networks.

2.3.3 Anchor-Based Street and Traffic Aware Routing

Unlike other greedy position-based unicast routing protocols, Anchor-Based Street and Traffic Aware Routing (A-STAR) [74] utilizes city bus routes as a strategy to find routes with a high probability for delivery. A-STAR also aims to remedy the problem where the perimeter mode of GPSR utilizes next-neighbor hops along a street instead of selecting the farthest neighbor along a street for a next hop. In addition, the authors of A-STAR note that the use of the right-hand rule at a local optimum is inefficiently biased toward one direction while going in the opposite di-

rection (following a left-hand rule instead) sometimes results in a shorter route to a destination.

The A-STAR algorithm uses "anchor-based" unicast routing, which involves inserting a sequence of geographic forwarding points into a packet, through which the packet must travel on its route to the destination. Like GSR, A-STAR also utilizes a static street map to route messages around potential radio obstacles such as city buildings. In order to take advantage of the fact that some streets contain denser traffic than others, the A-STAR algorithm also utilizes bus route information. The assumption here is that buses typically follow major streets, and major streets are more likely to contain vehicle traffic in greater densities. A weight inversely proportional to the number of bus routes that serve that street is assigned to each street. This information is used to compute an anchor path using Dijkstra's least-weight path algorithm.

When reaching a local optimum, the recovery mode of A-STAR works differently than that of its predecessors. In recovery mode, A-STAR computes a new route along an anchor path from the node at which the local optimum was reached and rewrites the path stored in the packet header. To prevent other packets from reaching the same local optimum, A-STAR uses geographic marker messages. The street that led to the local optimum is temporarily marked as "out of service" by piggy-backing this information along with the message that reached the local optimum. Any node that receives a message that contains the "out of service" information updates its local map so that "out of service" routes are not used for computing anchor paths. These routes remain inoperative until a timeout duration threshold is met.

In their simulations of A-STAR, the authors assume that buses are equipped to act as communicating nodes; a noteworthy assumption since this would insure that a city would have additional vehicular nodes traveling along regular paths. The addition of bus schedules into the algorithm is noted as potential future work. The authors observe that estimating the traffic density of a street based on bus route information is less optimal than more dynamic approaches that utilize latest traffic condition information, and they leave the details of this for future work also.

2.3.4 Spatial and Traffic Aware Routing

In designing the Spatial and Traffic Aware Routing (STAR) algorithm [34], the drawbacks of the SAR algorithm were observed. The SAR algorithm had the advantage of its underlying spatial model, allowing it to forward packets along streets. However, SAR did not have knowledge of whether any vehicles were actually positioned along

the streets it selected. The STAR algorithm is designed to overcome this problem by forwarding packets only along streets that are occupied by vehicles. Under STAR, the source node computes its route more lazily than SAR by providing a partial route in the packet header and then relying on intermediate nodes to determine additional segments of the route. This provides the advantages of a fixed packet header length and a route that can be adjusted dynamically to accommodate accurate local traffic information and the movement of the destination.

The traffic monitoring component of STAR is interested in determining two primary conditions: streets that contain dense traffic and streets that contain sparse or non-existent traffic. Beacon messages are utilized to observe "node neighborhoods" and this information is maintained in a neighbors-table stored at each node. The neighbors-table contains the position of each neighboring node. The neighbors-table and two dependent data structures called the "presence vector" and the "persistence vector" are used to determine sparse and dense traffic conditions relative to each node. The presence vector counts the number of neighboring nodes in each of the cardinal directions, and when a value in this vector reaches a low or high threshold, this event causes an update to the persistence vector. The persistence vector is used to capture sparse and dense traffic situations that have persisted for a substantial amount of time while ignoring temporary abnormal conditions. When the persistence vector shows that traffic is significantly sparse or dense in any direction for a considerable period of time, this observation is recorded in the node's traffic-table. Each node periodically sends a beacon message that includes its ID, its position, and information from its traffic-table.

The routing and forwarding component of STAR computes routes on-demand, utilizing information from the node's traffic-table. Each node dynamically maintains a weighted graph of street map and traffic information. Streets with dense traffic are represented by edges with lesser weights, making them preferred for routing. When a source node s wishes to send a message to a destination node d, it computes part of the route toward d using its weighted graph. When the end of that route is reached, the node where the route ends computes another route toward d, stores this in the packet header, and continues forwarding the packet. If a route fails (due to sparse traffic or other reasons), it is handled as if the end of the route was reached, and the node at which the route failed computes a new route to the destination from its weighted graph.

The STAR algorithm's reliance on beacon messages may introduce scalability and wasted bandwidth problems since there appear to be no heuristics for adapting the

beacon to conditions such as high node density or network congestion. Furthermore, the authors show that the size of beacon messages in STAR are approximately six times the size of beacons in GPSR and SAR. It is also unclear how STAR maintains its presence vector when moving at high speed in one direction for an extended period of time, such as on a freeway. For example, if a node is moving east at a high velocity, it appears that over time the presence vector may become skewed such that it appears that traffic is sparse to the east and dense to the west. It is true that a node's traffic table is updated via beacon message exchanges in STAR, but a node moving at high velocity has less time to exchange large messages, thus exacerbating the possibility of a skewed presence vector.

2.3.5 Greedy Perimeter Coordinator Routing

Like GSR, the Greedy Perimeter Coordinator Routing (GPCR) algorithm [57] takes advantage of the fact that city streets form a "natural planar graph". This algorithm improves upon GSR by eliminating the requirement of an external static street map for its operation. GPCR consists of two components: a restricted greedy forwarding procedure and a repair strategy for routing along streets towards a destination when the greedy strategy reaches a local optimum. Both algorithms avoid graph planarization by utilizing the innately planar graph of city streets.

To avoid potential radio obstacles such as buildings, in GPCR, the typical destination-based greedy forwarding strategy is modified such that it only routes messages along streets. In this way, routing decisions are only made at street intersections. As such, the goal is to forward messages to nodes at an intersection, rather than forwarding them to a node that is already past the intersection. (Contrast this to the traditional greedy strategy that always forwards to the farthest node in the direction of the destination.) However, since this algorithm does not require an external static street map, there is a problem in determining which nodes are located at intersections. GPCR solves this problem by defining two heuristic methods for determining whether nodes are in an intersection, and it designates those nodes as "coordinators". A coordinator has the responsibility of making routing decisions, and it periodically broadcasts its role along with its position information. Nodes can determine whether they are coordinators in one of the following two ways.

The first approach for coordinator determination is known as the neighbor table approach. In this approach, nodes periodically transmit beacon messages which contain their position information and the last-known position information of all of their neighbors. By listening to these beacon messages, a node has information about its

own position, the position of its neighbors, and the positions of its neighbors' neighbors. Using this information, a node x considers itself to be within an intersection if it has two neighbors y and z that are within transmission range of each other but neither of them lists the other as its neighbor. Such a situation implies that y and z are separated by an obstacle and that node x can forward messages around this obstacle.

The second approach (the correlation coefficient approach) for coordinator determination does not use beacon messages. Instead, each node uses its position information and the position information of its immediate neighbors to calculate the correlation coefficient, ρ_{xy}, of its position with respect to its neighbors. A correlation coefficient indicates the strength and direction of a linear relationship between two independent variables. A strong linear correlation between the positions of the node's neighbors (i.e., ρ_{xy} is close to 1) indicates that the node is on a street. If there is no linear correlation among the positions of the node's neighbors (i.e., ρ_{xy} is close to 0), this indicates that the node is within an intersection. Through adjustment of a threshold ε, a node can evaluate the correlation coefficient and assume that $\rho_{xy} \geq \varepsilon$ indicates the node is on a street and $\rho_{xy} < \varepsilon$ indicates the node is at an intersection. Performance studies show that this correlation coefficient approach performs better than the neighbor table approach, therefore GPCR utilizes this strategy solely.

When a forwarding node that is not a coordinator forwards a message from its predecessor, it considers only those neighbors whose paths are approximate extensions of the line between the forwarding node's predecessor and the forwarding node itself. If none of the considered nodes are coordinators, then the farthest of these nodes is chosen as the next hop. If however, some of the considered nodes are coordinators, then one of the coordinators is chosen at random to be the next hop. Since GPCR avoids radio obstacles by only sending messages along streets, once a packet reaches a coordinator, a routing decision can be made. The decision about which street should be followed is performed using a greedy strategy once again. The street occupied by the neighboring node that is closest to the destination is selected.

By using the correlation coefficient approach for coordinator determination, the algorithm can avoid any dependency on an external street map. However, despite the improvements of using calculated street information for the greedy strategy, scenarios still occur where the greedy strategy reaches a dead-end in a local optimum. This is where the repair strategy is exercised. Like the forwarding procedure, the repair strategy makes routing decisions at coordinator nodes in intersections, but it continues to use greedy forwarding between coordinators (to eliminate the extra hops noted in

GPSR above). To determine which street to follow at an intersection, the repair strategy uses the well-known right-hand rule. Using this rule, the coordinator node chooses the street that is the next one counter-clockwise from the street that the packet arrived on. Since streets do not move, the problems that plagued GPSR in VANET scenarios (creating routing loops and sending packets in the wrong direction) are avoided. When a node is reached whose distance to the destination is less than the distance from the destination to the node where the repair strategy began, the greedy strategy is resumed.

The authors acknowledge that, like GPSR, GPCR does not account for low node density on selected streets, and they plan to augment the algorithm with "a very low overhead proactive probing scheme" for determining whether a selected street allows another intersection to be reached.

2.3.6 Connectivity-Aware Routing Protocol

Like other position-based vehicular routing protocols, the Connectivity-Aware Routing (CAR) protocol [63] finds a route to a destination, but a unique characteristic of CAR is its ability to maintain a cache of successful routes between various source and destination pairs. This characteristic was prompted by observations of other position-based routing protocols and their inability to utilize information gathered about disconnected paths (due to unoccupied streets, for instance) after those disconnections are detected. CAR also predicts positions of destination vehicles, repairs routes as those positions change, and employs geographic marker messages.

Nodes using the CAR protocol send periodic HELLO beacons that contain their "velocity vector" information (heading and speed). Upon receiving a HELLO beacon, a node records the sender in its neighbor table and calculates its own velocity vector and the velocity vectors of its neighbors. Entries expire from the neighbor table when distance between nodes exceeds a threshold or after two HELLO intervals. The beaconing interval adapts to traffic density by increasing in frequency when there are few neighbors (to maintain contact in sparse traffic) and decreasing in frequency when neighbors are abundant (to reduce unnecessary message overhead). Beacons can also be piggybacked on forwarded data packets. These measures help to reduce wasted bandwidth and network congestion.

The CAR protocol establishes the notion of a "guard", a geographic marker message that is buffered and passed from one vehicle to another to proliferate forwarding information about a node that has moved to a new location. A guard is a temporary message that has an ID, a TTL (time-to-live) counter, a radius, and some state

information (usually about the velocity vector of a node, but other types of information could be stored). CAR provides two different forms of guards: the "standing guard" and the "traveling guard". A standing guard is tied to specific geographic coordinates, while a traveling guard has initial coordinates, initial time, and a velocity vector. A guard, which is kept alive by nodes in its geographic area, is transmitted as part of the nodes' HELLO beacons, and is stored in a Guards-Table by any node that receives such a beacon. To maintain traveling guards, nodes use the guard's initial coordinates, time, and velocity vector to compute the guard's current geographic area. A node that is aware of a guard can use the information to ensure delivery of messages to destination nodes that have moved.

Velocity vectors are considered parallel if the smallest angle between them is less than a predetermined value and non-parallel otherwise. Since CAR does not use external maps, this information is useful to determine when a message travels around a curve or changes course in an intersection. When an intermediate node's velocity vector is not parallel to the previous message-bearer's velocity vector, an "anchor" is added to the path discovery (PD) message to indicate that the message changes direction in this location. The anchor contains the coordinates and velocity vectors of the current and previous nodes. When the PD message reaches the destination, it contains a series of anchors to mark the path from the source.

After receiving one or more PD messages, the destination chooses a path with high connectivity and low delay. It then sends a route reply (RREP) back to the source node, containing its coordinates, its velocity vector, and information from the original PD message. Advanced Greedy Forwarding (AGF) is used to send the RREP back to the source. When it reaches the source, the route is recorded by the source node, and communication begins. Communication over the established path also utilizes a slightly modified AGF strategy. Instead of always choosing the farthest neighbor in the direction of the destination, the farthest neighbor in the direction of the next anchor point is chosen.

To maintain the established routing paths as the vehicles in the network change positions, guards are utilized to avoid the repetition of the costly discovery phase. If a node at a route endpoint changes its direction, then the node activates a guard that contains its old and new velocity vectors. Depending on the values of the old and new velocity vectors, the guard may be a standing guard that remains at the point of direction change or a traveling guard that moves in the direction of the old velocity vector. If a node that is aware of the guard receives a message addressed to the relocated node, it adds the guard coordinates as an anchor point to the message,

calculates a new estimated position for the destination, and forwards the message. The endpoint also sends a notification message to the other endpoint of the route, to inform that node of the new anchor points along the route. Notification messages may also be sent if either endpoint passes an anchor point, crosses the established routing path, or if the endpoint's change in velocity may cause a substantial position shift between its estimated and actual positions.

Routing errors may occur due to communication gaps between anchor points or due to guards that were not maintained due to low traffic density. When routing errors occur, CAR has two recovery strategies to cope with the problem. The first recovery strategy is called "timeout algorithm with active waiting cycle". In this mode, the forwarding node places the packet in a suspension buffer and periodically checks to see if a potential next-hop neighbor has entered into range. To do this, it sends a "non-propagating next-hop request" to inform other nodes that it has detected a disconnection and tries to find a next-hop node. Upon receiving this request, which contains the forwarder's coordinates and the next anchor point in the route, a node may identify itself as being between those locations and reply with a HELLO beacon. The second recovery strategy is called "walk around error recovery". In this mode, the forwarding node informs the source node of the disconnection and initiates a local path discovery process. The node also initiates a guard to declare that it is a buffering node, and it receives and buffers all messages on this route. If the local path discovery fails, this information is sent back to the source node, the buffered messages are either dropped or returned to the source, and the source starts a new path discovery. However, if the buffering node's path discovery was successful, then the new path is sent to the source node and the buffered messages are delivered.

The CAR protocol's ability to generate a virtual infrastructure in the form of guards give it a distinct advantage over other protocols. It provides street awareness, performs basic traffic awareness during path discovery, maintains routes, adapts well in low traffic densities, and it does not require map or location services.

2.4 Delay Tolerant Protocols

In urban daytime scenarios where vehicles are densely packed, locating a node to carry a message is not typically a problem. But in rural highway situations or even in cities at night, fewer vehicles are running, and establishing end-to-end routes may not be possible. Even in densely-populated urban scenarios, sparse sub-networks can be prevalent. Law enforcement, military, and financial armored vehicles may each wish

to exchange data privately within their own vehicular networks due to the sensitivity of the information exchanged. One existing application is observed in the VMesh Demand-Response project [21] where utility pricing information is exchanged from roaming utility vehicles for utility usage information from consumer homes.

In such cases, some consideration needs to be given to routing in sparse networks. In particular, in the early stages of vehicular networks, when few vehicles are equipped with wireless transceivers, networks will frequently be sparse. In these situations, delay-tolerant routing algorithms are needed. Delay-tolerant networks are sometimes also known as disruption-tolerant networks.

2.4.1 Motion Vector Routing Algorithm

In routing a message from a vehicle to a roadside unit in a fixed location, the Motion Vector (MOVE) routing algorithm [51] uses knowledge of neighboring vehicles' velocities and trajectories to predict which vehicle will physically travel closest to the destination. This algorithm assumes a sparser network where rare opportunistic routing decisions must be made predictively.

The MOVE algorithm is an algorithm for specialized sparse VANET scenarios. In these scenarios, vehicles act as mobile routers that have intermittent connectivity with other vehicles or stations in their network. The network is so sparsely populated that there is seldom, if ever, a completely connected path from source to the static destination (a fixed data collection point or roadside unit). A carry-and-forward approach is used for vehicles to store data for long periods between connections. Connection opportunities must be scrutinized carefully since they occur infrequently and the global topology is unknown and rapidly changing. At each opportunity, the algorithm must predict whether forwarding a message at that instant provides progress toward the intended destination.

Under the MOVE algorithm, it is assumed that each node has knowledge of its own position and heading, and it is assumed that the message destination D is a fixed location that is known globally. From that information, the current node C can calculate d_C, the closest distance between the vehicle and the message destination, D, along its current trajectory. The current node periodically sends a HELLO beacon. If a neighbor vehicle N in the network receives the beacon, it replies with a RESPONSE message to make itself known. Given the direction that the neighbor vehicle N is heading, the current vehicle C determines the shortest distance d_N to the destination D along the neighbor vehicle's trajectory. The current vehicle then makes a forwarding decision based on this information, each vehicle's current distance from

the destination, and rules about each vehicle's trajectory relative to each other and to the destination. (One example rule: if the current vehicle is moving away from the destination and the neighbor vehicle is moving toward it, forward the message.)

In sparse networks, data delivery rates were higher for the MOVE algorithm than a greedy, position-based routing algorithm. The MOVE algorithm also used less system-wide buffer space. However, the authors noted that, in a preliminary performance evaluation where vehicle routes were consistent and uniform (bus routes), a greedy, position-based routing algorithm performed better than MOVE. Further work is planned in this area to take advantage of knowledge of the bus routes and schedules to optimize the algorithm's performance. The MOVE algorithm is designed for sparse networks and for vehicles that transfer data from sensor networks to a base station.

2.4.2 Scalable Knowledge-Based Routing

Observing that knowledge-based schemes such as MOVE can improve the effectiveness of a vehicular network, the Scalable Knowledge-Based Routing (SKVR) algorithm [1] utilizes the relatively predictable nature of public transport routes and schedules. In particular, bus routes are used. Position knowledge of each bus is not required. SKVR assumes that bus routes are not looped routes; they have a start and an end between which they travel forward and then backward.

SKVR divides the network into domains such that each bus route is a separate domain. The algorithm works on two hierarchical levels: the top level is inter-domain routing where a source and destination are on different bus routes, while the bottom level consists of intra-domain routing within a bus route. Even the most dependable public transport system is not completely predictable, as vehicles are prone to delays due to traffic, mechanical failures, driver habits, road conditions, or other factors that may cause them to deviate from their routes and schedules. Because of this, SKVR makes use of two types of knowledge: static knowledge that does not change over time and dynamic knowledge that changes often. For inter-domain routing (among bus routes), SKVR relies on static knowledge only, namely the bus routes themselves and their intersections, which are fixed and do not change often. For intra-domain routing (when the source and destination are in the same bus route), SKVR uses both static and dynamic knowledge. The dynamic knowledge used in intra-domain routing is the schedule of the bus time tables, which may often be prone to variations due to the aforementioned circumstances.

For intra-domain routing, when a source node wants to send a message to a destination node that is within the same physical route, there are only two directions in which to send the message, forward or backward. To determine the destination node's position without location information, a simple method is used: each vehicle maintains a list of other vehicles it has encountered that are also traveling along the same route, clearing the list when it reaches the end of the route. Each vehicle also transmits its direction (a single Boolean value) along the route as it travels. If the destination is in this previous-contact list, then it must be in the backward direction, and so the message's direction flag is marked accordingly. When vehicles along the same route encounter one another, a node carrying a message must decide whether to continue buffering the message or to forward it based on the direction information of the other vehicle. In this way, messages are passed to vehicles traveling along the route in the direction of their direction flag until the destination is reached. Eventually, each vehicle along the current route reaches the end of the route, at which time it continues back down the route in the reverse direction. This ensures that even when nodes are temporarily disconnected, their buffered messages will eventually be delivered when they cross paths with another vehicle in their route.

For inter-domain routing, SKVR forwards a message to a vehicle traveling in the destination domain, or to a vehicle that will eventually make contact with that domain. When the destination's domain is reached, the intra-domain behavior can complete the delivery. If the sending vehicle's contact list does not contain any vehicle in the destination's domain, then copies of the message are sent to other vehicles in the contact list.

2.4.3 Vehicle-Assisted Data Delivery

The Vehicle-Assisted Data Delivery (VADD) protocol [87] uses a "carry-and-forward" strategy to allow packets to be carried by vehicles in sparse networks for eventual forwarding when another appropriate node enters the broadcast range, thereby allowing packets to be forwarded by relay in case of sparse networks. VADD requires each vehicle to know its own position and also requires an external static street map that includes traffic statistics.

With the VADD protocol, when wireless channels are not available, the carry-and-forward strategy is used by transferring packets along vehicles on the fastest roads available. Since vehicles may deviate from predicted paths, the routing path should be recomputed continuously during the forwarding process. To aid in this process, VADD uses a street graph weighted with expected packet delivery delays.

Each packet has three modes: Intersection, StraightWay, and Destination, where each mode is based on the location of the node carrying the packet. Appropriately, the *Intersection* mode is used when the packet has reached an intersection, at which routing decisions can be made for the packet to be forwarded to a vehicle along any of the available directions of the intersection. In *StraightWay* mode, the current node is on a road, and there are only two possible directions for the packet to travel: in the direction of the current node or in the opposite direction. Both directions are considered using a simplified version of the algorithm found in Intersection mode, and when the optimal direction is chosen, packet forwarding occurs in a greedy manner. *Destination* mode is briefly entered when the packet is close to its final destination.

The most complex mode is the Intersection mode because of the many decisions that are made while in this mode. Within an intersection, the forwarding node prioritizes the available streets (based on distance to destination, expected packet delivery delay, and probability of packet delivery) and chooses the next intersection in the highest prioritized direction as the "target intersection". It then attempts to select a candidate node for that intersection. If no candidate node can be found for the current target intersection, the forwarding node chooses the next best direction, chooses a target intersection in that direction, and resumes the strategy of finding a candidate node for that intersection. If no nodes are found in any suitable direction, the forwarding node carries the packet.

When choosing a candidate node to which a message will be forwarded, there are several variations of VADD with different selection criteria. Location First Probe VADD (L-VADD) selects nodes that are located closely to the target intersection, regardless of the direction the node is traveling. In this way, L-VADD is similar to the position-based greedy strategy: choose the farthest node in the routing direction. Because the selection of the next hop node is based solely on proximity to the target intersection, as nodes approach an intersection, L-VADD has the potential for causing routing loops. For instance, consider node A that selects north as the highest prioritized direction for its packet. There are no nodes to the north of A, and east is chosen as the next best direction, so as A approaches an intersection to the east, it may forward its packet to another node B that is southeast of A and closer to the eastern intersection. However, when B receives the packet, it determines that north is the best direction, and A is further north than B, so it sends the packet back to A. These simple loops can be avoided by recording the previous hop and not returning a packet to that node, but this method is not scalable for larger loops, and results in making many valid forwarding paths unusable in order to prevent loops.

Another variation of the protocol, Direction First Probe VADD (D-VADD), prefers nodes that are moving in the same direction as the highest priority route direction. This strategy eliminates the possibility of routing loops because all nodes agree that a carrying node should always be a vehicle traveling in the direction with highest priority. However, due to the possibility that a node may not encounter the most optimal transfer node immediately upon entering an intersection, the authors introduced Multi-Path Direction First Probe VADD (MD-VADD), which is a multi-path variant of D-VADD. MD-VADD is more likely to find the most optimal routes, but it also wastes bandwidth by duplicating packets among multiple receivers.

Hybrid Probe VADD (H-VADD) is a hybrid protocol that harnesses the strengths of L-VADD, D-VADD and MD-VADD. Upon entering an intersection, H-VADD operates in the greedy manner of L-VADD. However, if a routing loop is detected, D-VADD or MD-VADD is used in the current intersection instead. The source paper is not clear on how D-VADD or MD-VADD is selected in the presence of a routing loop. In simulations, H-VADD shows the best performance in terms of delivery ratio, message delay, and packet loss.

VADD makes an interesting assumption that vehicles with GPS capability are generally programmed with long-range trajectory information (for example, to map driving directions to a destination for the driver). VADD proposes that this trajectory be sent along with a packet so that when a return packet is sent, the intermediate nodes can continually update the predicted location of the packet's destination. The authors leave the details for this idea as future work.

2.4.4 Static-Node Assisted Adaptive Routing Protocol

The Static-Node Assisted Adaptive Routing Protocol in Vehicular Networks (SADV) [28] aims at reducing message delivery delay in sparse networks. In experiments using VADD, the authors observed that the network became unstable as vehicle density decreased because optimal paths were not available and because VADD relies upon probabilistic traffic density information. To remedy this, the authors proposed the idea that static nodes could be placed at intersections to assist with packet delivery. Each static node has the capability to store a message until it can forward the message to a node traveling on the optimal path. SADV also dynamically adapts to varying traffic densities by allowing each node to measure the amount of time for message delivery. SADV assumes that each vehicle knows its own position through GPS and that each vehicle has access to an external static street map.

SADV has three different modules: "Static Node Assisted Routing" (SNAR), "Link Delay Update" (LDU), and "Multi-Path Data Dissemination" (MPDD). SNAR attempts to deliver messages on a path with the shortest expected delay to the destination, using both vehicular nodes and static nodes. It operates in two modes: the "in-road mode" and the "intersection mode" (similar to VADD's StraightWay and Intersection modes). The in-road mode operates while a vehicular node carries a message in a road. In this mode, greedy geographic forwarding is used to transport a message toward a static node at an intersection. When the static node is reached, the intersection mode assumes control. In intersection mode, the static node calculates the optimal next intersection for the message based on its "delay matrix", explained below. The packet is stored at the static node until it can be forwarded to a vehicle traveling toward the optimal next intersection. If multiple vehicles traveling in that direction are immediately available, the static node forwards the message to the farthest of those vehicles.

As with any store-and-forward communication, buffer management is an important issue in SADV. When the buffer at a static node becomes full, packets are eliminated from the buffer by forwarding them along the best available path. The strategy used for determining which messages are eliminated from the buffer is called "least delay increase", which attempts to send packets along paths which will not significantly increase their delivery delay. Under this strategy, the static node checks which paths are currently available, and eliminates packets that will be least affected by traveling along the available paths.

As described above, SNAR makes extensive use of optimal paths. Optimal paths are determined based on a graph abstracted from a static road map and weighted with "expected path forwarding delays" from a delay matrix. SADV generates these expected path forwarding delays based on real-time traffic density information. The LDU module maintains the delay matrix dynamically by measuring the delay of message delivery between static nodes. When a static node s_i receives a message, it places a timestamp in the message header and proceeds to forward the message. When that message reaches the next static node s_j, the elapsed time is calculated, and the timestamp in the message header is updated again. The elapsed time is retained by s_j, and in this way, a static node s_j can obtain the delay for packets arriving from all incoming directions. Each static node calculates the mean delay for each incoming direction over a specified time interval and propagates this information to neighboring static nodes in a periodic "delay update message" distributed by vehicles.

When traffic is low, the MPDD module of SADV takes advantage of the situation

by utilizing multi-path routing. To reduce the impact on the protocol stack of vehicles, only static nodes may deliver messages on multiple paths. In the MPDD module, each static node chooses the best two paths for each message and attempts to forward the message along both of those paths. To avoid excessive routing overhead caused by loops, each static node maintains a record of the messages it has sent during a time interval and ignores any duplicate packets received during that time.

The simulations used in the performance analysis did not include any one-way streets. Since one-way streets are not effective for sending data in the opposite direction of traffic flow, a scenario that includes one-way traffic may have an impact on the performance results of SADV.

2.4.5 Geographical Opportunistic Routing

Like other routing algorithms designed for delay-tolerant networks, the Geographical Opportunistic (GeOpps) routing algorithm [52] uses opportunistic routing and a carry-and-forward approach to route messages in sparse networks. Its uniqueness derives from its utilization of navigation information to route packets efficiently. GeOpps assumes that each vehicle has knowledge of its position through GPS and a navigation system that provides a suggested route to a traveling destination. It is also assumed that each vehicle's navigation system can provide the location of static roadside units.

When a message is being sent from a source node to a destination using GeOpps, intermediate nodes use the following method to select the next hop. Each neighbor vehicle that is following a navigation-suggested route calculates its future nearest point to the message destination. It also uses a "utility function" built into its navigation system to calculate the amount of time required to reach that point. The vehicle that can deliver the packet fastest or closest to the destination is chosen as the next hop for the message.

GeOpps also considers some special cases. If drivers cease to follow their navigation-suggested routes, then the algorithm forwards the message to any neighbor. If a vehicle remains stopped for a certain amount of time, or if the engine is switched off, then all messages carried by the vehicle are forwarded to the next available vehicle. The algorithm does not require all vehicles to be equipped with navigation systems. In these situations, a greedy algorithm is used to route messages instead.

GeOpps performs well in sparse networks and with delay-tolerant applications. The differences in navigation systems that are available to consumers may create challenges for GeOpps. For instance, there may be some difficulties in the differences

between "utility functions" for time calculations, depending on the factors (speed limit, traffic conditions, road types, average driver speed) that each navigation system considers.

2.4.6 MaxProp

Another algorithm in the delay-tolerant network category, MaxProp [18] utilizes carry-and-forward and packet prioritization techniques to maximize message delivery in a network with limited transfer opportunities between nodes. MaxProp is implemented in a real network called UMass DieselNet where it is deployed on buses, allowing each bus to communicate its location and performance information to wireless access points or other buses as they are encountered. The algorithm is an extension of a previous routing protocol called MV [19], and it assumes that each node in the network has knowledge of its position through GPS.

MaxProp operates in three basic stages: the *neighbor discovery* stage in which nodes discover each other before packet transfers can begin, the *data transfer* stage in which a limited amount of data can be transferred between nodes, and the *storage management* stage in which each node manages its local storage buffer by selecting packets to delete according to a prioritization algorithm.

By default, a node carries a message until the next neighbor discovery phase occurs. The node continues to forward this message at each node meeting until the message's timeout is met, the delivery of the message is confirmed by an ACK in a data transfer stage, or until the message is dropped in the storage management stage due to a full buffer.

In the data transfer stage, messages are transferred based on a highest-priority-first scheme. In the storage management stage, MaxProp deletes messages based on a lowest-priority-first scheme. Each message stored at a node is prioritized based on the estimated cost to deliver that message to its destination. To determine the estimated delivery cost, each node in MaxProp maintains a probability of meeting every other node in the network. MaxProp uses incremental averaging to modify these probabilities at each meeting between nodes, and at each meeting, these probabilities are exchanged between the nodes. With these values, each node calculates the estimated cost to deliver a message for each possible path to that message's destination. This is done by summing the probabilities that each connection on the path will not occur. The estimated cost for that message to reach its destination is the least costly path among all of those calculated.

When the data transfer stage begins, information exchange occurs in a predetermined order. First, messages destined for the neighbor node are transferred. Secondly, the aforementioned meeting probability information is exchanged between the nodes. Third, delivery acknowledgements are exchanged. This allows delivered messages remaining in the node's buffer to be safely deleted. Fourth, messages with low hop count are transferred, regardless of estimated delivery cost. Finally, the remaining messages that have not yet been transmitted are sent according to their priority, based on estimated delivery cost.

Prioritizing based on two different factors (delivery cost and hop count) is done in the following manner. The priority-sorted buffer at each node is divided into two groups according to whether messages in the buffer have a hop count of less than a threshold t hops. Messages with a hop count below this dynamically set threshold t are sorted by their hop count (newer messages have higher priority), while messages with larger hop counts are prioritized by the aforementioned estimated delivery cost criterion.

The storage management stage handles the maintenance of each node's finite buffer. Messages are deleted when the buffer is full. First, messages that have been acknowledged as delivered are deleted. These acknowledgements are distributed across the entire network (not just to the source) to clean up message copies that might otherwise remain in each node's buffers for an extended period of time. Secondly, messages that have a hop count of greater than t hops are deleted in the order of the highest estimated delivery cost first. Finally, if the buffer is still full, messages with hop counts below t are deleted in the order of the largest hop counts first.

MaxProp offers novel ideas about message prioritization, buffer management, and routing messages based on probability of node meetings. System-wide acknowledgements and tracking the probability of meeting every other node in the network are practical techniques in a network of 30 buses, but these approaches may not scale well to larger populations.

2.5 Quality of Service Protocols

Using the term Quality of Service (QoS) to describe current protocols in this category is a trifle misleading. In the strictest sense, a QoS protocol should provide guarantees about the level of performance provided. This is often achieved through resource reservation and sufficient infrastructure. However, in an ad-hoc wireless network, this is a difficult task. With the exception of the potential for roadside units, there

is no infrastructure to be relied upon for guaranteed bandwidth. The dynamic and cooperative nature of an ad-hoc network does not lend itself to resource reservation. What variables can we adjust in order to provide guarantees about service?

These QoS routing strategies attempt to provide a robust route among vehicles. Factors such as link delay, node velocity and trajectory, node position, distance between nodes, and reliability of links all contribute to the stability of a particular route. Some performance guarantees can be made in vehicular routing, and we survey the algorithms that can estimate the duration for which a route will remain connected, and minimize the amount of time required to rebuild the connection if it is broken. With those exceptions, the current suite of QoS VANET routing protocols is most aptly described as a set of "best effort" protocols.

2.5.1 Multi-Hop Routing Protocol for Urban VANETs

Merging position-based and QoS factors, the Multi-Hop Routing Protocol for Urban VANETs (MURU) [60] balances hop minimization with the ability to provide a robust route connection. In doing so, a new metric called the "expected disconnection degree" (EDD), is introduced to estimate the quality of a route based on factors such as vehicle position, speed, and trajectory. MURU requires each vehicle to know its own position and to have an external static street map available. The presence of an efficient location service is also assumed.

EDD, the expected disconnection degree, is an estimation of the probability that a given route might break during a given time period. Using this measure, a low EDD is preferred. Given knowledge of vehicle positions, speeds, and trajectories, one can make some guesses about the stability of a route along a sequence of nodes. Intuitively, nodes moving in similar directions at similar speeds are more likely to maintain a stable route. The authors also show that, given certain assumptions about vehicle traffic, routes with very small and very large routes have higher packet error rates. The formula for calculating EDD takes these factors into account to make a prediction about the breakability of a route. In MURU, each RREQ message stores the cumulative EDD for the path that message traveled.

To find a route to a destination, a source node calculates the shortest trajectory to the destination, based on their locations and the static street map. It then initiates a RREQ message, broadcasting it in a rectangular "broadcast area" that encloses the shortest trajectory and is bounded by the positions of the source and destination. The shortest trajectory is stored in the packet and is used as a directional guideline for the RREQ message. Nodes outside of the "broadcast area" will drop the packet.

A node n_i, upon receiving a RREQ message from node n_{i-1}, calculates the EDD of the link from node n_{i-1} to node n_i using the formula

$$EDD_{i-1,i} = \alpha * |D(i-1,i) - D_0|^l + \beta * f(L(i), T(0,d)) + \gamma * g(M(i-1), M(i), T(0,d)).$$

Here, α, β and γ are predefined tuning parameters, l is the path loss exponent which is determined by the signal propagation model used in the urban areas. $D_{i-1,i}$ is the geographic distance from node n_{i-1} to node n_i, D_0 being a correction factor. $L(i)$ is the current location of node n_i, $M(i)$ is n_i's predicted movement information including expected velocity during the time period T. $T_{0,d}$ is the shortest trajectory from source n_0 to destination n_d. $g(M(i-1), M(i), T(0,d))$ returns 0 if n_i and n_{i-1} are expected to be within the transmission range of each other for a period longer than T, and 1 otherwise. $f(L(i), T(0,d))$ returns 0 if n_i is on the shortest trajectory towards the destination for a time period longer than T, and 1 otherwise. RREQ message has a field to carry $EDD_{path}(0,i)$ which is the EDD of the path from source n_0 to the current node n_i through which the RREQ has propagated. When RREQ is received by node n_i from node n_{i-1}, it calculates $EDD_{path}(0,i)$ using the following formula and includes it in the RREQ before rebroadcasting it.

$$EDD_{path}(0,i) = \begin{cases} 0 & \text{if } i = 0 \\ EDD_{path}(0,i-1) + EDD_{path}(i-1,i) & \text{otherwise} \end{cases}$$

If every node receiving the RREQ message immediately re-broadcast it, the message overhead would quickly become not scalable. To avoid this, the MURU protocol provides a pruning mechanism for route requests which allows each node to delay forwarding a RREQ message. A node receiving the RREQ waits for a calculated backoff delay (milliseconds) that is directly proportional to the EDD between the previous forwarder of the RREQ and the current node. (If the EDD is large, then the current node's forwarding delay is also large to allow for the possibility that a path with a smaller EDD may be found.) During this backoff delay, the node listens for RREQ messages at other nodes. If during that window of time, the node overhears a counterpart to this RREQ (from the same source with the same sequence number) whose EDD is smaller than its own EDD to the source, then it drops the RREQ. (This causes RREQ messages received along the path to be dropped by nodes whose links are more likely to be broken.) If the node has not dropped the RREQ during this time, then it rebroadcasts the RREQ into a rectangular area defined by its position and the position of the destination. Thus, each time the RREQ is re-broadcast, the broadcast area becomes an iteratively smaller rectangular area. When the destination

finally receives some RREQ messages from different routes, it selects the route with the smallest EDD.

In the paper, the authors show that MURU is loop-free and that MURU always chooses a path from source to destination with the smallest EDD, both initially and when repairing a route. Due to the rectangular broadcast areas used in MURU, it may be susceptible to local optimums. If the rectangular broadcast area is strictly bounded by the forwarding node and the destination node, this problem could occur in cases where the only available next hop node is located outside of the broadcast rectangle. MURU aims to provide a quality route that delivers a high percentage of packets while controlling overhead and delay.

2.5.2 Prediction-Based Routing

While many algorithms have concentrated on vehicle-to-vehicle communication in VANETs, the Prediction-Based Routing (PBR) algorithm [61] is focused on providing Internet connectivity to vehicles. Building an infrastructure of roadside static gateways is costly, especially outside of urban areas. Instead, PBR explores the possibility of mobile gateways with wireless WAN connections that can act as Internet gateways for other vehicles, focusing specifically on highway scenarios. The PBR algorithm assumes that each vehicle has knowledge of its own position through GPS or other means. The algorithm takes advantage of the less erratic vehicle movement patterns on highways to predict the duration and expiration of a route from a client vehicle to a mobile gateway vehicle. Just before a route failure is predicted, PBR preemptively seeks a new route to avoid loss of service.

To communicate to a location on the Internet, a node checks its routing table for an existing route. Like many reactive MANET protocols, if the node finds no existing route, the node broadcasts a RREQ message with a limited number of hops. When the RREQ reaches a mobile gateway, a RREP message is returned to the source node through the sequence of nodes stored in the RREQ. (Note that due to the low lifetime of vehicular routes, intermediate nodes with cached gateway routes do not send RREP messages.) If multiple gateways are found, the source node chooses the route with the shortest number of hops where the most hops are moving in the same direction as the source node. If multiple routes to the same gateway are found, then the route with the longest predicted lifetime is chosen. Once the route to the gateway is established, communication begins. If this route remains active over time, then just before the route is predicted to fail, PBR attempts to establish a new route to a gateway.

The RREP message is used to predict the lifetime of the route. When the gateway receives the RREQ message, it writes its position and velocity information and a maximum lifetime value in the RREP message that it returns to the source node. As each intermediate node handles the RREP message it estimates the lifetime of the link between its predecessor and itself, based on wireless communication range, direction of travel, velocities of nodes, and distance between the nodes. If that lifetime is less than the lifetime stored in the RREP message, then it updates the message with the lesser lifetime. When the source node receives the RREP message, it contains the estimated lifetime of the route.

Regardless of the effectiveness of the PBR algorithm in the mobile gateway situation, it is unknown how realistic the situation itself is. It is unclear how a vehicle would be motivated to share its wireless Internet connection with others when that connection is likely to be costly, although the integration of micropayments may be a possible area for exploration. The prospect of Internet providers charging for the use of their roaming WAN-connected vehicles may be economically feasible, but further analysis would be required to determine this. In addition, the bandwidths of the mobile gateway's wireless WAN connections would need to be significant enough to support the bandwidth demand of numerous client vehicles.

Chapter 3 RIVER: Reliable Inter-VEhicular Routing

Reliable Inter-VEhicular Routing (RIVER) is a position-based, greedy, vehicle-to-vehicle routing protocol for vehicular ad-hoc networks. This protocol prefers transmitting messages using routes along paths it deems to be reliable through its traffic monitoring components. This traffic monitoring happens in real-time by actively sending probe messages along streets and by passively monitoring messages that are transmitted between adjacent intersections. Furthermore, RIVER takes traffic monitoring a step further by propagating reliability information within the network without the use of broadcast, network flooding, or other means that have been shown to cause network congestion. Instead, street reliability data is distributed in a more localized manner by piggybacking the information on routing messages, probes, and beacons. To take full advantage of this localized information throughout the life of a routing message that may travel outside of its sender's limited zone of knowledge, our protocol allows routes to be recalculated dynamically at any anchor point during message transmission. This route recalculation is also used as part of the protocol's route recovery mechanism when no neighboring nodes can be found along the current route.

Based on our classifications in Chapter 2, our protocol would fit best in the position-based, greedy, vehicle-to-vehicle category. Like its peers in that group, RIVER is a geographic protocol that identifies neighboring nodes with beacon messages, has street-awareness, utilizes position-based routes, and forwards greedily. A node can identify neighboring nodes and their locations via beacon messages. RIVER uses these coordinates to choose appropriate forwarding nodes for the purpose of transmitting a message toward its current anchor point. Our protocol is street-aware in the sense that it attempts to route messages through vehicles along streets. VANET protocols typically route along streets because this is where vehicles are likely to be located. In urban areas, buildings located alongside streets can often block radio transmissions. Street-awareness requires a basic knowledge of the physical location of streets and their intersections, which RIVER acquires from static pre-loaded data. Based on this street-awareness, the protocol generates routes that are anchored at specific geographic positions, typically street intersections, as opposed to defining route hops using transient network nodes in motion. Although the protocol is not strictly greedy from source to destination, it greedily forwards a message between each of the anchor points it sets in its routes.

However, RIVER also differs in some significant ways from its peers. While the seminal VANET protocols such as GPSR [56] and SAR [79] did not take traffic factors into account at all, A-STAR [74] utilized static traffic information from bus schedules. The creators of A-STAR hypothesized that buses travel on major thoroughfares that are more likely to have dense vehicular traffic. A-STAR was therefore programmed to prefer these roads for forwarding. The real-time, active traffic monitoring component of the RIVER protocol allows it to avoid routes along streets containing gaps in node coverage that would prevent message transmission.

A fundamental mechanism of its active traffic monitoring component is also a unique aspect of RIVER: probe messages. Unlike a unicast message, a probe message is sent to an unknown network node along a particular street edge to determine the connectivity of that particular street edge. Some VANET protocols have made limited use of real-time traffic information within their protocols. STAR [34] monitors the number of nodes it encounters in each of the cardinal and intercardinal directions relative to each node to aid in routing decisions. Each node in CAR [63] adapts its beaconing interval to the number of neighboring nodes it has detected so that beacons do not saturate network bandwidth in dense traffic conditions. SADV [28] measures message delivery delays to estimate traffic densities. RIVER's probe messages differentiate it from its peers because of the real-time feedback they provide.

Also singular to our protocol is its method for using the traffic monitoring information to calculate the reliability of a path. This reliability information is based on first-hand observation by each node and by second-hand information that is distributed between nodes. The reliability data allows RIVER to route messages around network voids that a simpler shortest-path algorithm cannot detect. Finally, while the protocol has a route recovery mechanism like other VANET routing protocols, its dynamic route recalculation may prevent the need for recovery prior to a route failure. We will expand on these differentiating aspects of the RIVER protocol in Section 3.2, Section 3.3, and Section 3.4.

3.1 Network Stack and Topology

As a VANET routing protocol, RIVER depends on lower layers of the network stack to manage physical network hardware, establish data links with other nodes, identify nodes by unique network addresses, manage packet hop counts, etc. Likewise, the protocol provides a VANET routing service to higher network layers while maintaining a loose coupling with those layers.

In Figure 3.1 a typical (TCP/IP) RIVER network stack is depicted. At its left are the corresponding layers of the Internet Protocol Suite [15] [16]. At its right are the corresponding layers of the OSI reference model [40]. Since the overwhelming majority of network protocols rely on the Internet protocol, we make these same assumptions with our protocol, and an IP layer is assumed beneath it. RIVER also may optionally utilize a connection to the link layer for a dropped-link detection feature and to provide eavesdropping capabilities for passive monitoring of network traffic. These features will be described in detail in Section 3.1.1, Section 3.2.5, and Section 3.4. Note that Figure 3.1 depicts this direct connection between the RIVER layer and the link layer, which includes the Media Access Control protocol (MAC) [38] in this diagram.

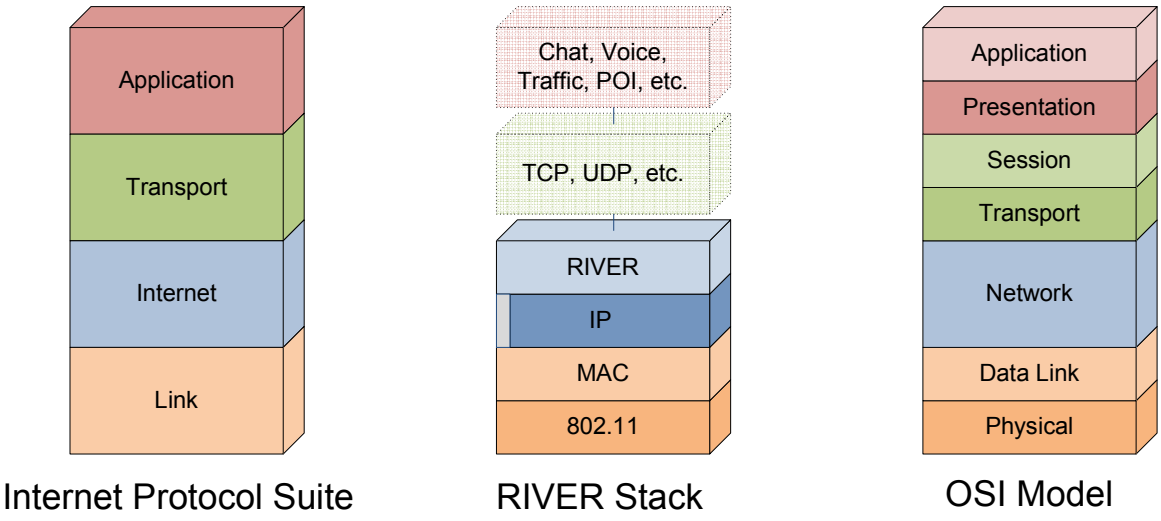


Figure 3.1: Typical RIVER Network Stack

The network topology of a VANET lacks a great deal of determinism when compared to a physical network such as the Internet. Internet nodes are connected to one another through intermediate network devices, and they have a fixed network topology. The intermediate network devices can route messages efficiently because each network node can be reached through a path that is assigned to a particular port on that device. Typically, a hierarchical addressing and subnetworking scheme (as is used in Internet protocol) aids in the routing process. However, nodes in a VANET do not have these same means of routing messages to one another. Network nodes are in motion, so their unique addresses do not follow any hierarchical pattern based on their physical location or the network to which they belong. Their connection to the network is maintained wirelessly, and because of their frequent movement, each

node's neighbors are changing as the nodes move in and out of each others' radio ranges.

For example, in a physical network, a message might be routed from source S to destination D by forwarding through network nodes N_x as follows:

$(S \rightarrow N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow D)$

Unless some node N_x in that path is physically removed from the network, that path can be reliably repeated to send a message to node D in the future.

However, in a VANET, the network topology is highly dynamic. A source node S cannot reliably assume that any previous neighbor node N_x will still be within wireless range when S sends its next message. Each node in the network must establish a list of its neighbors and maintain that list as those neighbor nodes move out of range and as new nodes become neighbors by moving within range. This is achieved through a mechanism called beaconing. Furthermore, each node N_x in the path from S to D is also likely to be in motion, so node S cannot guarantee these same nodes will continue to form a wirelessly connected path to node D. Because of this, nodes do not define their routes by specifying intermediate node addresses. Instead, anchor points are coupled to geographical locations between the sender and receiver.

3.1.1 Beaconing

Like several other VANET protocols, RIVER uses a beaconing system for identifying neighboring nodes. Each node N_x intermittently broadcasts a one-hop beacon message to other nodes within its transmission range. At a minimum, the beacon message contains the sending node's geographical location, obtained via GPS. By doing this, nodes receiving the beacon message can identify node N as a neighboring node within their receiving range, and they can associate node N with a last-known geographical location. This allows a sending node S to determine whether node N is physically located along a path to a given destination node D. The information about node N is recorded in a data structure commonly known as a **neighbors-table**. Beacons are sent at a semi-regular interval, but as in GPSR [56], it is randomly adjusted with a "jitter" duration between beacon intervals in order to help prevent collision of beacons from different nodes if their beacon intervals became synchronized.

The neighbors-table must be updated as nodes move out of range of each other. In our protocol, this function is accomplished with a predefined timeout value and by information received from the link layer of the network. The timeout is called the **dead-neighbor-timeout**. If node N_x has not received a message from one of its neighbor nodes within the number of seconds defined in the dead-neighbor-timeout,

that neighbor node is assumed to be out of range and is deleted from the neighbors-table. There may be cases where a node N_x removes node N_y from its neighbors-table because it has not received a beacon from that node, but the node N_y is still within range of N_x and simply has not broadcast a beacon (or that beacon was not received for some other reason). If a beacon is subsequently received from node N_y, it will be reinserted into the neighbors-table of N_x. Conversely, it is possible for a node N_x to contain an entry for N_y in its neighbors-table, but node N_y has, in fact, moved out of range. In this case, if N_x should attempt to send a message to N_y, the link layer (for 802.11 networks, this occurs at the MAC sublayer [38]) would be unable to establish a link. When this occurs, the RIVER protocol intercepts this MAC layer information, directs node N_x to remove node N_y from its neighbors-table, and then directs N_x to choose another neighbor node in order to retry the transmission.

In addition to sending explicit beacon messages, the RIVER protocol's other kinds of messages may also be used as **implicit beacons** by setting an implicit beacon bit in the packet. When this is enabled, each forwarding node N_x for the message modifies the packet's **beacon origin geolocation** and **beacon origin address** blocks with their own information. All nodes that overhear the message over the wireless network may treat it as a beacon for the forwarding node N_x. Another unique aspect of the beaconing scheme in our protocol is that beacons also carry a timestamp and a **known-edge list** for distributing reliability information within the network. Wireless eavesdropping and the known-edge list are discussed further in Section 3.2.3. For a full diagram of RIVER's explicit beacon packet, see Section A.1.

3.1.2 Street Graph

Because vehicles are generally restricted to roadways, network voids are very likely to occur unless communication is similarly constrained along these roadways. In urban areas, gaps between roads are often occupied by buildings and other obstacles to radio communication, so establishing routes along roads is made even more necessary. To avoid these problems, RIVER employs a method that originated with GPSR [56] and SAR [79] and other VANET protocols have also utilized: a geographic map of streets and their intersections. Due to the fact that roadways change infrequently, the presence of a street map becomes a convenient substitute for a network topology.

For our protocol's street graph implementation, each street is represented as a series of geographical coordinates of locations where the road intersects with another roadway or where the road curves. These geographical coordinates represent the vertices of the graph, and the edges between these vertices represent (relatively)

straight roadways between those coordinates. The graph is an undirected graph since communication can take place in both directions along roadways (even on one-way streets). The street graph, then, is defined as $G = (V, E)$ where G is an undirected, connected graph of street vertices V and their respective street edges E. A vertex $v \in V$ is defined $v = \{i, x, y\}$ where i represents its internal identifier i and (x, y) represents its geographical coordinates. An edge $e \in E$ is a set $e = \{u, v, w\}$, where $u, v \in V$ and $u \neq v$ and w is the weight of the edge, as used in Dijkstra's shortest-path algorithm.

3.1.3 Practical Considerations

To meet these requirements, a simple source of external data that provides two elements is needed. First, the geographical coordinates for the vertices of the street graph are needed. Secondly, an indication of how these vertices are connected to form street edges is also required. The existence of map data is generally considered to be a reasonable assumption in the literature since modern vehicles commonly have on-board navigation systems. Map data can also be extracted from available Geographic Information Systems (GIS) sources.

Within a heterogeneous real-world environment, it would be impractical to assume that all vehicular nodes would have the same map data source available. To allow for this possibility, the vertices and edges in a RIVER street graph do not require any common labels or naming conventions among nodes, nor do they require any identification beyond their geographical coordinates since nodes communicate with each other simply by describing their geographical positions; the internal identifiers of vertices are not communicated. Edges are also devoid of names or labels, as they are identified by the vertices on which they are incident.

In addition, to accommodate any potential imperfections that may be present in the position data of different map data sources, the coordinates of a particular vertex need not be exactly identical between all nodes in the network. For example, suppose we have two nodes a and b with street graphs G_a and G_b, respectively, that identify some real-world street vertex u with different geographic positions u_a and u_b and node a sends position u_a in a message to node b. Node b considers $u_a \equiv u_b$ if $\exists$ no other vertex $v \in G_b$ with coordinates nearer to u_a than u_b.

3.2 Traffic Monitoring

A fundamental aspect of the success of any VANET is the presence of a sufficient number of network nodes to allow forwarding of messages in the network. As we have discussed, messages in a VANET are forwarded along streets due to the unique constraints of this kind of network. However, due to various factors in a real-world situation, there is no guarantee that network-participating vehicles are present on any particular street at a given time. A lack of networked vehicles may occur due to factors such as date and time, road construction, detours, community events, traffic laws, bad weather, etc. Some of these factors affect all streets in a particular area, while other factors may cause only a few selected streets to be void of network nodes.

As discussed previously, A-STAR utilized static traffic information from bus schedules. Other static methods of traffic monitoring could include caching "typical" traffic data and potentially supplementing that data with updates about less-frequently scheduled traffic conditions. For example, nodes might store data about typical traffic patterns such as rush-hour commuter traffic on weekdays, and then they might also receive periodic updates about road construction or community events that disrupt these typical patterns.

While these kinds of typical traffic patterns have potential to persist for a significant amount of time, it is quite probable that temporary gaps in network coverage are common on most streets at frequent intervals. Any distance between two nodes that is greater than both nodes' transmission ranges causes a communication gap. These kinds of gaps may occur frequently because of traffic signals that stop vehicular traffic, for example. They may also be caused even when the road is full of vehicles if many of the vehicles are not network-equipped. These temporary gaps can be extremely disruptive because they often happen in a non-deterministic manner. A typical network gap is depicted in Figure 3.2 where vehicular traffic on a street is moving away from each other, thus partitioning the network.

Traffic monitoring in our protocol consists of both active and passive components that operate in real-time. For active traffic monitoring, the primary mechanism is the probe message: a RIVER protocol packet that is periodically sent by each node in the network. Probes perform dual functions of traffic detection and traffic information distribution. In addition, each node performs passive traffic monitoring by gathering data from each packet that it receives. Beyond the implicit beacon markers discussed in Section 3.1.1, probe and routing packets carry two other forms of traffic information: the **known edge list** and **weighted routes**.

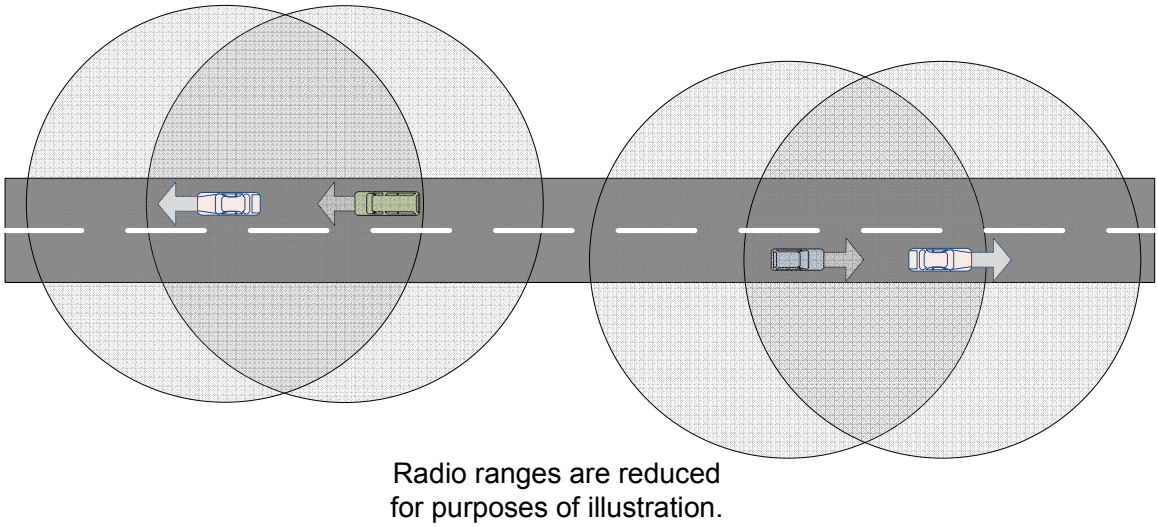


Figure 3.2: Formation of a Network Gap

3.2.1 Active Monitoring

In vehicular ad-hoc networking, network topology is constrained by the physical network of roadways because vehicular nodes are constrained to these roadways, and (especially in urban scenarios) there are often obstacles to radio communication along each side of the street. Therefore, like the edges of a graph, the road segments between intersections are one-dimensional in terms of communication: messages can be sent either to vehicles ahead of the current node or to vehicles behind it. As such, the majority of routing decisions are made at intersections. These decisions are crucial: sending a message down a street that contains a network gap causes the message to either be dropped, buffered, or to backtrack. With these factors in mind, it becomes clear that the shortest path between a sender and receiver is not always the most successful path. Instead, a VANET routing protocol must have a method to determine which street edges are most likely to result in delivery of a packet to the next intersection.

In VANETs, beacon messages primarily function as a mechanism for a node to advertise its existence to its neighbors. In a sense, this is a form of traffic awareness. Beacon-oriented traffic monitoring is employed by some of the routing protocols that have made limited use of real-time traffic monitoring, such as STAR and CAR. However, a node can detect beacons only within its radio range, and frequently, the reliable range of a radio may be less than the distance between street intersections. Consider city blocks as one example. City block sizes differ between cities (and also within

the same city), but a standard Manhattan city block is 270 m on its longest side. Superblocks are used to accommodate larger buildings in downtown settings or to form perimeters of neighborhoods in suburban settings. They create even longer distances between intersections along well-populated thoroughfares, where surrounding streets are often 1 km in length. However, typical 802.11b and 802.11g wireless range is about 100 m outdoors in the best case. For this reason, beacon-oriented traffic monitoring is insufficient to determine whether a street edge is reliable for message delivery to the next intersection.

3.2.2 Probe Messages

To determine if a message can be delivered along a particular street edge to the next intersection, RIVER uses a probe message. A probe is best described as an anycast message: it is sent to any node in a group of nodes defined by a particular geographic area. Its content is similar to a beacon message in that it does not carry a data payload. However, probe messages are not a one-hop broadcast.

Each node maintains a copy of the surrounding street layout in its street graph (Section 3.1.2) where each road segment is represented by an edge in the graph, incident on two vertices. A probe message is sent by a node that is located near a street vertex, and it is forwarded greedily to intended next-hop recipients along the street that is incident to that vertex. The destination node of a probe message is not known to its sender; the probe traverses a street edge and is finally received by any node within range of the opposite street vertex. If there is a gap in the network coverage along the street edge, the probe is dropped. However, if the probe is delivered to its destination vertex, any nodes at that vertex become aware that the vertex is traversable at that moment. When a departing probe is received, a return probe is generated back to its original sender so that the sender will also be aware of the connectivity of the probed street edge. (For pseudocode describing the receiving of packets and the processing of probes, see Section B.1 and Section B.3.)

Our protocol's probe messages act as implicit beacons for each forwarding node by including each forwarder's geographic position and address. They also carry the address and geographic position of their original sender, and the position of their destination vertex. Finally, each probe message also contains a **known edge list**, to be discussed further in Section 3.2.3. For a full diagram of a RIVER probe packet, see Section A.2.

3.2.3 Passive Monitoring

Each node also monitors traffic passively by monitoring messages that are sent within the network. Each message contains, either implicitly or explicitly, reliability information about edges in the network. These monitored messages may be messages that are sent directly to a node as a next-hop or destination. However, each node also taps into the link layer of its network stack and listens for RIVER packets that are addressed to another node. The learned reliability information is then shared within the network in a distributed manner.

In the RIVER model, routing is aided by gathering and distributing knowledge regarding the connectivity of edges in the street graph. This is partially enabled through passive monitoring. Whenever a node near a street vertex V_x receives a packet that has traversed an edge that is incident on V_x, this implies that the traversed edge is currently connected. (By connected, we mean that sufficient nodes are present along the edge to transmit a message along that edge.) Similar to the probe mechanism described earlier, our routing packets also contain information that allow a node to determine what edges the packet has traversed. Therefore, when node N_x near street vertex V_x receives a probe or routing packet that has traversed an edge incident to V_x, node N_x resets the weight of that edge in its street graph to the minimum value, which indicates that the traversed edge is connected.

Passive monitoring also enables a node to learn about edges of the street graph that may be far away from the node. As depicted in Figure 3.3, suppose a node receives a routing packet from a distant node, either for the purpose of receiving the packet or forwarding it on. (Alternatively, the node may also simply overhear such a packet, as we will describe in Section 3.2.6.) The node is already aware of the reliability of edges near it because it sends and receives probe packets along those edges (marked with an "x" in the figure). In addition, every edge in the routing packet's route (marked with a "y" in the figure) will be represented with an edge weight in the packet. Finally, any edges incident on the route will likely also have their reliability captured because the nodes that forward the packet from the source to the destination may add into the packet any reliability weights known to them also (marked with a "z" in the figure) within the known edge list. These features will be described further in this section.

Figure 3.3: Data Gained from Passive Monitoring of a Routing Packet

3.2.4 Weighted Routes

Every RIVER routing packet contains a list of anchor points for the route, identified by their geolocation. Any consecutive pair of these route anchor points represents an edge in the street graph of the sender node and has an edge weight associated with it. When constructing the routing packet, the sender includes this edge weight in the packet, along with a timestamp of when that reliability value was last updated.

When a routing packet is received at a node, the node analyzes the route and processes the reliability information associated with it. If the node is not the final recipient for this routing packet, it also updates the reliability information within the route prior to forwarding it on. The rules in Section 3.3.2 govern the processing of incoming reliability information and updating of outgoing reliability information. For a full diagram of a RIVER routing packet, see Section A.3.

3.2.5 Known Edge List

Each node using our protocol monitors beacon, probe, and routing messages, each of which contain a **known-edge list** (KEL). The known-edge list identifies edges by their endpoint geolocations and communicates reliability information about each edge. Upon sending a RIVER packet, the sending node selects edges from its street graph to share with other nodes, and places them in the known edge list with their reliability values and the time when each reliability value was last updated. Likewise, whenever a RIVER packet is received at a node, the node reads the known edge list and processes any edge reliability values found there. If the packet is a probe or routing packet that the node will forward on, the node selects edges to share from its street graph (which now includes the information contained in the received KEL) and updates the known-edge list in the packet before sending it on.

The known-edge list does provide benefits of improved throughput, as shown in Section 3.5. It is imperative however, that the protocol does not allow the list to grow too long. As the packet size grows longer, each node must transmit for a longer period of time, and this increases the probability that another node will attempt to transmit simultaneously. In a congested network, these simultaneous transmissions degrade network performance, which has a negative impact on throughput. In our simulations, we only encountered this problem when we artificially increased packet size to determine these effects. Routing packets are more likely to be affected by this problem because they also carry a route and data payload, so their known-edge list sizes should be limited appropriately. Maximum lengths can be set independently for known-edge lists in beacon, probe, and routing packets.

To minimize the packet size, edges are carefully selected for sharing from a known edge list. Each edge in a node's street graph is timestamped with the last time it was selected for sharing. Likewise, each update to an edge's reliability rating is also tracked. Only an edge whose reliability has been updated since the last time it was shared is eligible for sharing again. Edges whose reliability is unknown (and set to a default value) are not eligible for sharing. Furthermore, the selection algorithm prefers edges that have been updated recently, and it further prefers edges whose updates originate from first-hand knowledge, as opposed to an edge whose reliability is updated because of its inclusion in a known edge list from some other node. These differences are further discussed in Section 3.3.2.

Known edge lists exist on all types of RIVER messages: beacons, probes, and routing packets. The information on a beacon travels only a single hop, so recipients are likely to have similar information already, either from first-hand knowledge (since

they are located near the same edges as the sender) or from other nodes' beacons. However, beacons are the most frequently transmitted message type in the protocol, so they can spread updates more quickly than other message types. Probes travel along edges, and therefore they distribute the edge information farther from the sender than beacons, increasing their likelihood of sharing useful information. Probes are only sent when a node is near a street vertex, and they are transmitted less frequently. Routes generally carry their known edge lists the farthest distance, benefiting many nodes who may not otherwise travel near the sender's location to learn about traffic there first-hand.

3.2.6 Eavesdropping

Since beacon packets are broadcast to every node within one-hop radius of the sender, every node in radio range processes the beacon information. However, probe and routing packets are forwarded to a specific recipient at each hop. By default, other nodes within radio range of the packet transmission discard the packet at the link layer of their protocol stack. However, information contained within these probe and routing packets carries value for other nodes in the area besides their intended recipients. If these other nodes received the packets also, they could use the implicit beacon information in the packet to update their neighbor tables, process the known edge list in the packet to update their street edge reliability data, and (for routing packets) process the reliability information of edges within the route itself. In order to perform passive traffic monitoring, each node running our protocol taps into the link layer of its network stack. By eavesdropping at this level, any RIVER probe and routing packets that are not addressed to the current node can be pushed up the protocol stack for processing. Figure 3.4 depicts a sender at left transmitting a packet to its intended receiver at right. Any other nodes within radio range of the sender (such as the middle node in the diagram) may overhear this packet. Using this approach, each node gains useful information that would otherwise require additional packet transmissions, consuming valuable network bandwidth.

3.3 Edge Reliability

A crucial component of our protocol is its ability to estimate the reliability of a particular street edge. RIVER uses this reliability data as the primary factor in determining a successful routing path from a sender node to a receiver node. As mentioned in Section 3.2, temporary, non-deterministic traffic gaps pose a particularly

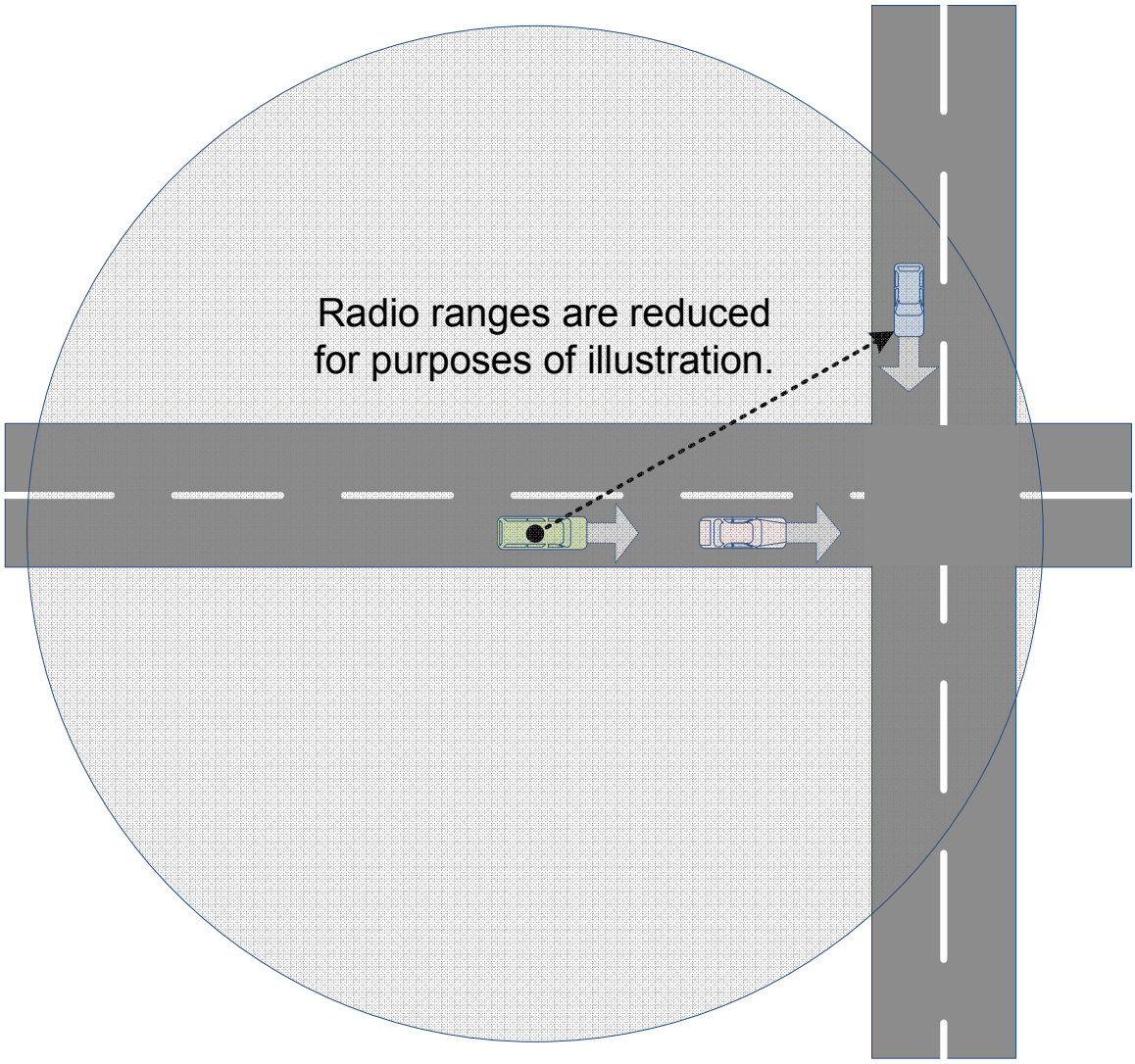


Figure 3.4: Link Layer Eavesdropping

challenging problem for any VANET protocol that relies on static data to detect traffic anomalies. These kinds of gaps occur suddenly, often without predictability. There are cases when sparse traffic may persist for long periods of time (eg. road-closed conditions), but these cases happen infrequently, and their presence is still captured by real-time monitoring. Vehicular nodes move quickly and frequently, so it is infeasible for each node to track the movement of all other nodes across a particular area to determine usable routes. Instead, we hypothesize that it is more efficient to determine if a particular street edge was recently reliable and share this information with others.

3.3.1 Determining Reliable Paths

Each node in the RIVER model assigns a weight to every edge in its street graph. To determine reliable paths, the protocol assigns these weights using both first-hand observation and second-hand knowledge. First-hand observations include the information that each node gains when it receives a packet or when it attempts to send a probe or routing message to another node. Second-hand observations include the passive monitoring of known edge lists stored in beacons, probes, and routing packets, and the monitoring of edge weights contained within routing messages.

In shortest-path routing algorithms, each edge weight would be based on the length of the street segment represented by the edge. Our protocol is not a shortest-path routing algorithm in this sense; its edges are weighted with their reliability rating. A small weight (the minimum weight is zero) indicates greater reliability; a large weight indicates an unreliable edge, and the maximum weight indicates an edge that is known to be not traversable.

With these weights assigned to each edge, our protocol uses Dijkstra's least weight path algorithm [27] to calculate what it considers the most reliable routing path. This calculation is performed at the originating sender of a routing packet and may be performed again later. The route, along with each reliability rating used in the calculation, is written into the packet.

Note that when using reliability as a path metric, distance is still taken into account. Dijkstra's least weight path algorithm finds the least-weighted path based on the sum of the path weights. If two paths P_x and P_y have equal weights on each edge but P_x has more edges (is a longer path) than P_y, then P_y is chosen because its total weight is less. The shorter of the two paths is chosen.

An example of this is demonstrated in Figure 3.5. In this figure, all edge weights (shown as w) except for $V_s \rightarrow V_d$ are of equal weight. The shortest path $V_s \rightarrow V_d$ represents an unreliable path where packets would be dropped if transmission were attempted along this path. The other two paths from V_s to V_d have equal edge weights along each edge but the paths are different lengths. For path
$(V_s \rightarrow V_1 \rightarrow V_3 \rightarrow V_d)$
the total weight is 3. Each edge of the other remaining path
$(V_s \rightarrow V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_4 \rightarrow V_d)$
is equally reliable, but the total weight is 5, so RIVER chooses the shorter path.

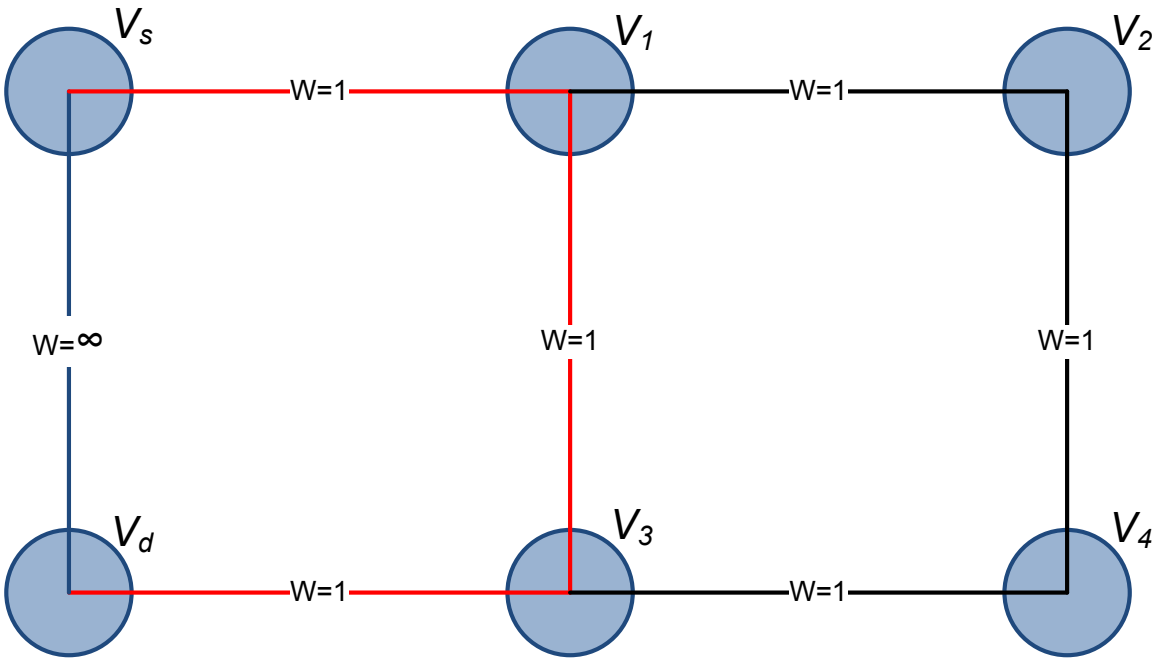


Figure 3.5: Three Potential Paths

3.3.2 Reliability Distribution

When a node sends a beacon, probe, or routing packet that contains a known edge list, that node distributes its street graph reliability information within the packet. For clarity here, we define an edge's reliability rating as **shared** when a node writes the edge's reliability rating into a packet's known-edge list for distribution. We define an edge's reliability rating as **declared** when a node reads this rating from a known edge list in a packet that it has received.

In addition to the reliability rating, each node also tracks other values relative to each edge in its street graph, as shown in Table 3.1, while other important data points are calculated, as shown in Table 3.2. These values are used to make a number of decisions about edges, calculate the reliability of each edge, and to determine when a declared value should be used or discarded.

In an effort to conserve network bandwidth, a node running our protocol does not simply write all of its known edge information into every packet it sends. Instead, it selects an edge for sharing based on several criteria: whether it has been updated since the last time it was shared, how recent the update was, and whether the update originated from first-hand observation or a second-hand declared value. The most selective factor is whether the edge has been updated since the last time it was

Table 3.1: Edge Data Fields

Field	Description
Packet Received	Stores a timestamp of the last time when this node received a packet that had traversed this edge.
Last Marked	Stores a timestamp of the last time when this node last marked an edge as disconnected.
Last Declared	When this node accepts a declared reliability value, this field stores the timestamp associated with the declared value.
Last Shared	Stores a timestamp of the last time when this node shared this edge's reliability by writing it into a packet.
Last Probe Sent	Stores a timestamp of the last time when this node sent a departing probe along this edge.
First Probe Sent	Stores a timestamp of the first time when this node first sent a probe along this edge.
Static Value	When a static reliability rating is in effect, it is stored here.

Table 3.2: Calculated Edge Data

Data	Description
Reliability	Based on the data known about the edge, this may be a calculated value (Section 3.3.3) or equal to the static reliability value above.
Shareability	A ranking that dictates how worthy an edge reliability value is to be shared, based on several factors, discussed below.
Last Updated	Equal to the most recent of the edge's packet received, last marked, and last declared timestamps.

shared: an edge is only shared if this condition is true. Beyond that factor, edges are ranked relative to one another for "shareability". An edge that was updated more recently is favored over an edge that was updated less recently, so a relative shareability ranking is given to each edge based on the time that has elapsed since its last update. Then, the algorithm further increases the ranking of any edge whose reliability rating is not based solely on a declared value. When a limited number of edges may be written into a known-edge list, these rankings are used to select the most eligible edges, and the others are excluded from the packet.

When a node receives declared information about the reliability of an edge, it must decide whether to accept or reject the declared value based on the timestamp associated with the declared value and the timestamp information the node associates

with its current edge rating. If the node has no reliability information for the edge from any source (receiving a packet over the edge, marking the edge unreliable in the past, or from a prior declaration of the edge), then it accepts the declared value. If the node does already have reliability information for the edge, then it compares the declared timestamp information to its own last updated timestamp and accepts the declared rating if the declared timestamp is more recent. In either case that the declared value is accepted, the node sets the edge's last declared timestamp to the timestamp recorded in the packet (not to the current time when the value is accepted) and sets the static reliability value for the edge to the declared value.

3.3.3 Reliability Calculation

Network gaps frequently emerge and dissolve, so the RIVER protocol discards notions of persistent, static traffic models in favor of a more dynamic model. Traffic density metrics (typically the number of vehicles per unit distance) are irrelevant if they are inconsistent across the length of a street edge because dense traffic on one area of a street followed by sparse traffic in another area can still result in a disconnected edge. Traffic patterns change frequently and persistent models may often lag actual conditions. Real-time information ensures that the network can adapt to sudden changes. The transmission of a packet from sender to receiver happens on a much shorter time scale than traffic movements, so even a network gap that has only formed for a few seconds can cause many packets to be dropped or delayed. To ensure fewer packet delays, up-to-date information is preferable. The freshness of a node's reliability information is important to take into account. Older information is less likely to reflect reality than recent information.

In order to give preference to recent information, when first-hand observed information is available, our protocol calculates the reliability of an edge as the number of milliseconds since the edge was last known to be traversed by a packet. With this model, a low reliability value represents a recently-traversed edge. These low values are preferred over high values when generating a route.

At the moment that a node receives a packet that has traversed some edge e, the node sets the reliability value of e to zero (most reliable). As time elapses from that event, the reliability value for the edge decays in a linear fashion to a higher (less reliable) value until another packet traverses the edge. To accelerate the decay of an edge that appears to be unreliable, a constant **waiting multiplier** is used in the calculation. When a node attempts to send a probe along an edge but does not receive a response, the waiting multiplier is used in the calculation to discourage the

use of that edge for routing. The waiting multiplier remains in effect for that edge until the edge weight is updated with new information. Another constant **never-received multiplier** is used in cases where no packet has ever been received along the edge.

In addition to the calculated value, there are some static values used in RIVER reliability ratings. When no packets are received along an edge (and the node has not sent a probe along this edge to test it), a time period known as the **reliability default** eventually elapses. This value, also measured in milliseconds, acts as a default value for any edge whose reliability is undetermined. If the reliability data about a particular edge is not updated within this period, its reliability reverts to this default value. Furthermore, if a node on some edge attempts to forward a packet along that edge but can find no neighbor to whom the packet can be sent, the node instantly marks that edge as unreliable by setting it to ∞ (represented by the largest value that can be stored in the data range). This unreliable rating is distributed to other nodes through the known edge list of the packet.

The complete set of ordered cases for evaluating the reliability of an edge are shown in Table 3.3.

As modeled, the reliability ratings in our protocol have a number of notable characteristics. Reliable edges have a relatively small range of values when compared to unreliable edges. This design point is coupled with the use of Dijkstra's least weight path algorithm. The intention here is to prevent Dijkstra's algorithm from preferring a single unreliable edge over several reliable edges. By increasing the reliability value for unreliable edges, we make this less feasible. That is, the values are set up in such a way that the sum weight of many reliable edges tends to be less than the weight of a single unreliable edge. Another notable trait of reliability values is that a reliable edge eventually "times out" and returns to a state of unknown reliability when the reliability default time elapses. Inversely, an unreliable edge remains in an unreliable state until it is observed to be more reliable. These characteristics reflect the transitory nature of the network connectivity of street edges, and our desire to prefer routing along edges known to be reliable. One final characteristic of note is that an unreliable edge's reliability rating stabilizes when a node can no longer test the edge by sending probes along it.

Table 3.3: Evaluating Reliability

	Condition	**Reliability Result**
X	This node marked the edge as "unreliable" within the reliability default duration and this is still the most up-to-date data	"unreliable" rating
0	This node accepted another node's declared rating and the declared timestamp is within the reliability default duration and this is still the most up-to-date data	declared reliability rating
1	This node received a packet along this edge within the reliability default duration and this is still the most up-to-date data	time elapsed since packet received
2	This node has neither received nor sent any packets along this edge.	reliability default
3	This node received a packet along the edge but the reliability default elapsed since that event and the node has never sent a probe along the edge	reliability default
4	This node received a packet along the edge but the reliability default elapsed since that event and the last probe this node sent along the edge was before that.	reliability default
5	This node received a packet along the edge but the reliability default elapsed since that event and the last probe this node sent along the edge was after that.	reliability default + ((time elapsed between last packet received and last probe sent) $*$ waiting multiplier)
6	This node has sent a probe along this edge but has never received a packet along the edge.	reliability default + ((time elapsed between first and last probe sent) $*$ waiting multiplier $*$ 2)

3.4 Routing

At its most basic level, the routing algorithm in RIVER is not unlike other geographic routing algorithms; our protocol identifies a path that connects a number of geographic locations and attempts to forward the message along that path. Because RIVER is a VANET routing protocol, the geographic locations are vertices of its street graph (typically road intersections) and the edges that connect those vertices are roadways. The reliability features of the protocol cause it to select these street edges based on their estimated reliability.

When a node originates a new message, it must first identify the geographic location of the message destination. In reality, the node may have cached this information from a previous message exchange with the destination, or it may need to inquire about the location. The design of an efficient location service is outside the scope of this routing protocol and is a separate area of research [7] [25] [46] [47] [55] [58]. For our simulations of RIVER, the sending node identifies the initial geographic location of its message destination using an external location database. This is the only instance during a message transmission when an external location database is consulted.

After identifying the geographic location of the destination, the distance to the destination is computed. If this distance is small (for example, if the sender and receiver are already within radio range of each other or they are located on the same street edge), then the sending node simply forwards the packet greedily toward the destination. Otherwise, the sending node consults its reliability-weighted street graph and uses Dijkstra's least weight path algorithm to calculate the most reliable path to the destination. The geographic locations of the vertices of the street graph that make up the routing path are known as **anchor points**, and we often refer to the route itself as an **anchor path**. A RIVER routing header is generated around the data packet, and the anchor path is written into that routing header.

The process of identifying a next-hop node begins at the sender node and repeats at each forwarding node. The node in possession of the routing packet consults the anchor path for the current anchor point. Then, it refers to its neighbors-table to identify the node that it believes is within radio range and is nearest to the current anchor point. The node sets the next-hop address in the routing packet's encapsulating header (eg. IP header destination address [69]) and attempts to send the packet. At this point, our protocol takes advantage of a link-level transmission failure detection feature, as is described in GPSR [45]. If this feature is enabled and the link layer

cannot recognize that a link was established with the next hop (eg. no link-layer acknowledgement from the next hop), then the forwarding node repeats this process and attempts to send the packet again.

As the packet is received at each hop (for pseudocode, see Section B.1 and Section B.2), the node performs its passive monitoring functions: conditionally updating its edge weights with values declared in the known edge list and in the weighted route of the packet (Section 3.3.2). Then, the node examines the current anchor point in the packet. When an anchor point has been reached or passed (Section 3.4.8), then a pointer is incremented to set a new "current anchor point". If only one anchor point remains in the route, the node checks whether it is between the last anchor point and the destination and increments the anchor pointer if that is the case. Finally, the algorithm chooses its next hop. If anchor points remain, the next hop is chosen based on the current anchor point, otherwise the next hop is chosen greedily toward the destination geolocation stored in the route header. If a next hop is found in the neighbor table, the packet is forwarded to that node.

If no neighbor can be found closer to the current anchor point, then the node tries to find a neighbor closer to the subsequent anchor point instead (explained further in Section 3.4.9).

Figure 3.6 shows an example where the two intersections shown represent the anchor points of the route. Each node greedily forwards to the farthest node within its radio range that is also in the direction of the next anchor point.

3.4.1 Route Recovery

When a node attempts to find a next-hop for routing a packet as described above, if no suitable next-hop neighbor can be found, RIVER's recovery function is engaged. First, the failed anchor path is examined. The edge where the failure occurred is determined by its vertices, which consist of the last anchor point that was successfully reached and the current anchor point in the route. (If the route has failed at the first anchor in the route, our protocol cannot recover and drops the packet.) This failed edge is marked in the street graph by giving it the maximum weight possible, which we will refer to as the **disconnected edge weight**. With the disconnected edge weight in place, Dijkstra's least weight path algorithm is run on the graph again. If the algorithm finds a route whose mean weight is less than the current route's mean weight by a significant threshold, then the routing header's remaining anchor path is overwritten with the proposed anchor path. The significant threshold is defined as 50% of the reliability default value.

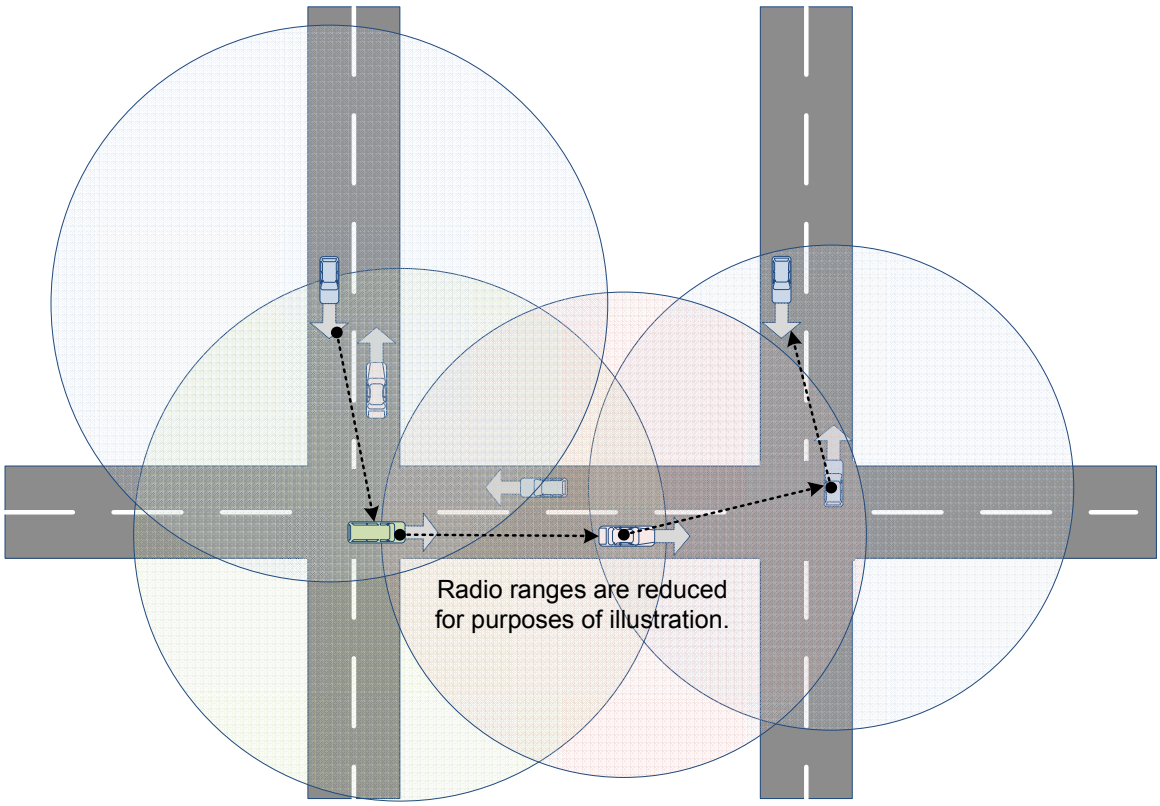


Figure 3.6: Routing Example

Once the new anchor path is established in the routing header, the next edge in the proposed route is examined. If the path's next edge weight is less than the disconnected edge weight, the process of searching for a next-hop neighbor resumes by a recursive call. If no next hop can be found, the recovery process may be engaged again. The routing process drops the packet and exits out of the recursive call if the weight of the proposed routing path's next edge is equal to the disconnected edge weight. This indicates that Dijkstra's algorithm found that the least weight path is a path whose leading edge is already marked as a failed edge.

3.4.2 Route Recalculation

Our protocol also has a route recalculation feature that is similar to the recovery feature described above. This feature has the potential to prevent a route recovery scenario before a failure occurs in selecting a next-hop neighbor. For this reason, the recalculation feature can be considered a proactive version of the recovery feature, while recovery only occurs as a reaction to a failed next-hop neighbor selection.

If this feature is enabled, the opportunity for recalculating a route is evaluated at a forwarding node when the current anchor point in the route has been reached or passed at that node. When this occurs, the node runs Dijkstra's algorithm to propose a new anchor path. If the proposed path's mean weight is less than the current path's mean weight by a significant threshold, then the remaining anchor path in the packet is overwritten with the proposed anchor path. As in route recovery, the significant threshold is defined as 50% of the reliability default value. This threshold was introduced specifically for the route recalculation feature. In early versions of our protocol, it was sometimes the case that two or more nodes performing recalculation along a route would have a slight discrepancy about the weight of one or more edges in that route and send the packet back and forth to each other in a loop. The threshold reduces the possibility of this occurrence.

3.4.3 Routing Loops

One common problem for routing algorithms is the occurrence of loops within the route of a packet. Unnecessary forwarding of packets along a loop increases network congestion. Packets may be dropped when their time-to-live (TTL) values are exceeded prematurely or because excessive network congestion prevents delivery.

We categorize routing loops in three inclusive groupings. A repeat-node loop occurs when a node receives a packet that it previously forwarded. Similarly, we define a repeat-vertex loop as the condition of a route that traverses a particular street vertex more than once. Finally, we define a repeat-edge loop as the condition of a route that traverses a particular street edge in the same direction more than once. The distinction about edge direction for a repeated edge loop is important since it permits backtracking to be outside the definition of a repeat-edge loop. Note that a repeat-edge loop implies a repeat-vertex loop. Due to the movement of nodes, it is possible to encounter a repeat-vertex or repeat-edge loop without a repeat-node loop.

Dijkstra's algorithm finds a least-weight path between two vertices in a graph. It does this by generating a shortest path tree: a set of paths with the lowest weight between the destination vertex and every other vertex in the graph. Since a path containing a repeat-vertex or repeat-edge loop produces a path with a greater sum weight than the same path without the loop, we observe that the path found by Dijkstra's algorithm must be free of these kinds of loops. However, our protocol allows anchor paths to be recomputed when a failure occurs (if recovery mode is enabled) or at each anchor point (if route-recalculation mode is enabled). When a route is recomputed, the edge weights recorded in the street graphs of different nodes

may provide contradictory least-weight-paths, and repeat-vertex or repeat-edge loops may result.

3.4.4 Repeat-Node Loops

The most trivial example of a repeat-node loop occurs due to the movement of nodes between beacon intervals. Consider two nodes, N_x and N_y traveling on parallel paths toward each other and within radio range of each other. Suppose N_y beacons its position and N_x notes this position in its neighbors-table. Suppose also that before N_y can beacon its position again, the nodes travel past each other. Next, N_x forwards a packet that is destined in the same direction as N_x, and based on its neighbor-table information, N_x considers N_y to be the farthest neighbor in that direction. N_x forwards the packet to N_y, and the packet contains an implicit beacon with the position of N_x. N_y adds the new position information for N_x into its neighbors-table. N_y now determines that the received packet needs to be forwarded, and it identifies N_x as the farthest neighbor in the direction of the packet's destination. N_y then forwards the packet to N_x, and the packet experiences a repeat-node loop.

Without accurate position information at all times, such repeat-node loops may occur. Increasing the frequency at which beacons are sent by each node may reduce the occurrence of this kind of loop, but it would also increase network traffic and may lead to dropped packets due to congestion. A more acceptable strategy for reducing repeat-node loops of this kind would be to require each node to estimate the current position of its neighbors based on their last-known velocity and heading, such as in the velocity vectors used in CAR [63]. If velocity and heading were calculated based on consecutive beacons from a neighbor node, such information would not be available during the time interval between the first and second beacons from a particular neighbor. Furthermore, if the nodes were moving rapidly in opposite directions, only one beacon may be possible before they move out of radio range of each other. For this reason, it may be beneficial to include velocity and heading information in each beacon if this position-prediction strategy were to be used. Note also that position estimates cannot entirely prevent repeat-node loops of this kind since the estimates will be inaccurate when a vehicle changes its velocity or heading after its beacon is sent.

3.4.5 Repeat-Vertex Loops

For a repeat-vertex loop example, refer to the street graphs in Figure 3.7 with vertices V_s, V_1, V_2, V_3, and V_d, each containing a node (not depicted) that we will refer to with a corresponding subscript (N_s, N_1, N_2, N_3, and N_d) and each edge weight marked as w.

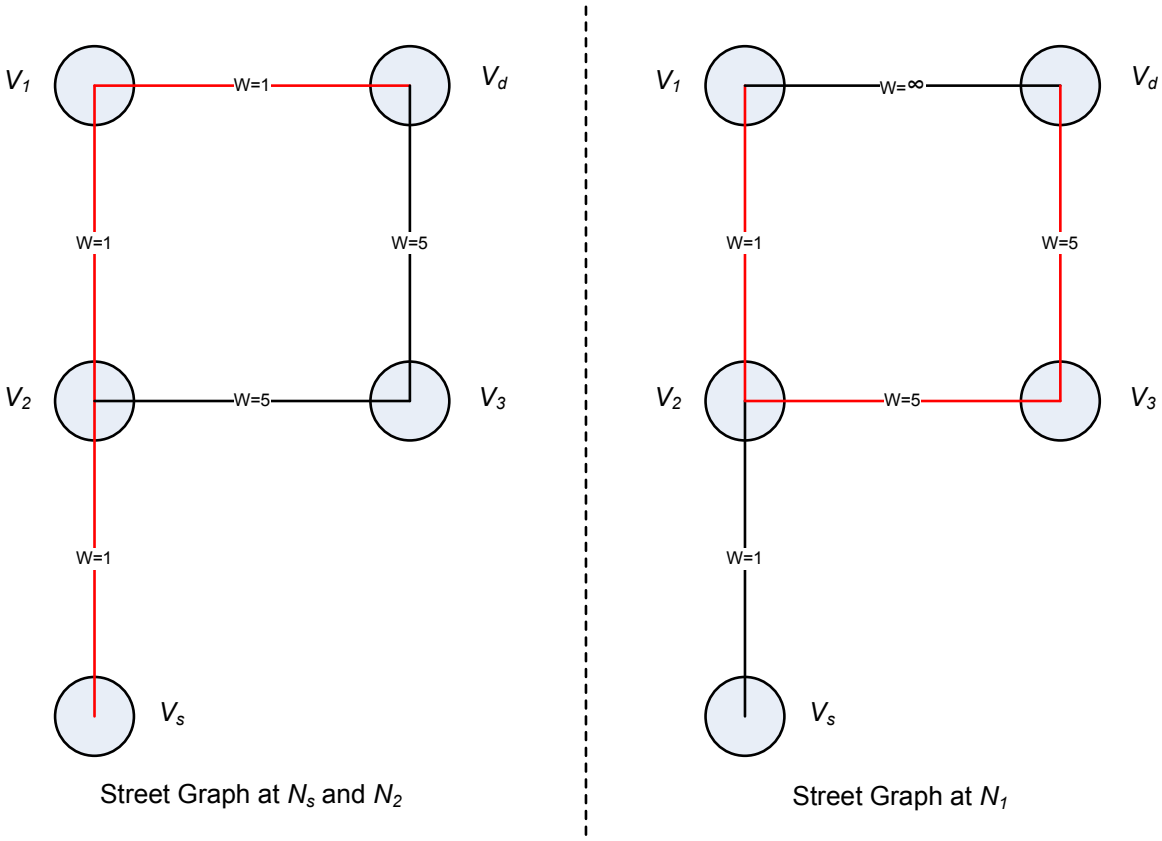


Figure 3.7: Repeat-Vertex Loop

Suppose N_s sends a message to N_d and its street graph contains edge weights as marked in the left side of the figure. Then, N_s will choose the following least-weight path for routing the message: $(V_s \rightarrow V_2 \rightarrow V_1 \rightarrow V_d)$

However, suppose also that route recovery is enabled and when the message reaches N_1 at vertex V_1, N_1 attempts to send the message to V_d but can find no neighbor along the edge. N_1 marks that edge as disconnected and evaluates its edge weights (as shown on the right side of the figure) to calculate a new least-weight path for the remainder of the route, which overwrites the unused portion of the route in the routing header with: $(V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_d)$

Then when the message reaches vertex V_1, the full anchor path traversed by the packet would have traversed vertex V_2 twice: $(V_s \rightarrow V_2 \rightarrow V_1 \rightarrow V_2)$

Although a repeat-vertex loop has occurred, the alternative would be to drop or queue the packet at N_2. Despite the repeat-vertex loop, backtracking is the desired outcome in this scenario.

3.4.6 Repeat-Edge Loops

For a repeat-edge loop example, refer to the street graphs in Figure 3.8 with vertices V_s, V_1, V_2, V_3, V_4, and V_d, each containing a node (not depicted) marked with a corresponding subscript (N_s, N_1, N_2, N_3, N_4, and N_d) and each edge weight marked as w.

Suppose N_s sends a message to N_d and its street graph contains edge weights as marked in the top graph of the figure. Then, N_s will choose the following least-weight path for routing the message: $(V_s \rightarrow V_4 \rightarrow V_2 \rightarrow V_3 \rightarrow V_d)$

However, suppose also that route recovery is enabled and when the message reaches N_3 at vertex V_3, N_3 attempts to send the message to V_d but can find no neighbor along the edge. N_3 marks that edge as disconnected and evaluates its edge weights (as shown in the middle graph of the figure) to calculate a new least-weight path for the remainder of the route, which overwrites the unused portion of the route in the routing header with: $(V_3 \rightarrow V_2 \rightarrow V_4 \rightarrow V_d)$

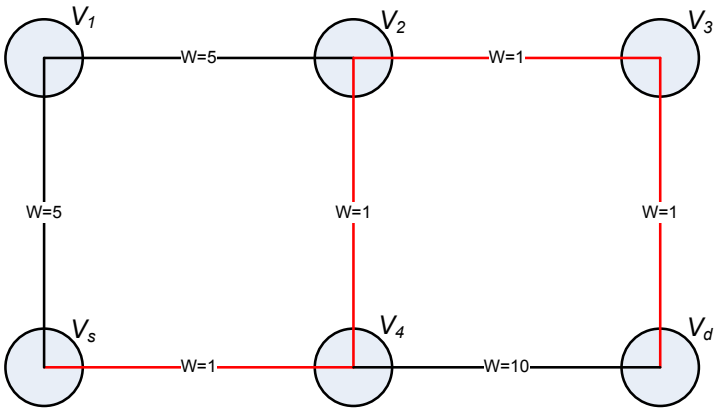
Then when the message reaches vertex V_2, node N_2 attempts to forward the message to V_4 but can find no neighbor along the edge. N_2 marks that edge as disconnected and calculates a new path (as shown in the bottom graph of the figure), overwriting the unused portion of the existing route with this new path in the routing header: $(V_2 \rightarrow V_1 \rightarrow V_s \rightarrow V_4 \rightarrow V_d)$

Then when the message reaches vertex V_4 for the second time, the full anchor path traversed by the packet would have traversed edge $V_s \rightarrow V_4$ twice:
$(V_s \rightarrow V_4 \rightarrow V_2 \rightarrow V_3 \rightarrow V_2 \rightarrow V_1 \rightarrow V_s \rightarrow V_4 \rightarrow V_d)$

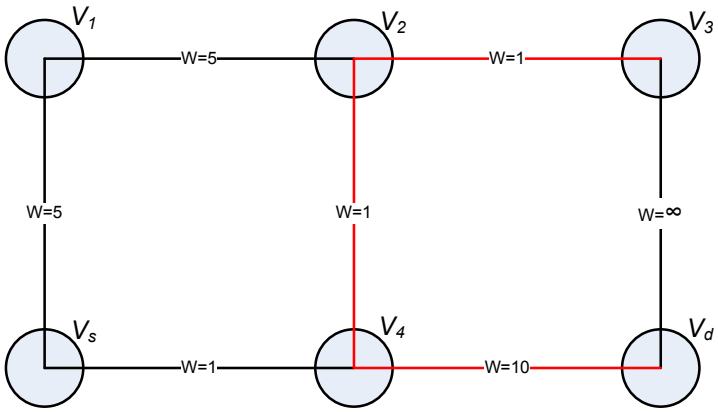
Although a repeat-edge loop has occurred, the alternative would be to drop or queue the packet at N_3 or N_2. Despite the repeat-edge loop, backtracking is the desired outcome in this scenario.

3.4.7 Reducing Loop Occurrences

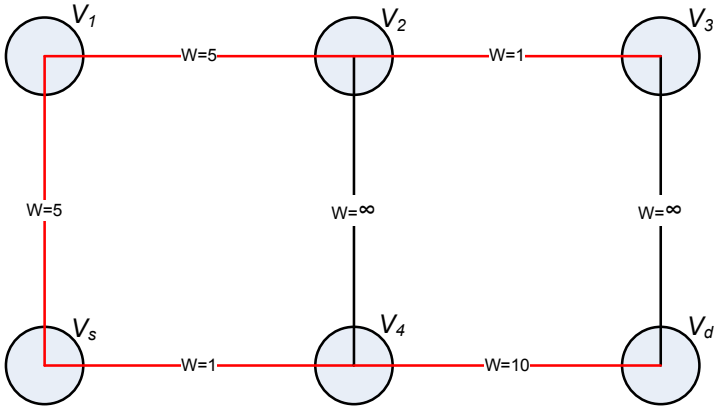
Due to the non-deterministic nature of vehicular movements, we have shown how sudden network gaps can force our protocol into a scenario where a routing loop



Street Graphs at N_s, N_4, and N_2



Street Graph at N_3



Street Graph at N_2 (Repeat Visit)

Figure 3.8: Repeat-Edge Loop

must occur for packet delivery to continue. We choose not to eliminate routing loops entirely because doing so reduces throughput (by dropping packets) or delays delivery (by queuing them until the network gap is reconnected). Instead, RIVER adopts a perseverance strategy for packet delivery.

To reduce the occurrence of routing loops, each packet header contains the last-known weight for each edge in its anchor route. This ensures that when a route recalculation occurs, the node performing the recalculation is provided with the most up-to-date traffic information possible for each edge in the already-traversed anchor route. In addition, the packet header contains a known-edge list for adjacent edges encountered during the packet's lifetime. If the packet has attempted to traverse an edge and found it failing, then it includes the weight of the failed edge in the packet's known-edge list. Therefore, if another node later along the anchor route's path must recalculate the anchor route (e.g. to recover from another edge failure), it will have the most recent traffic information possible about edges that the packet has attempted to traverse. Unless the node has more recent information (indicating reliability) about an edge that the packet has already failed to traverse, it will not attempt to send the packet down that previously-failed edge again. Finally, each packet is expected to have a TTL field in its network-layer header (for example, this is present in the IP datagram [69]) that will eventually cause the packet to be discarded when the TTL expires.

Even with the level of throughput that our protocol provides, some packets are dropped due to loops and other unavoidable factors. In a typical TCP/IP network stack, it is the responsibility of the transport layer to detect these problems and resend dropped packets if an application requires 100% delivery of packets. In VANET applications, there are many use-cases where some delivery failures are acceptable. Likewise, an appropriate transport protocol is necessary for applications that expect delivery of all packets in a VANET.

3.4.8 Past Anchor Point, Outside Zone

In the strictest sense, forwarding packets along an anchor route involves greedily forwarding toward each anchor point until the packet arrives at a node that is within some predefined range of the anchor point, called the **vertex range**. However, during this process, complexities arise due to the differences between the vertex range and each node's radio range and the density of traffic.

Consider Figure 3.9 where node N_a is forwarding a packet toward the anchor point at the depicted intersection. The subsequent anchor point for this packet is along the

street edge in the direction beyond node N_b. The vertex range for the current anchor point is shown as a circle, and node N_b is the closest node to the anchor point but is still outside the vertex range within which the anchor point is considered "reached".

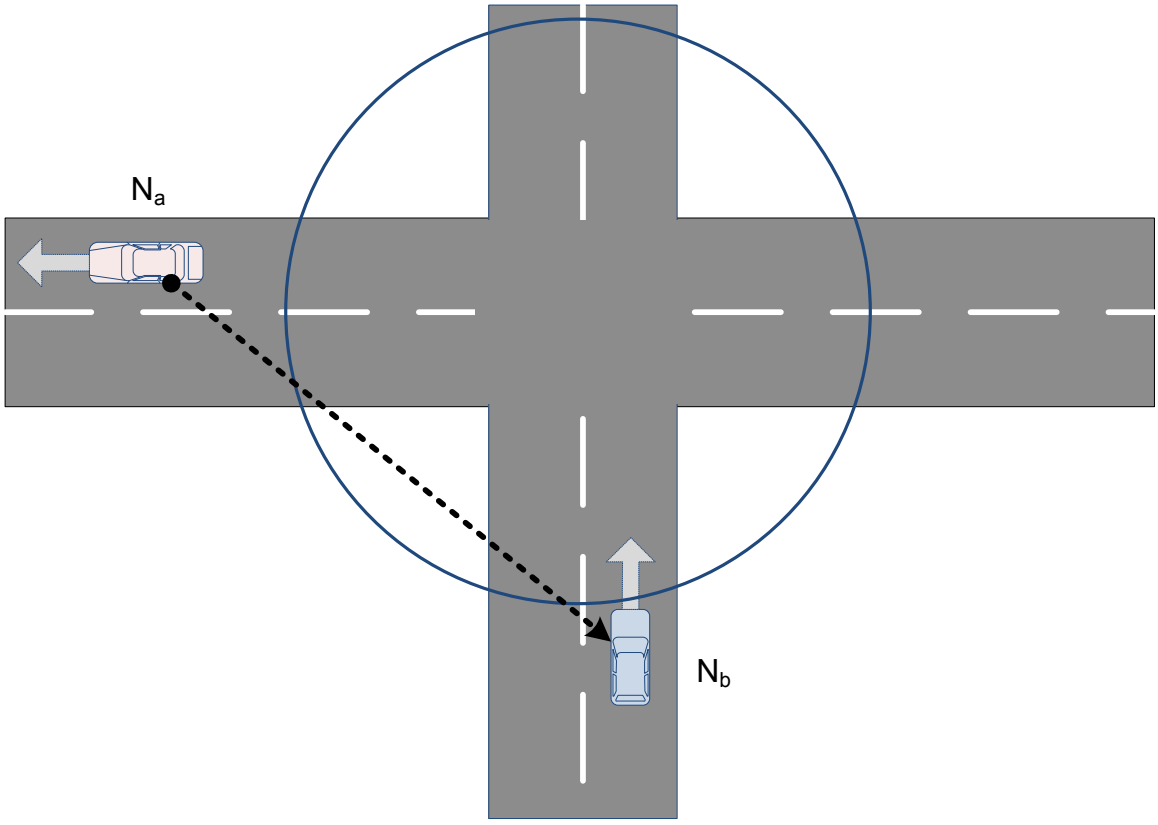


Figure 3.9: Past Anchor Point, Outside Zone

According to greedy forwarding, node N_a forwards the packet to the closer node N_b. When N_b receives the packet, there is still no node closer to the anchor point than node N_b, and N_b is still outside the vertex range. Since the anchor point has not yet been reached, this is technically a local maximum. Strict greedy routing would dictate that node N_b should drop the packet.

However, since node N_b is on the street edge that leads to the subsequent anchor point in this anchor route, it is premature to drop the packet at this point. Our protocol contains an optimization to handle this scenario. When a node receives a routing packet with multiple anchor points remaining, it retrieves the current anchor point and the subsequent anchor point (or the final destination if no more anchor points exist) for the anchor route. If the node determines that it is located between those two points, it increments the AP pointer in the packet. Thus, RIVER detects

when a packet has passed an anchor point, even if the packet never actually reached it.

3.4.9 Outside Zone, No Closer Neighbor

A similar scenario happens when node N_a is closer to the anchor point than node N_b. Consider Figure 3.10 where node N_a is forwarding a packet toward the anchor point at the depicted intersection. The subsequent anchor point for this packet is along the street edge in the direction beyond node N_b. The vertex range for the current anchor point is shown as a circle, and node N_a is the closest node to the anchor point but is still outside the vertex range within which the anchor point is considered "reached". The packet has technically reached a local maximum at node N_a, prompting it to be dropped in a typical greedy algorithm.

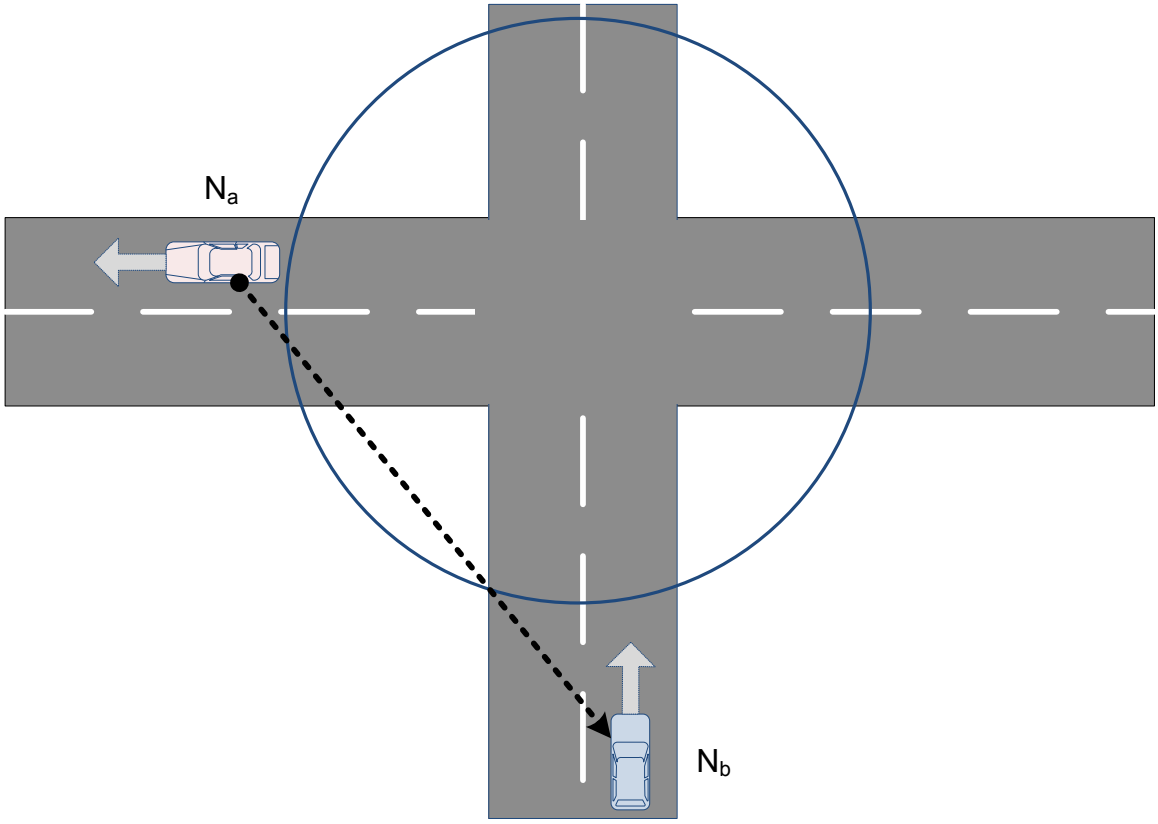


Figure 3.10: Outside Zone, No Closer Neighbor

Since node N_b is within radio range of node N_a, and node N_b is in the direction of the subsequent anchor point, dropping the packet is a poor choice in this case. Our protocol contains an optimization to handle this scenario. If a node fails to find a

neighbor closer to the current anchor point than itself and the current anchor point is not the final anchor point in the route, then it may instead look for a neighbor nearest to the subsequent anchor point such that the neighbor is located on the street edge between the current anchor point and the subsequent anchor point. Instead of dropping the packet when node N_a encounters a local maximum for the current anchor point, node N_a finds node N_b and forward to that node. As described in Section 3.4.8, node N_b detects that the packet has passed the anchor point and increments the AP pointer appropriately.

3.5 Performance Evaluation

To evaluate RIVER, we simulated the protocol with the *ns-2* simulator [30] at version 2.33 using the CMU wireless extension. The simulations were performed with various parameter settings to test different scenarios and feature sets of our protocol. The protocol was also compared against some of its peers: the STAR routing protocol [34], the GPSR routing protocol [45], and a shortest-path VANET routing algorithm. For all results, each simulation configuration was repeated for 20 iterations with a different random number generator (RNG) seed at each iteration, and the statistical mean of these iterations was calculated. For RIVER throughput measurements, standard deviation was about 7.3% of throughput. As expected, 95% of throughput values were within two standard deviations of the mean.

We used the following metrics to evaluate performance through our simulations. *Data throughput* represents the mean percentage of routed data packets that were successfully delivered. *Route header size* measures the average size of a routing packet, excluding the data portion of the packet. *Forwards per route* represents the average hop count of a routing packet. *Route transit time* represents the number of seconds required to deliver a routing packet from its original sender to its final destination.

3.5.1 Simulation Setup

For these simulations, an urban "Manhattan" street grid was used with 5 streets running in the horizontal and vertical directions spaced approximately 400 m apart for a total area of approximately 6.05 km^2. This simulation area was populated with varying traffic densities of 100 to 300 vehicular network nodes. Vehicles traveled in both directions along each street, and vehicles may turn at intersections. To simulate urban conditions, vehicle speeds range from 11 km/h to 51 km/h, with an average speed of 36 km/h. Each simulation iteration used the same movement pattern.

The connection pattern consisted of 5 sender/receiver pairs using constant bit rate (CBR) data flows of 512 Kbps. That is, each sender transmitted a 512 byte packet every 8 seconds, and each sender sent 21 packets, for a total of 105 packets sent during each simulation. Each packet sent was offset by at least one second from the previous send to ensure that no senders transmitted simultaneously. Each simulation ran for 20 seconds prior to any routing packet transmissions, and the simulation ran for 15 seconds following the final send, for a total of 200 seconds of simulation time (consistent with the simulation times for STAR [34]). For each simulation iteration, the sender/receiver node pairs were randomly selected.

3.5.2 RIVER Feature Analysis

To analyze the effectiveness of various features of our protocol, we simulated RIVER performance under a multitude of feature combinations and studied the results.

3.5.3 Route Recalculation and Recovery

We evaluated several RIVER protocol options for preventing and/or recovering from network gap routing failures as discussed in Section 3.4. The *recovery* option is a reactive mechanism that engages when a forwarding node on a route encounters a network gap in the direction it intended to forward the data packet. When this happens, a new route is calculated and compared to the existing one. If the new route is estimated to be more reliable, then the data packet's route header is rewritten, and the packet is forwarded along the new route. The *recalculation* option is a proactive mechanism that evaluates a data packet's route when it is received by a forwarding node that is within range of a vertex. If the forwarding node determines that a significantly more reliable route is available, the route in the packet is overwritten, and the packet is forwarded along the new route. In addition, our performance evaluation also included a *combined* strategy that proactively recalculates routes and also reacts with the recovery option if a network gap is encountered. For comparison purposes, a *none* option was also evaluated where neither the proactive nor the reactive strategies were employed. Under this option, data packets are dropped when a network gap is encountered.

In Figure 3.11, we find that the recovery strategy delivers the best overall throughput, while the combined strategy performs nearly as well. The recalculation strategy only yields the best throughput in the sparsest of vehicle traffic conditions.

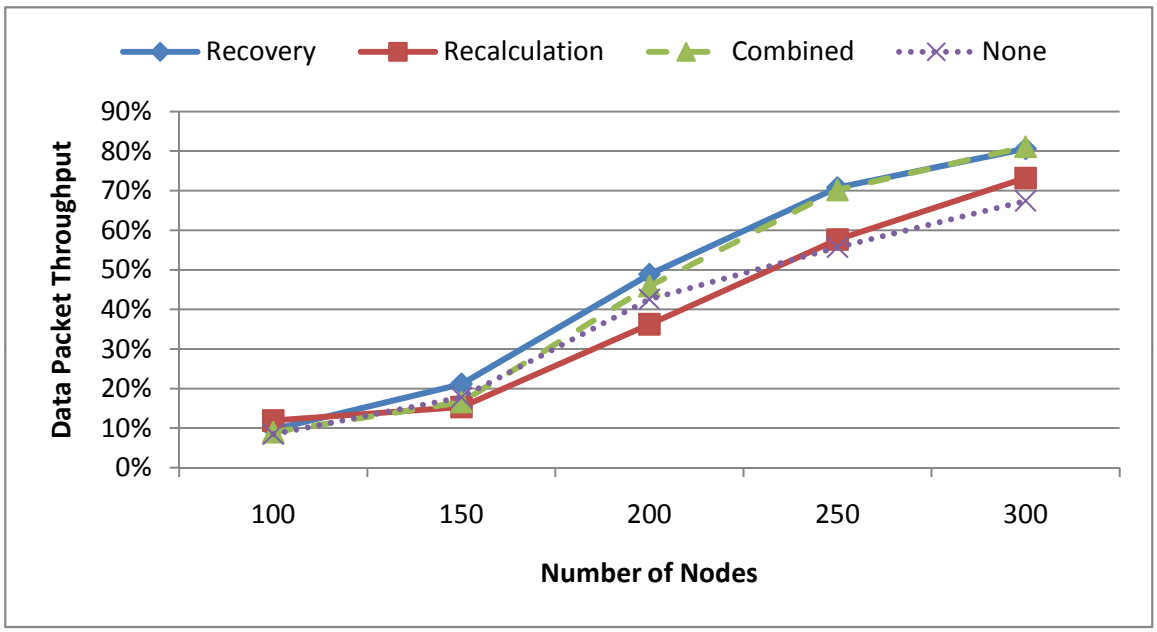


Figure 3.11: Data Packet Throughput with Recovery and Recalculation Strategies

In Figure 3.12, we observe that the recalculation strategy offers the smallest routing header size (excluding the *None* strategy), and the recovery and combined strategies produce nearly equal results in terms of routing header size.

In Figure 3.13, we find that the recalculation strategy requires the fewest number of hops per route (excluding the *None* strategy).

In Figure 3.14, we observe that the recalculation strategy delivers route packets more quickly than either the recovery or combined strategies.

3.5.4 Reliability Distribution

A significant component of our protocol is each node's ability to distribute reliability information about street edges through the use of mechanisms within beacon packets, probe packets, and routing headers. All of these messages may contain a known edge list to which the sending node and each forwarding node may contribute. In addition, routing headers also may include reliability weights for each edge represented in the encapsulated data packet's route. We evaluated the impact of these mechanisms on data packet throughput, routing hop count, and route header size.

In Figure 3.15, we observe that in average to dense traffic scenarios, active distribution of reliability information via the known edge list and weighted route mech-

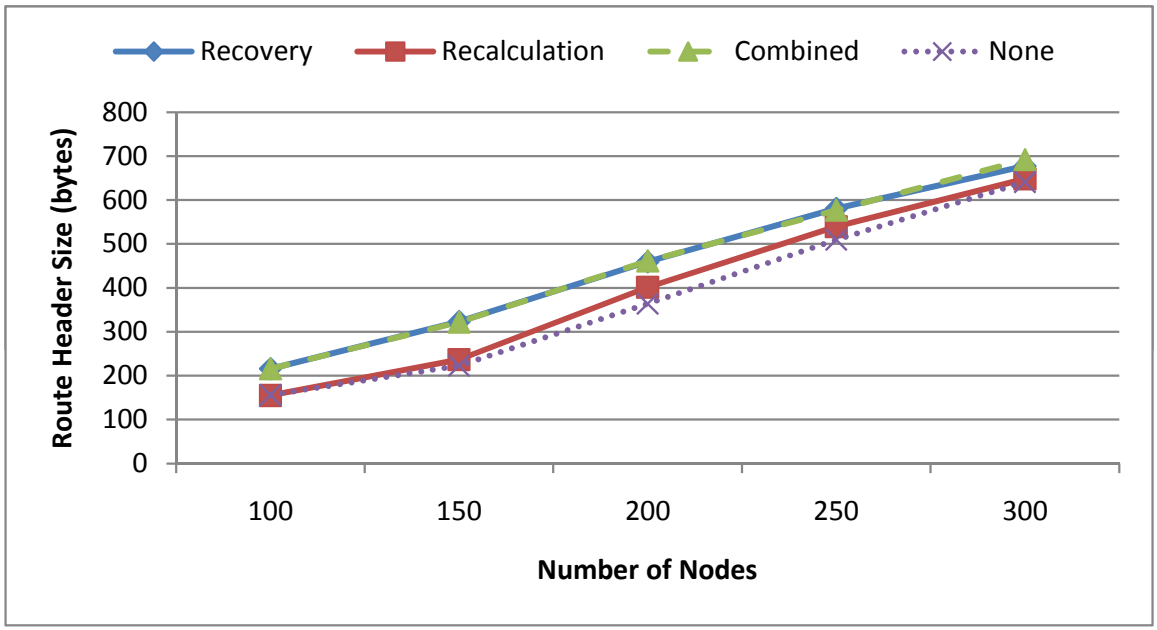


Figure 3.12: Route Header Size with Recovery and Recalculation Strategies

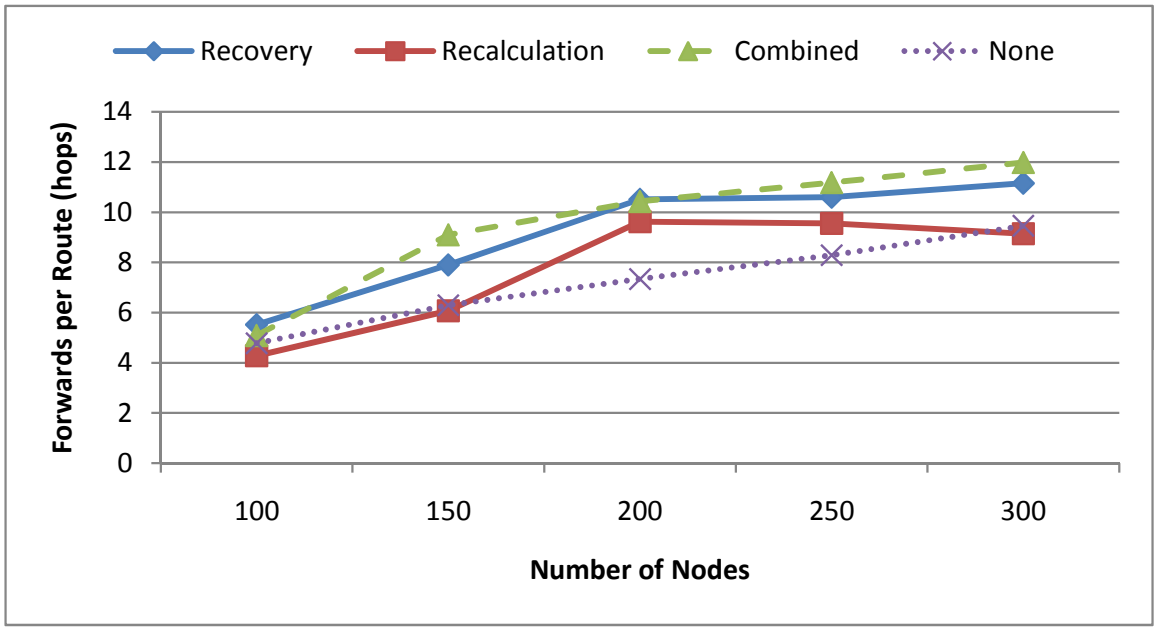


Figure 3.13: Forwards per Route with Recovery and Recalculation Strategies

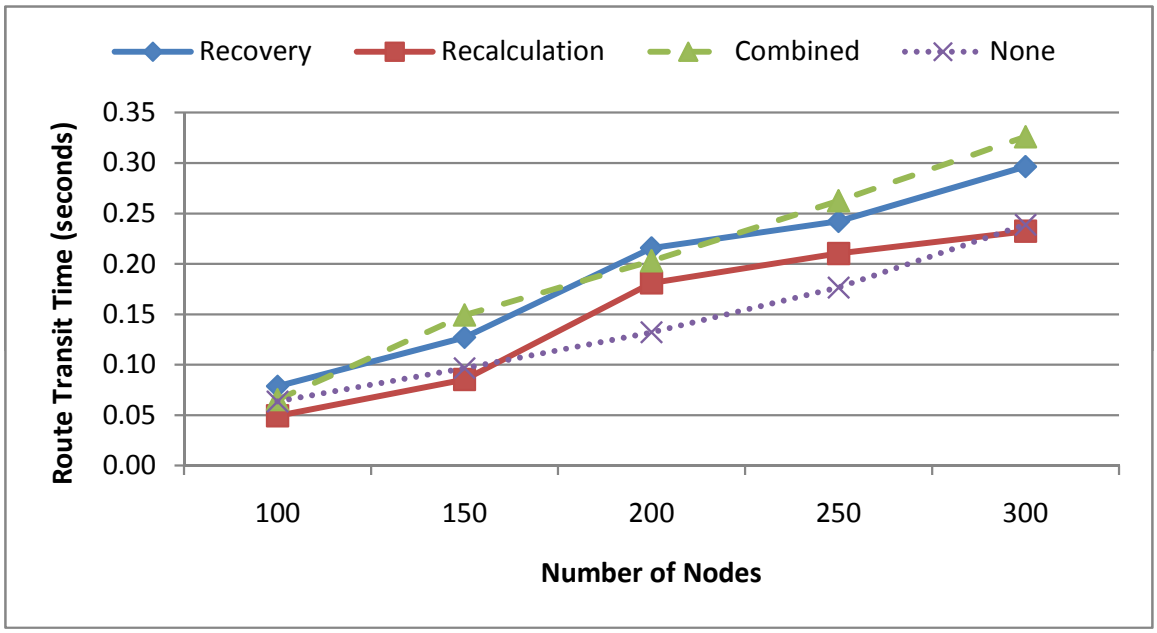


Figure 3.14: Route Header Size with Recovery and Recalculation Strategies

anisms contributes to a positive effect on data packet throughput. In sparse traffic scenarios, there is a small negative effect.

In Figure 3.16, we find that the number of hops per route increases by a small amount as active distribution mechanisms are added. We hypothesize two reasons for this. First, as we observed previously, the active reliability distribution produces higher data throughput. This means that more routes are reaching completion and therefore a greater number of hops are observed. Secondly, as streets are identified as unreliable and that data is distributed, fewer routes follow the shortest path, preferring reliable paths instead. A longer path requires a greater number of hops to reach its destination.

In this figure, we also observe that increased traffic density has a proportional effect on forwards per route. We hypothesize that increased traffic density is the primary driver here, as it allows for more reliable street edges that permit transmissions to succeed when the sender and receiver are farther apart.

In Figure 3.17, we observe that the use of known edge lists and weighted routes on routing packets has a significant effect on routing header size. We also find that active distribution of reliability information via the beacon and probe known edge list mechanisms has little effect on the route header size. (Although not observed from this graph, the beacon and probe known edge list mechanisms do increase beacon and

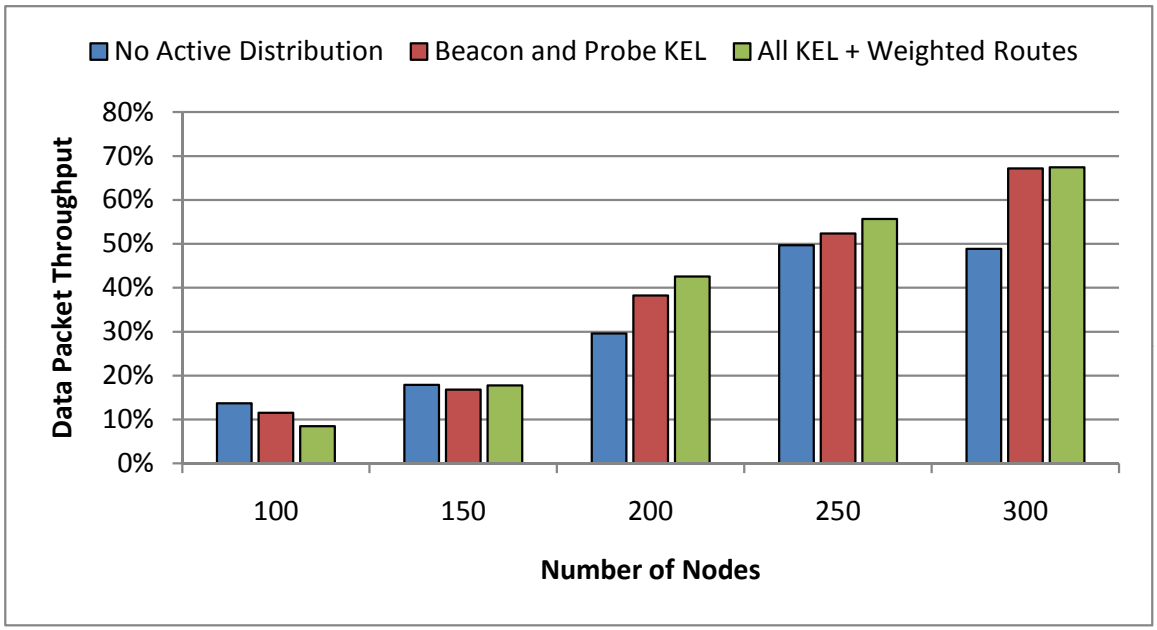


Figure 3.15: Effect of Active Reliability Distribution on Data Packet Throughput

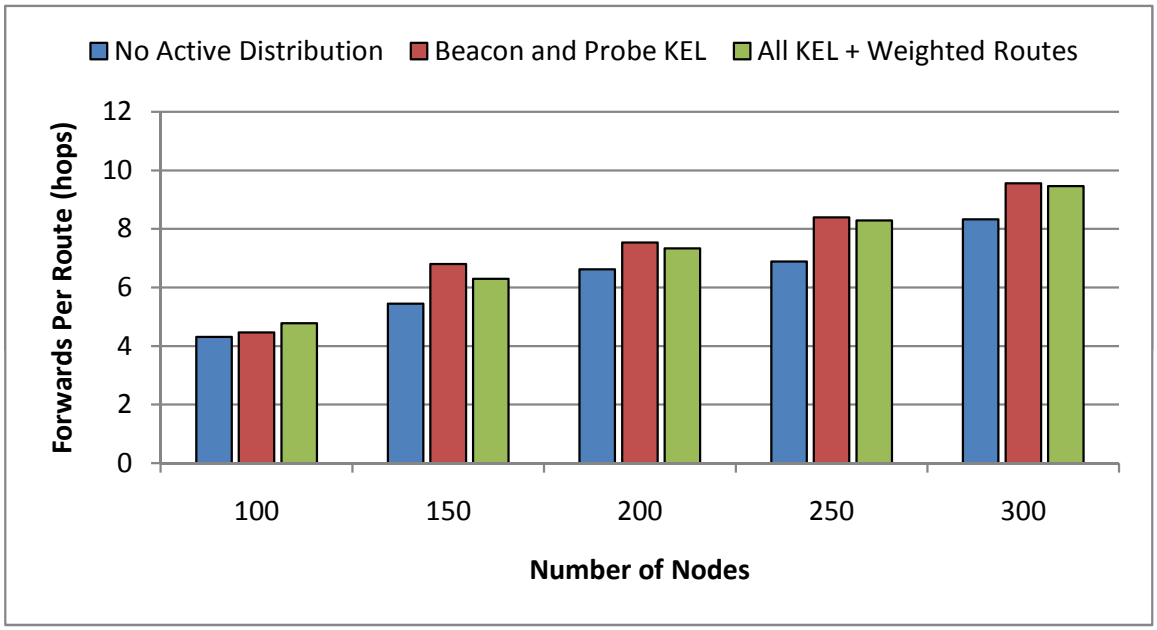


Figure 3.16: Effect of Active Reliability Distribution on Forwards Per Route

probe header size). We also note that a limit on the number of entries in a known edge list may be used to prevent the route header size from growing excessively if it is found to have a detrimental effect on throughput. Since throughput increases only marginally due to the addition of the known edge list and weighted routes within the routing packets, these functions could be eliminated from the routing packet with little apparent impact on throughput.

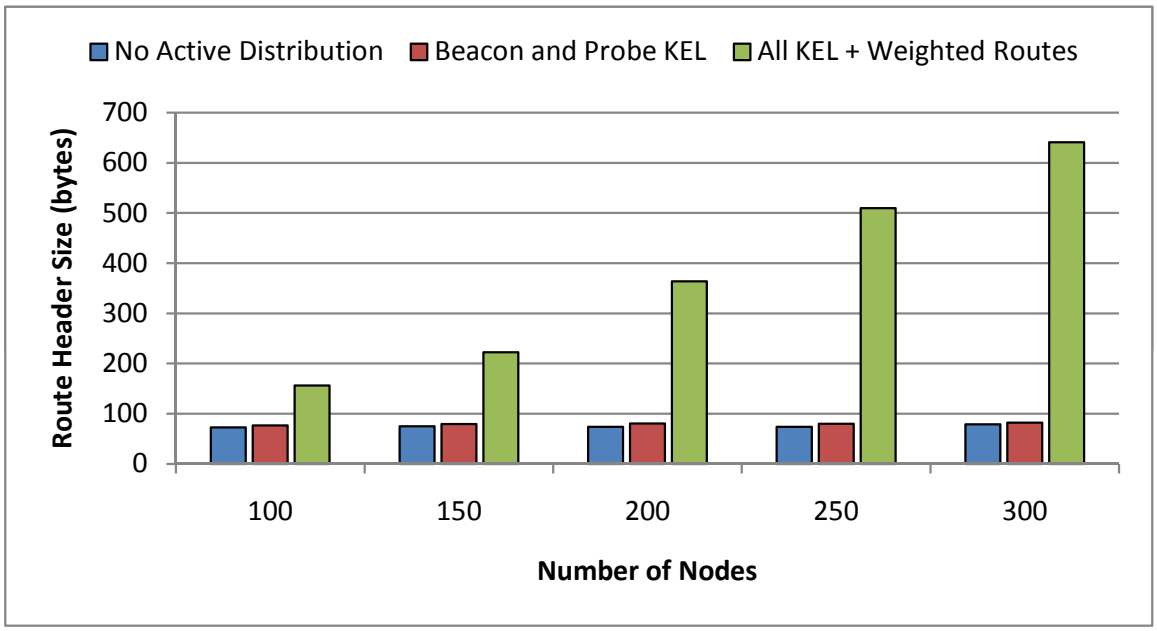


Figure 3.17: Effect of Active Reliability Distribution on Route Header Size

3.5.5 Probe Messages

To quantify the benefits of the active traffic monitoring system in RIVER, we have run four different sets of simulations. We simulate our protocol using two variations of the recovery strategy described above – without any probe messages transmitted and then with probes enabled. Then, we simulate the protocol using two variations of the recalculation strategy described above – without any probe messages transmitted and then with probes enabled.

In Figure 3.18, we can see that the recovery mechanism nearly nullifies the data throughput benefits of probe messages. However, a distinct positive effect on throughput is observed when the recalculation strategy is employed. Although not depicted in this graph, we have observed similar positive data throughput benefits from probe

messages in other scenarios, such as when neither the recovery nor recalculation mechanisms are enabled.

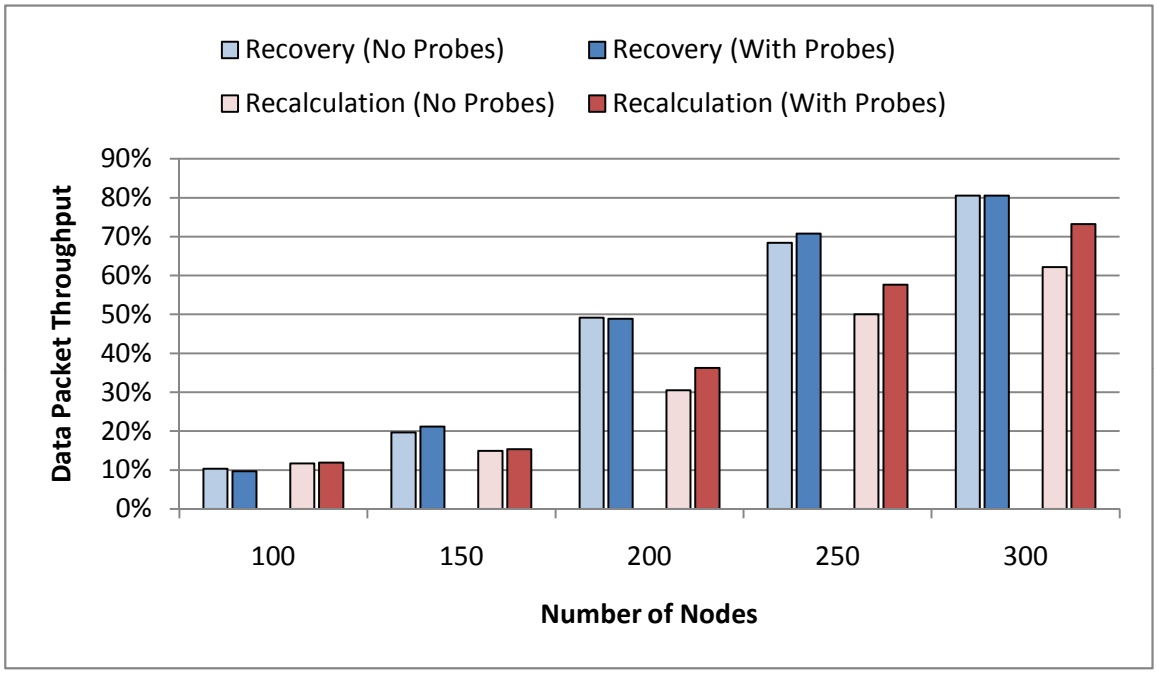


Figure 3.18: Effect of Probe Messages on Data Packet Throughput

3.5.6 Optimized Greedy Strategy

As described in Section 3.4, if the algorithm forwarded packets toward each anchor point in a strictly greedy manner, then some packets would be dropped inappropriately if no nodes were located within the zone of the anchor point but nodes were located around it. Our protocol uses an optimized greedy forwarding strategy (Section 3.4.8, Section 3.4.9) that detects these scenarios and handles them appropriately. We compared RIVER using its strict greedy forwarding strategy and the optimized strategy for comparison.

In Figure 3.19, we see that the optimized greedy strategy is beneficial to data throughput in every test, regardless of routing protocol or node density. We observe that the optimized greedy strategy is not merely useful for the RIVER protocol but also for a simple shortest-path greedy routing protocol as well.

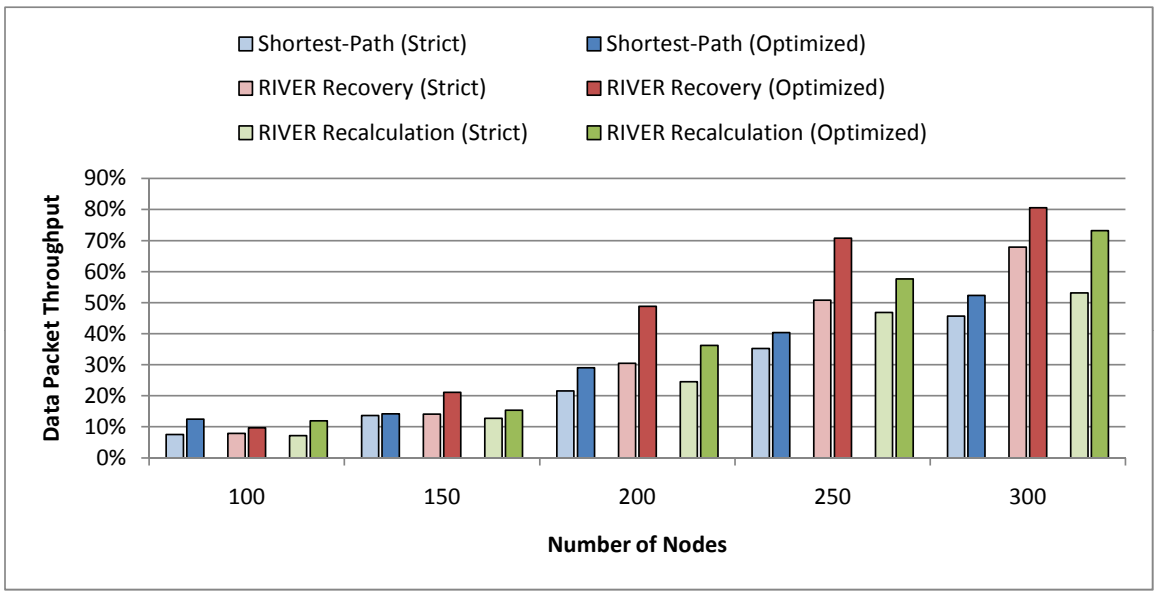


Figure 3.19: Effect of Optimized Greedy Strategy on Data Packet Throughput

3.5.7 Protocol Comparison

To determine how our protocol performs against its peers, we simulated RIVER and several other routing algorithms using the same suite of traffic density scenarios. We compared RIVER with the STAR routing protocol for VANETs [34], the GPSR geographic routing protocol [45], and a generic routing algorithm called Short-Path.

GPSR operates as described in Section 2.3.1 and Section 2.3.2. STAR is described in detail in Section 2.3.4. Both protocols use greedy forwarding and MAC layer link failure detection. STAR is designed specifically for VANETs, creates routes along streets, and contains a traffic monitoring component.

Short-Path generates greedy routes along streets using a pre-populated street map like RIVER, but it chooses its routes based on the shortest path available instead of reliability. Short-Path utilizes the optimized greedy strategy described in RIVER and also utilizes MAC layer link failure detection. Short-Path uses beacons and a neighbor-table to identify nearby nodes but has no traffic monitoring or data distribution components.

From Figure 3.20, we find that RIVER delivers data packet throughput up to 75% better than Short-Path (45% better on average), up to 157% better than GPSR (103% better on average), and up to 39% better than STAR (9% better on average).

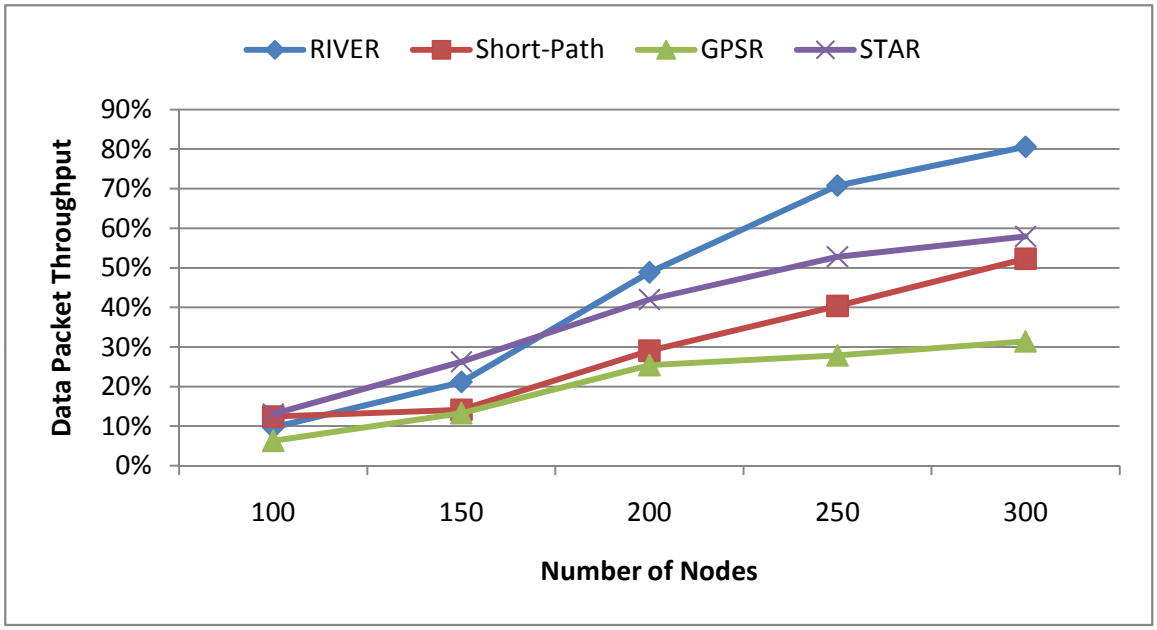


Figure 3.20: Peer Protocol Data Packet Throughput Comparison

Chapter 4 Conclusions and Future Work

In this thesis, we have proposed "Reliable Inter-Vehicular Routing" (RIVER), a VANET protocol for routing based on estimated network reliability. We presented the advantages of real-time traffic monitoring using active and passive methods. We submitted a metric for estimating the network reliability of streets and demonstrated how reliability-based traffic data can be effectively distributed throughout a VANET using known edge lists and weighted routes. We compared strategies of route recalculation and route recovery in RIVER and outlined their strengths and weaknesses.

In our simulation environment, we found that RIVER provides the highest throughput in most traffic density scenarios when using its recovery strategy, but the recalculation strategy yields higher throughput in low traffic density with less overhead in terms of hop count, routing header size, and transit time. We also found that RIVER's reliability distribution components perform best in average to high density scenarios. We observed that these components cause a significant increase in routing header size, and this cost can be effectively negated by only performing reliability distribution via beacon and probe packets. We compared the beneficial effect of probe messages on the recovery and recalculation versions of RIVER. We also learned that RIVER's optimized greedy forwarding strategy at anchor points can produce a significant increase in packet throughput with no known negative effects, and this strategy can be successfully applied to other routing protocols that do not share RIVER's reliable-path routing approach. Finally, simulations showed that RIVER performs well against peer protocols – especially in average to high-density traffic.

Additional improvements to RIVER may yield further benefits. At the time of writing, RIVER had never previously been tested on low-density traffic scenarios. Low-density traffic performance was not a focus point during its design, so this is an area where its performance could be enhanced. Additionally, performance evaluation revealed that routing header size could be greatly reduced without much loss of throughput by eliminating traffic distribution via routing packets. This should be investigated further as routing packets do disseminate information farther than other types of packets, and seeking a balance between range of distribution and network congestion seems wise.

Elimination of the need for a static street map could be added to the protocol. As discussed in Section 3.1.3, RIVER does not require common identifiers for street edges or vertices because it communicates in terms of geographical positions. This

design choice increases the potential for the elimination of the protocol's static street map without massive changes to the messaging scheme. The GPCR [57] protocol uses a neighbor-table approach and a correlation-coefficient approach for detecting street intersections. CAR [63] uses a velocity-vector approach to detect changes in a packet's direction during its route. These are promising approaches, and the use of probe messages in RIVER may provide another method for detecting streets and intersections. However, as vehicular navigation systems become more ever-present, an external static map seems to be a reasonable requirement.

While in the current implementation, a probe message traverses only a single edge of the street graph, they could conceivably traverse multiple edges for the purpose of retrieving information from (and distributing data to) a greater area. Note that messages must return in a relatively short amount of time in order to return to their original sender before that vehicle moves too far away from its original position. To ensure this, a distance or time limit could be imposed on the probe. Also in the case of a multi-edge probe, if a node that is forwarding that probe has no neighbors in the specified direction (local maximum) and the probe has already traversed at least one edge, the node could simply return the probe instead of dropping it, and useful information would still be gained from the probe on its return trip.

Some security issues could be explored, although these are outside the scope of a particular routing protocol and should be investigated in a more generalized form. When a RIVER node accepts a beacon, probe, or routing message, it does so with the assumption that all other nodes are trustworthy and no packet tampering has occurred. Instead, the protocol could be made more secure if some form of validation was performed before accepting declared edge data.

Other miscellaneous enhancements could be evaluated in future work. First, the recovery strategy in RIVER has been shown to yield great benefits in throughput. When recovery fails, the protocol might be improved by sending a failure message back to the packet's sender to allow their street graph reliability information to be adjusted accordingly. Second, the addition of velocity and heading information in RIVER beacon messages should be considered. This information would allow each node to predict the location of its neighbors between beacon intervals. These predictions could lead to better next-hop selections and the reduction of repeat-node loops. Third, dynamic adjustment of beacon frequency based on each node's number of known neighbors as suggested in CAR may help reduce network congestion in high traffic scenarios. Finally, a sending RIVER node currently assumes that the nearest vertex is a suitable place to begin an anchor route, but this is not always true. It may be

beneficial for each node to also track whether the vertices at either end of the current street edge are reachable. Then if one of these vertices was unreachable, the node could recalculate the anchor route from the opposite vertex.

A generalized MANET packet/message format has been recently proposed to the Internet Engineering Task Force (IETF) [24]. Among its features are a mechanism for defining variable-sized packet fields (as required for supporting both IPv4 and IPv6 address fields). It also provides a feature that could allow several RIVER messages to be transported in a single packet. This could be useful, for example, when a node is forwarding several RIVER messages to the same node. Conforming the RIVER message formats into this generalized format would make an interesting exercise.

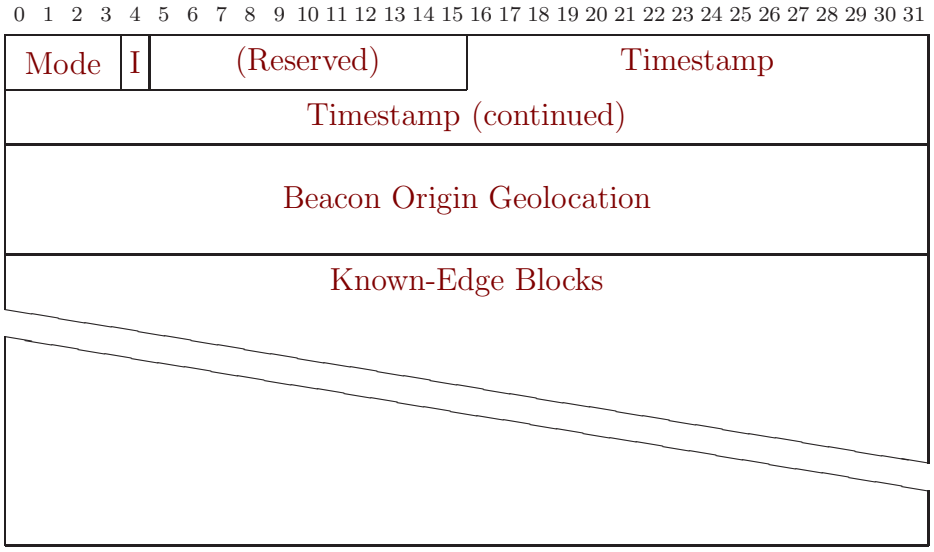
Further simulations for RIVER would provide additional information into the protocol's performance. It would be interesting to learn how the protocol handles varied street topologies. Simulations currently assume an urban scenario, but suburban and highway scenarios will also be encountered. Simulations using a map of a real location and actual vehicle traffic data would likely yield additional insights.

While VANETs are an exciting area of research, they are not yet a practical reality. RIVER provides a glimpse into the potential of reliability-based metrics for routing packets within a VANET and demonstrates convincing performance for high throughput within the VANET paradigm.

APPENDICES

Appendix A RIVER Protocol Messages

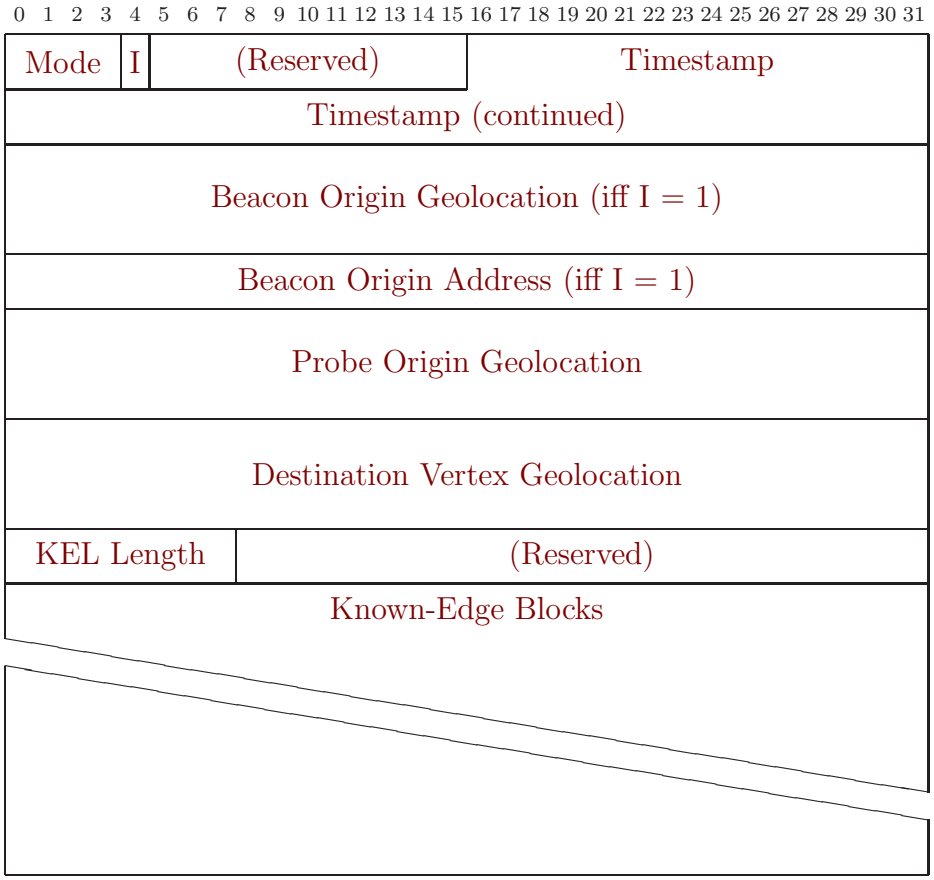
A.1 Explicit Beacon Message Diagram



Explicit Beacon Message Packet

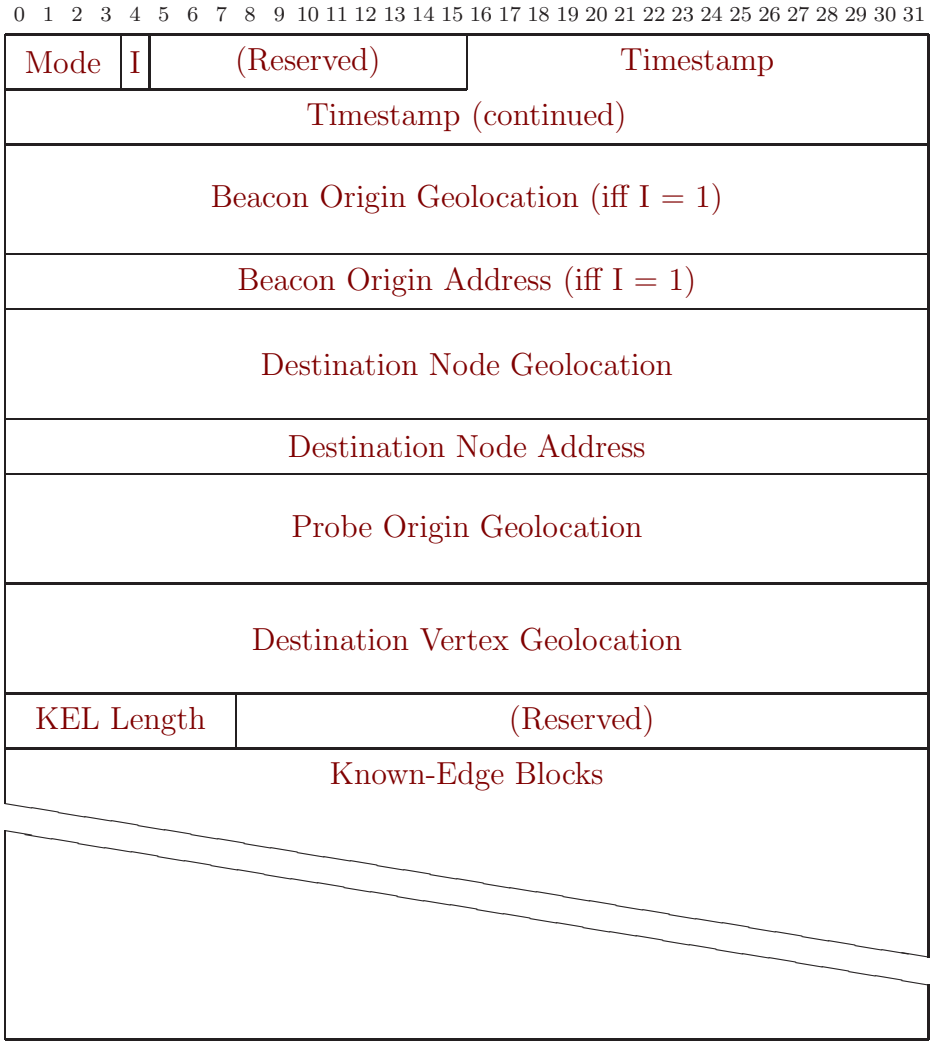
In the RIVER beacon packet, note that the known-edge blocks are not preceded by a known-edge list length because there are no other fields that follow the known-edge list in this packet. Therefore, the list ends when the packet ends. Also, note that no explicit beacon origin address is provided because that would be redundant with information stored in the Internet layer encapsulating packet.

A.2 Probe Message Diagrams



Departing Probe Message Packet

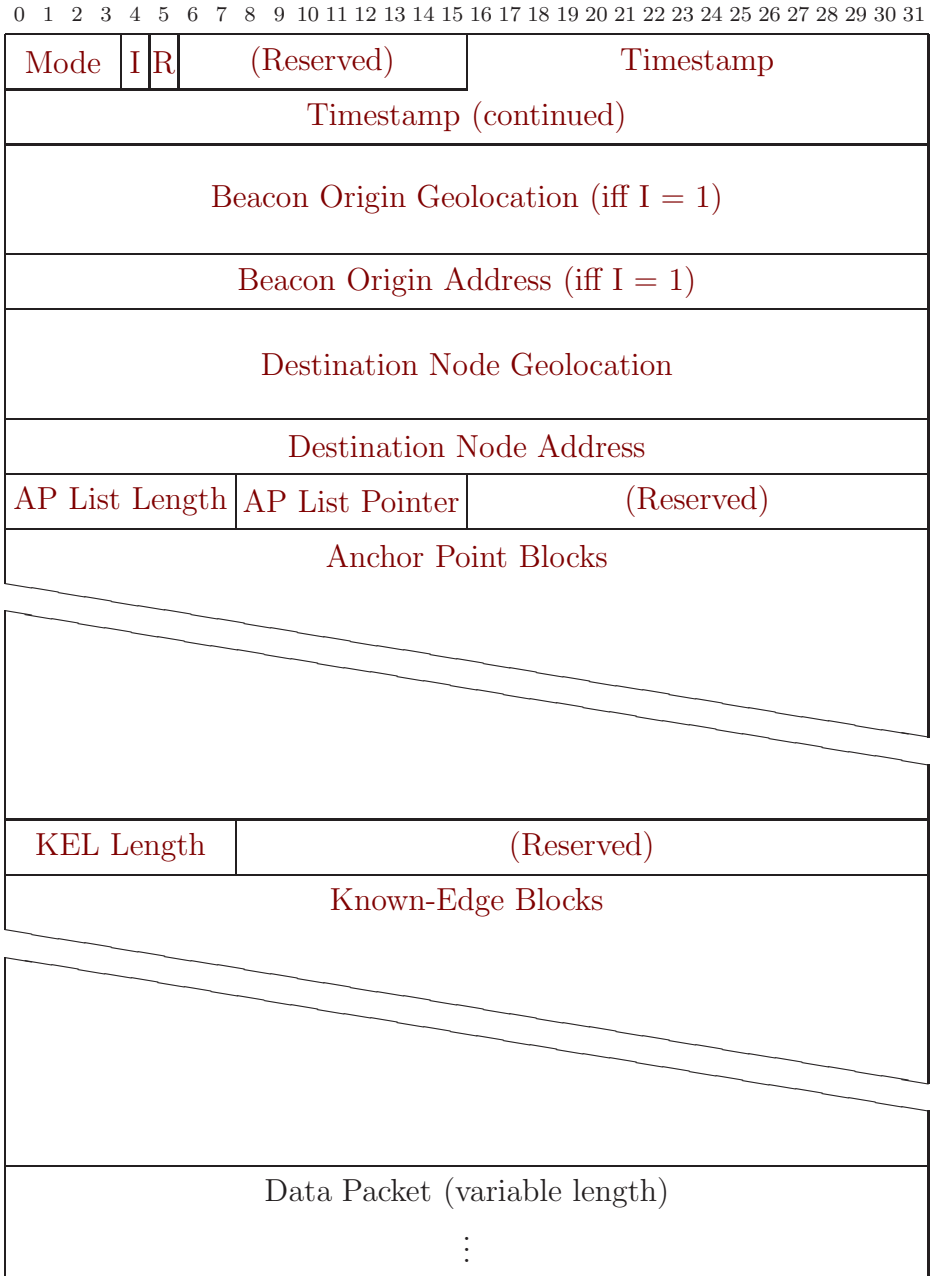
The Probe Origin Geolocation field represents the vertex nearest to the probe origin; this is the starting vertex of the probed edge.



Return Probe Message Packet

Note that return probes have two additional fields, the Destination Node Geolocation and Destination Node Address. These fields provide the endpoint of the probed edge and are required for return probes only since departing probes are not destined for a specific target node. Note also that the destination geolocation is different from the destination vertex geolocation because the former field represents the last-known position of the node to receive this returning probe while the latter represents the vertex endpoint for the probed edge. The former position may change as forwarding nodes update the packet with the destination node's current location. In the case that the receiving node gets closer to some other vertex than it was originally located at, the Destination Vertex Geolocation field preserves the original vertex location.

A.3 Routing Header Diagram



Routing Header / Data Packet

A.4 Data Blocks and Fields

This section describes in detail the data blocks and fields that appear in the packet diagrams above.

A.4.1 Mode

The mode field indicates the kind of RIVER packet that will follow.

Mode Field (Mode)

Mode	Meaning
0	Explicit Beacon
1	Departing Probe
2	Returning Probe
3	Routing

A.4.2 Implicit Beacon Flag (I)

When the implicit beacon flag is set to 0, no implicit beacon information is included in the packet. For a beacon packet (ie. mode = 0), the implicit beacon flag is always set to 0 because the packet itself is a beacon.

When the implicit beacon flag is set to 1 (on probe packets and/or routing packets), the beacon origin geolocation and beacon origin address blocks are included in the packet in the positions specified in the packet diagrams above.

Implicit Beacon Flag (I)

I	Meaning
0	No implicit beacon information is included in the packet.
1	The "Beacon Origin Geolocation" and "Beacon Origin Address" blocks are included in the packet.

A.4.3 Route Reliability Flag (R)

The route reliability flag (R) indicates whether reliability and relative age are included for each anchor point block in the routing packet. Reliability information applies to street edges (not street vertices), so the Anchor Point Reliability and Relative Age fields only occur between two anchor point geolocation blocks. Therefore, if there is

only one AP present in the route, no reliability or relative age information would be present in the packet, regardless of the value of the route reliability flag.

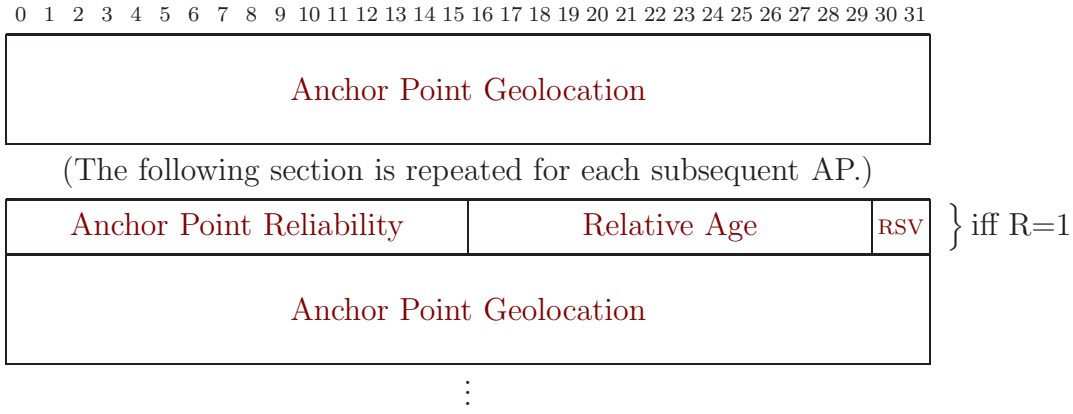
This flag was used in simulations for the purpose of quantifying any advantage or disadvantage to the inclusion of reliability information in routing packets. In a real-world implementation, it is likely that this flag would be eliminated and reliability information would always be included.

Route Reliability Flag (R)

R	Meaning
0	Anchor Point blocks contain geolocation information only.
1	Anchor Point blocks contain reliability information and relative age, in addition to geolocations.

A.4.4 Anchor Point Block

Anchor point blocks form the chain of geographical locations that make up a route in the RIVER protocol. Reliability information applies to street edges, so the Anchor Point Reliability and Relative Age fields only occur between two anchor point geolocation blocks. That is, there is always one less reliability/relative age block than the number of anchor point geolocation blocks.



Anchor Point Block Diagram

A.4.5 AP List Length

The AP List Length field represents the number of anchor points that follow in the Anchor Point Block. Based on the current size of this field, the maximum number of

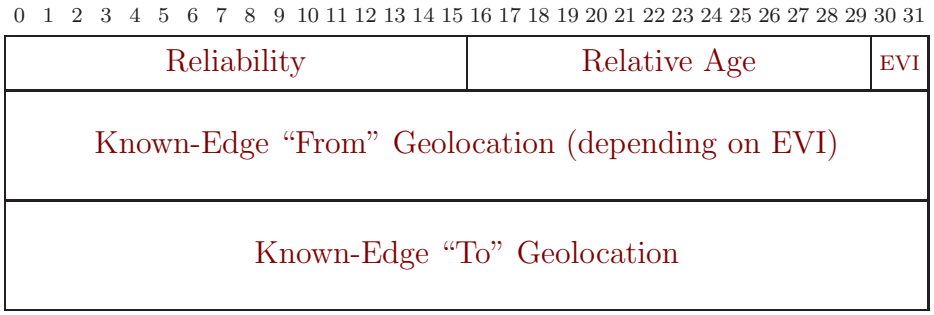
anchor points in the route is 256.

A.4.6 AP List Pointer

The AP List Pointer represents the current position in the anchor point list as the packet is forwarded from node to node.

A.4.7 Known-Edge Block

The known-edge list identifies edges by their endpoint geolocations and communicates reliability information about each edge.



Known-Edge Block Diagram

A.4.8 Edge Vertex Indicator (EVI)

For the purpose of allowing the known-edge list to be compressed as much as possible, a 2-bit value known as the Edge Vertex Indicator (EVI) is included in each known-edge block. For the current edge (e), the EVI indicates which of the vertices of e are explicitly defined in the next part of the block. Note that since RIVER street graph edges are undirected, we do not need to consider the "to-side vertex" of e because we can always arrange the vertices of e so that the vertex in common with the previous edge is the "from" vertex of e. Also, note that it is redundant to have the same undirected edge listed twice in the list with "from" and "to" vertices reversed, since they would both represent the same edge. The meaning of each EVI value is shown in the table below.

A.4.9 KEL Length

The Known Edge List Length field, marked as KEL Length, represents the number of known-edges that are represented in the list. Based on the current size of this field, the maximum number of known-edges per list is 256.

Edge Vertex Indicator (EVI)

EVI	Meaning
0	Both of the vertices of edge e are explicitly defined.
1	The "from" vertex of e is the same as the previous edge's "from" vertex, and therefore, the "from" vertex of e is not present in the block.
2	The "from" vertex of e is the same as the previous edge's "to" vertex, and therefore, the "from" vertex of e is not present in the block.

A reader or implementor should not confuse the value of the Known-Edge List Length field with the number of geolocation blocks that will follow in the known-edge list. The Known-Edge List Length is a count of the number of known-edges that are represented in the list. Since this is a count of edges, the number of reliability/relative age blocks should match this number exactly. Suppose n represents the value of the Known-Edge List Length field. The number of geolocation blocks is not necessarily equal to $2n$ since we can chain edges with EVI to reduce the packet size.

A.4.10 Timestamp

The Timestamp field contains a positive integer that represents the number of milliseconds since the UNIX epoch (00:00:00 UTC on 1 January 1970). This is an absolute timestamp to use in conjunction with relative times elsewhere in the packet for absolute time calculations. The Timestamp field is a 48-bit field to provide a sufficient range of time, as shown here.

Let m equal the number of milliseconds in a year.

$$m = 365.24 * 24 * 60 * 60 * 1000 = 3.1557 \times 10^{10}\,\text{ms/year} \tag{A.1}$$

Let r_b equal the largest b-bit positive integer to represent the number of milliseconds since the epoch and y_b equal the range of years allowed by an r_b ms size positive integer.

$$r_{40} = 2^{40} - 1 = 1.0995 \times 10^{12} \tag{A.2}$$

$$y_{40} = r_{40}/m = 34.8\,\text{years} \tag{A.3}$$

$$r_{48} = 2^{48} - 1 = 2.8147 \times 10^{14} \tag{A.4}$$

$$y_{48} = r_{48}/m = 8,919.6\,\text{years} \tag{A.5}$$

$$r_{56} = 2^{56} - 1 = 7.2057 \times 10^{16} \tag{A.6}$$

$$y_{56} = r_{56}/m = 2,283,429.9\,\text{years} \tag{A.7}$$

$$r_{64} = 2^{64} - 1 = 1.8446 \times 10^{19} \tag{A.8}$$

$$y_{64} = r_{64}/m = 584,558,050.4\,\text{years} \tag{A.9}$$

From this, we observe that a 40-bit millisecond timestamp would have already expired in the year 2004. However, since the continued use of VANET routing protocols in the year 10089 CE seems unlikely, we deem the 48-bit timestamp sufficient for our purposes.

Regarding Global Clocks

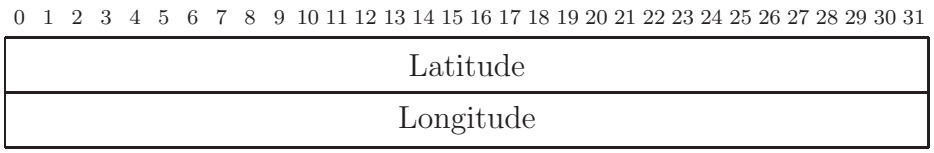
Readers who are familiar with basic principles of distributed systems may question the use of timestamps in the RIVER protocol. Much work has been done in the area of clock synchronization in distributed systems, including the classic paper[50]. While we cannot avoid the facts of clock drift and the absence of a true global clock, the timing needs of the RIVER protocol can be met as follows.

In the RIVER model, we assume that all vehicles are equipped with a global positioning system (GPS). GPS systems already depend upon accurate clocks, synchronized via satellite to within 1 microsecond. Indeed, NTP (network time protocol) considers a GPS clock to be a stratum zero clock. RIVER only requires clock synchronization accuracy within 1 millisecond since that is the most granular unit of a RIVER timestamp. Therefore, we reason that GPS time is more than sufficient for our purposes.

A.4.11 Geolocation

Each geolocation block consists of a floating-point number representing latitude $\{-90, 90\}$, and another representing longitude $\{-180, 180\}$. Latitude and longitude are measured in decimal-degrees notation.

Five digits of precision of a degree for latitude and longitude will provide an accuracy within about one meter of a physical location on Earth, (The exact proximity depends upon the degree of latitude since lines of longitude are more distant from each other near the equator than at the poles.) This level of accuracy is probably sufficient



Geolocation Block Diagram

for VANET applications, but six digits of precision enters within 10 cm, and seven digits enters within 1 cm, if further accuracy is needed. Since a typical lane of vehicle traffic ranges from 3 to 5 meters, having four digits of precision (accuracy within 10 meters) is not sufficient for our purposes, as it would not allow us to distinguish one vehicle's position from another if they were adjacent to each other.

The typical 32-bit "float" yields only seven digits of significance, which leaves only four digits of precision on longitude n where $(n \leq -100)$ or $(n \geq 100)$. (For example, 179.9999 has seven significant digits and four digits of precision.) However, this same 32-bit space can yield up to nine full digits of significance if fixed-point representation is used. This would provide longitudinal accuracy to within 10cm on Earth, so we assume that 32 bits for each coordinate in our packet is sufficient. (Note that the range for latitude $-90, 90$ uses one less significant digit than the range for longitude $-180, 180$.)

Regarding Altitude

The VANET protocols we have studied in the literature assume that the differences in elevation of neighboring vehicles are negligible, and therefore they do not take altitude into account. Likewise, RIVER packets do not incorporate altitude into geolocations, but we have performed some analysis of how altitude might be efficiently stored in a data packet.

Altitude is measured in meters above or below sea level. It is logical to assume we would measure altitude (in meters) with the same level of accuracy as our latitude and longitude coordinates. In that case, a 32-bit space (using fixed-point representation) would yield:

1. At 1 meter accuracy: a range of about ± 1,000,000 km
2. At 10 cm accuracy: a range of about ± 100,000 km
3. At 1 cm accuracy: a range of about ± 10,000 km

Since the ceiling altitude of the highest layer of Earth's atmosphere, the exosphere, is 10,000 km, and the radius of the earth is less than 7,000 km, we deem a 32-bit representation sufficient at 1 cm accuracy.

A.4.12 Reliability

The reliability field is a 16-bit positive integer field representing the reliability of an edge.

A.4.13 Relative Age

The relative age field is a 14-bit positive integer field representing a point in time relative to the packet's absolute timestamp. This strategy is used to reduce packet size. For each edge that we are distributing reliability information, we must also distribute a timestamp so that consumers of the reliability data can have knowledge of the "freshness" of that reliability data. An absolute timestamp requires 48 bits, but we can provide a range of time up to 16.384 seconds with 1 millisecond accuracy in a 14-bit relative timestamp.

A.4.14 Beacon Origin Address

This field contains the address of a beaconing node for a packet with its implicit beacon flag on. While RIVER is not coupled to a particular Internet layer, we assume IPv4 and use a 32-bit address here. However, if IPv6 is used, the address length may need to be extended. Another possibility would be to use a special field within the Internet layer protocol to store this information.

In explicit beacon mode, the beacon address can be retrieved from an Internet layer (eg. IPv4) header because the packet travels only one hop, so the IP header is never modified. In multi-hop cases such as probe modes or routing mode, the RIVER packet could be placed inside a different IP header at each hop (and a probe or route origin address would go in this space), or the beacon origin/address could be updated by each forwarding node while the IP header remains intact. In these cases, an origin address needs to be explicitly included.

A.4.15 Destination Node Address

This field contains the address of a return-probe's or routing packet's ultimate destination node. While RIVER is not coupled to a particular Internet layer, we assume IPv4 and use a 32-bit address here. However, if IPv6 is used, the address length may

need to be extended. Another possibility would be to use a special field within the Internet layer protocol to store this information. It is assumed that the destination address field of the Internet layer packet would be used to store the next-hop node's address, and therefore it is already in use.

A.4.16 Reserved (RSV)

This field, marked Reserved or RSV, represents bits that are reserved for potential future use and to round out a word evenly.

Appendix B RIVER Pseudocode

B.1 Receiving Packets

Algorithm 1 Receive

```
    ▷ This procedure is called when this node receives a packet on its network interface
    from some other node.
 1: procedure RECEIVE(pkt)
 2:     if pkt.type = RIVER then
 3:         PROCESSKNOWNEDGELIST(pkt)
 4:         PROCESSBEACONINFO(pkt)  ▷ Could be explicit or implicit.
 5:         if pkt.mode ∈ {PROBE-DEPART, PROBE-RETURN} then
 6:             orgn = pkt.probeOriginLoc
 7:             dest = pkt.destLoc
    ▷ Two street vertices' ranges can overlap. Don't consider the destination reached if
    we are closer to the origin than the destination.
 8:             if INRANGE(dest) ∧ (DISTANCE(dest) < DISTANCE(orgn)) then
 9:                 PROCESSINCOMINGPROBE(pkt)
10:                 return
11:             end if
12:         else if pkt.mode = ROUTING then
13:             PROCESSROUTEWEIGHTS(pkt)
14:             nearVtx ← NEARESTSTREETVERTEX
15:             if INRANGE(nearVtx) then  ▷ We are "at" a vertex.
    ▷ Our receipt of this packet implies that an incident edge is reliable.
16:                 TRAVERSEDROUTEEDGE(pkt)
17:             end if
18:             if ACCEPTDATAPKT(pkt) then
19:                 return
20:             end if
21:         end if ▷ packet must be outgoing at this point
22:         FORWARDPACKET(pkt)
23:     end if
24: end procedure
```

Algorithm 2 AcceptDataPkt

```
    ▷ This function returns TRUE if the packet should be accepted (consumed) by this
    node. Returns FALSE if the packet's delivery should continue.
 1: function AcceptDataPkt(dataPkt)
 2:     if dataPkt.destAddr = IP-BROADCAST then
 3:         RecvDataPkt(dataPkt)  ▷ but continue broadcast
 4:         return FALSE
 5:     else if dataPkt.destAddr = RIVER-GEOANYCAST then
 6:         if InRange(destLoc) then
 7:             RecvDataPkt(dataPkt)
 8:             return TRUE
 9:         else ▷ continue delivery
10:             return FALSE
11:         end if
12:     else if dataPkt.destAddr = MyAddress() then
13:         RecvDataPkt(dataPkt)
14:         return TRUE
15:     end if
16:     return FALSE
17: end function
```

B.2 Sending Packets

Algorithm 3 PrepareDataPacket

▷ This procedure is called when this node originates a packet at a higher level (eg. application layer) for sending on the network interface.

```
1:  function PrepareDataPacket(dataPkt)
2:      dataPkt ← AddRiverHeader(dataPkt)
3:      dataPkt.destPort ← RIVER-PORT
4:      dataPkt.type ← RIVER
5:      dataPkt.mode ← ROUTING
6:      dataPkt.destLoc ← LocationLookup(dataPkt.destAddr)
7:      if ¬ InRange(dataPkt.destLoc) then ▷ compute a route
8:          path ← LeastWeightPath(myLoc, destLoc)
9:          dataPkt.SetAnchorRoute(path)
10:     end if
11:     dataPkt ← AddIPHeader(dataPkt)
12:     ForwardPacket(dataPkt)
13: end function
```

Algorithm 4 ForwardPacket

Require: $fwdPkt$ is a packet that is not destined for this node

1: **procedure** FORWARDPACKET($fwdPkt$)
2: $myLoc \leftarrow$ CURRENTLOCATION
3: $destLoc \leftarrow fwdPkt$.DESTLOCATION
4: $nextHop \leftarrow \emptyset$ ▷ Reset the next hop for the packet
5: **if** $fwdPkt.destAddr =$ IP-BROADCAST **then**
6: $nextHop \leftarrow$ IP-BROADCAST ▷ Beacons are broadcast
7: **else if** $fwdPkt.destAddr =$ RIVER-GEOANYCAST **then** ▷ packet is a probe
8: $nextHop \leftarrow$ GREEDYFORWARDTO($destLoc$)
9: **else** ▷ packet is unicast; use RIVER routing
10: $nextHop \leftarrow$ NEXTANCHORROUTEHOP($fwdPkt$)
11: **end if**
12: **if** $nextHop = \emptyset$ **then** ▷ no neighbor was found to forward to
13: DROPPACKET($fwdPkt$)
14: **return**
15: **end if**
16: $fwdPkt$.SETNEXTHOP($nextHop$)
▷ Intermediate forwarding nodes update the packet's destination node's geolocation in the packet with their last known location.
17: UPDATEDESTGEOLOC(pkt)
18: $fwdPkt$.SETIMPLICITBEACON($myLoc$, MyAddress())
19: RESETBEACONTIMER
20: TRANSMIT($fwdPkt$)
21: **end procedure**

Algorithm 5 NextAnchorRouteHop

Require: $fwdPkt$ is a packet that is not destined for this node

```
1: procedure NextAnchorRouteHop(fwdPkt)
2:     myLoc ← CurrentLocation
3:     destLoc ← fwdPkt.DestLocation
4:     anchor ← fwdPkt.GetAnchor(fwdPkt.ptr)
   ▷ Advance to the next unreached AP in the route.
5:     while InRange(anchor) ∧ anchor ≠ fwdPkt.LastAnchor do
6:         ConsiderRouteRecalculation(fwdPkt)
7:         anchor ← fwdPkt.AdvanceAnchor
8:     end while
   ▷ If we are past the current anchor point, advance to the next AP.
9:     nextAnchor ← fwdPkt.GetAnchor(fwdPkt.ptr + 1)
10:    if (180 - DegOfAngle(anchor, myLoc, nextAnchor)) < 20 then
11:        anchor ← fwdPkt.AdvanceAnchor
12:    end if
   ▷ Choose the next hop based on the current anchor.
13:    nextHop ← GreedyForwardTo(anchor)
14:    if fwdPkt.AnchorsLeft > 0 then
   ▷ If that fails, try finding a neighbor between the current anchor and the next.
15:        if nextHop = ∅ then
16:            nextAnchor ← fwdPkt.GetAnchor(fwdPkt.ptr + 1)
17:            nextHop ← GreedyForwardOnEdge(anchor, nextAnchor)
18:        end if
19:        if nextHop = ∅ then  ▷ Next hop still not found. Try recovery.
20:            prevAnchor ← fwdPkt.GetAnchor(fwdPkt.ptr − 1)
21:            MarkFailedEdge(prevAnchor, anchor)
22:            path ← LeastWeightPath(myLoc, destLoc)
23:            fwdPkt.SetAnchorRoute(path)
24:            if LeadingEdgeNotFailed(fwdPkt.route) then
25:                nextHop ← NextAnchorRouteHop(fwdPkt)
26:            end if
27:        end if
28:    end if
   ▷ Last effort: is the destination one of our neighbors and we didn't find it greedily
   because we think we are closer to the destination position?
29:    if nextHop = ∅ ∧ IsNodeANeighbor(fwdPkt.destAddr) then
30:        nextHop ← fwdPkt.destAddr
31:    end if
32:    return nextHop
33: end procedure
```

B.3 Beacons and Probes

Algorithm 6 SendRiverBeacon

```
    ▷ This procedure is called by the beacon timer at a jittered time interval.
 1: procedure SendRiverBeacon
 2:     riverPkt ← PrepareBeacon
 3:     if riverPkt ≠ NULL then
 4:         Transmit(riverPkt)
 5:     end if
 6: end procedure
```

Algorithm 7 SendRiverProbe

```
    ▷ This procedure is called by the probe timer at a jittered time interval.
 1: procedure SendRiverProbe
 2:     nearVtx ← NearestStreetVertex
 3:     if InRange(nearVtx) then  ▷ We are "at" the vertex.
    ▷ Find the adjacent edge whose reliability rating is most out-of-date.
 4:         oldest ← MAX-AGE
 5:         for all edge e incident to nearVtx do
 6:             if e.IsOutdated ∧ e.LastUpdated < oldest then
 7:                 oldest ← e.LastUpdated
 8:                 probEdge ← e
 9:             end if
10:         end for

11:         now ← CurrentTime
12:         if oldest ≠ MAX-AGE then
    ▷ If we found an edge that needs updating, send a probe.
13:             riverPkt ← PrepareProbe(probEdge.fromVtx, probEdge.toVtx,
    RIVER-GEOANYCAST)
14:             ForwardPacket(riverPkt)
15:         end if
16:     end if
17: end procedure
```

Algorithm 8 ProcessBeaconInfo

Require: *beaconPkt* is an explicit beacon

```
1: procedure ProcessBeaconInfo(beaconPkt)
2:     address ← beaconPkt.GetBeaconSourceAddress
3:     location ← beaconPkt.GetBeaconSourceLocation
4:     UpdateNeighborTable(address, location)
5: end procedure
```

Algorithm 9 ProcessIncomingProbe

Require: The current node is within range of the destination of the probe and is closer to the destination vertex of the probe than to the origin vertex of the probe. (Vertex range of the endpoints of an edge may overlap, and we don't want to consider the current node at the destination if it is closer to the origin vertex than the destination vertex.

```
1: procedure ProcessIncomingProbe(probePkt)
2:     destVtx ← probePkt.GetDestination
3:     originVtx ← probePkt.GetProbeOrigin
4:     probedEdge ← streetDb.GetEdgeByEndpoints(originVtx, destVtx)

5:     if probePkt.mode = PROBE-DEPART then
   ▷ Generate a return probe back to the sender.
6:         addr ← probePkt.GetProbeOriginAddress
7:         riverPkt ← PrepareProbe(destVtx, originVtx, addr)
8:         Transmit(riverPkt)
9:         streetDb.ProbeSent(probedEdge)
10:    end if

   ▷ Update street edge with received probe
11:    edgRcvd ← GetEdgeByVertices(originVtx, destVtx)
12:    streetDb.PacketReceived(edgRcvd)
13: end procedure
```

Bibliography

[1] S. Ahmed, S. S. Kanere, SKVR: Scalable Knowledge-Based Routing Architecture for Public Transport Networks, in: Proceedings of the 3rd International Workshop on Vehicular Ad Hoc Networks (VANET '06), ACM, New York, NY, USA, 2006.

[2] L. Armstrong, W. Fisher, Status of Project IEEE 802.11 Task Group p: Wireless Access in Vehicular Environments (WAVE), http://grouper.ieee.org/groups/802/11/Reports/tgp_update.htm, meeting Update (Nov 2007).

[3] A. Bachir, A. Benslimane, A Multicast Protocol in Ad Hoc Networks: Inter-Vehicle Geocast, in: Proceedings of the 57th IEEE Semiannual Vehicular Technology Conference, 2003., vol. 4, IEEE, 2003.

[4] R. Bai, M. Singhal, DOA: DSR Over AODV Routing for Mobile Ad Hoc Networks, IEEE Transactions on Mobile Computing 5 (10) (2006) 1403–16.

[5] R. Bai, M. Singhal, BRD: Bilateral Route Discovery in Mobile Ad Hoc Networks, in: Proceedings of the 6th International IFIP-TC6 Networking Conference (Lecture Notes in Computer Science Vol.4479), 2007.

[6] R. Bai, M. Singhal, Y. Wang, On Supporting High-Throughput Routing Metrics in On-Demand Routing Protocols for Multi-Hop Wireless Networks , Journal of Parallel and Distributed Computing 67 (10) (2007) 1108–18.

[7] S. Basagni, I. Chlamtac, V. Syrotiuk, B. Woodward, A Distance Routing Effect Algorithm for Mobility (DREAM), in: Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking, ACM, 1998.

[8] R. Baumann, Vehicular Ad hoc Networks (VANET): Engineering and Simulation of Mobile ad hoc Routing Protocols for VANET on Highways and in Cities, Master's thesis, Swiss Federal Institute of Technology Zurich (2004).

[9] A. Benslimane, Optimized Dissemination of Alarm Messages in Vehicular Ad-Hoc Networks (VANET), in: Proceedings of High Speed Networks and Multimedia Communications, vol. 3079/2004 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2004, pp. 655–666.

[10] J. Bernsen, D. Manivannan, Unicast Routing Protocols for Vehicular Ad Hoc Networks: A Critical Comparison and Classification, Pervasive and Mobile Computing 5 (1) (2009) 1–18.
URL http://linkinghub.elsevier.com/retrieve/pii/S1574119208000758

[11] J. Boleng, T. Camp, Adaptive Location Aided Mobile Ad Hoc Network Routing, in: Proceedings of the 2004 IEEE International Performance, Computing, and Communications Conference, 2004.

[12] P. Bose, P. Morin, I. Stojmenovi, J. Urrutia, Routing with Guaranteed Delivery in Ad Hoc Wireless Networks, Wireless Networks 7 (6) (2001) 609–616.

[13] A. Boukerche, El-Khatib, L. Xu, An Efficient Secure Distributed Anonymous Routing Protocol for Mobile and Wireless Ad Hoc Networks, Computer Communications 28 (10) (2005) 1193–203.

[14] A. Boukerche, H. Owens, Energy Aware Routing Protocol for Mobile and Wireless Ad Hoc Networks, in: Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks, 2003.

[15] R. Braden, Requirements for Internet Hosts - Application and Support, RFC 1123 (Standard), updated by RFCs 1349, 2181, 5321, 5966 (Oct. 1989).
URL http://www.ietf.org/rfc/rfc1123.txt

[16] R. Braden, Requirements for Internet Hosts - Communication Layers, RFC 1122 (Standard), updated by RFCs 1349, 4379, 5884, 6093 (Oct. 1989).
URL http://www.ietf.org/rfc/rfc1122.txt

[17] I. Broustis, M. Faloutsos, Routing in Vehicular Networks: Feasibility, Security, and Modeling Issues, Tech. Rep. UCR-CS-2006-05219, Department of Computer Science and Engineering, University of California, Riverside (Oct 2006).

[18] J. Burgess, B. Gallagher, D. Jensen, B. N. Levine, MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks, in: Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006), IEEE, 2006.

[19] B. Burns, O. Brock, B. N. Levine, MV Routing and Capacity Building in Disruption Tolerant Networks, in: Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2005), vol. 1, IEEE, 2005.

[20] T. Camp, Y. Liu, An Adaptive Mesh-based Protocol for Geocast Routing, Journal of Parallel and Distributed Computing 63 (2) (2003) 196–213.

[21] H. C. Chang, H. Du, J. Anda, C.-N. Chuah, D. Ghosal, H. M. Zhang, Enabling Energy Demand Response with Vehicular Mesh Networks, in: Mobile and Wireless Communication Networks, vol. 162/2005 of IFIP International Federation for Information Processing, Springer Boston, 2005, pp. 371–382.

[22] B. Chen, K. Jamieson, H. Balakrishnan, R. Morris, Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks, Wireless Networks 8 (2002) 481–494.

[23] D. R. Choffnes, F. E. Bustamante, An Integrated Mobility and Traffic Model for Vehicular Wireless Networks, in: Proceedings of the 2nd ACM International Workshop on Vehicular Ad Hoc Networks (VANET '05), ACM, New York, NY, USA, 2005.

[24] T. Clausen, C. Dearlove, J. Dean, C. Adjih, Generalized Mobile Ad Hoc Network (MANET) Packet/Message Format, RFC 5444 (Proposed Standard) (Feb. 2009). URL http://www.ietf.org/rfc/rfc5444.txt

[25] S. Das, H. Pucha, Y. Hu, Performance Comparison of Scalable Location Services for Geographic Ad Hoc Routing, in: INFOCOM 2005. Proceedings of IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies., vol. 2, IEEE, 2005.

[26] D. De, S. Das, Dynamic Multipath Routing (DMPR): An Approach to Improve Resource Utilization in Networks for Real-time Traffic, in: Proceedings of the Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001.

[27] E. W. Dijkstra, A Note on Two Problems in Connexion with Graphs, Numerische Mathematik 1 (1959) 269–271, 10.1007/BF01386390.
URL http://dx.doi.org/10.1007/BF01386390

[28] Y. Ding, C. Wang, L. Xiao, A Static-Node Assisted Adaptive Routing Protocol in Vehicular Networks, in: Proceedings of The Fourth ACM International Workshop On Vehicular Ad Hoc Networks (VANET '07), ACM, New York, NY, USA, 2007.

[29] S. Eichler, F. Dötzer, C. Schwingenschlögl, F. J. F. Caro, J. Eberspächer, Secure Routing in a Vehicular Ad Hoc Network, in: Proceedings of the IEEE 60th Vehicular Technology Conference (VTC2004-Fall), IEEE, 2004.

[30] K. Fall, K. Varadhan, The ns Manual (formerly ns Notes and Documentation), http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf (2010).

[31] A. Festag, H. Füßler, H. Hartenstein, A. Sarma, R. Schmitz, FleetNet: Bringing Car-to-Car Communication into the Real World, in: Proceedings of the 11th World Congress on ITS, 2004.

[32] E. Fonseca, A. Festag, A Survey of Existing Approaches for Secure Ad Hoc Routing and Their Applicability to VANETS, Tech. Rep. NLE-PR-2006-19, NEC Network Laboratories (Mar 2006).

[33] H. Füßler, M. Mauve, H. Hartenstein, D. Vollmer, M. Käsemann, A Comparison of Routing Strategies in Vehicular Ad-Hoc Networks, Tech. Rep. 3/2002, Universität Mannheim (Mar 2002).

[34] F. Giudici, E. Pagani, Spatial and Traffic-Aware Routing (STAR) for Vehicular Systems, in: Proceedings of High Performance Computing and Communications, vol. 3726/2005 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2005, pp. 77–86.

[35] P. Golle, D. Greene, J. Staddon, Detecting and Correcting Malicious Data in VANETs, in: Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks (VANET '04) , ACM, New York, NY, USA, 2004.

[36] Z. Haas, A New Routing Protocol for the Reconfigurable Wireless Networks, in: Proceedings of the IEEE International Conference on Universal Personal Communications, 1997.

[37] Z. Haas, M. Pearlman, The Performance of Query Control Schemes for the Zone Routing Protocol, in: Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, Seattle, Washington, United States, 1999.

[38] IEEE, IEEE Std. 802.11-2007, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (Jun. 2007).

[39] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, F. Silva, Directed Diffusion for Wireless Sensor Networking, IEEE/ACM Transactions on Networking 11 (2003) 2–16.

[40] ISO/IEC, Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model, Tech. Rep. 7498-1, International Organization for Standardization (1994).

[41] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, L. Viennot, Optimized Link State Routing Protocol for ad hoc Networks, in: Proceedings of the IEEE 2001 International Multi Topic Conference(IEEE INMIC 2001), IEEE, 2001.

[42] Q. Jiang, R. A. Finkel, D. Manivannan, M. Singhal, RPSF: A Routing Protocol with Selective Forwarding for Mobile Ad-Hoc Networks, Wireless Personal Communications 43 (2) (2007) 411–436.

[43] D. B. Johnson, D. A. Maltz, Dynamic Source Routing in Ad Hoc Wireless Networks, in: Imielinski, Korth (eds.), Mobile Computing, vol. 353, Kluwer Academic Publishers, 1996.

[44] H. P. Joshi, Distributed Robust Geocast: A Multicast Protocol for Inter-Vehicle Communication, Master's thesis, North Carolina State University (2006).

[45] B. Karp, H. T. Kung, GPSR: Greedy Perimeter Stateless Routing for Wireless Networks, in: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom '00), ACM, New York, NY, USA, 2000.

[46] M. Käsemann, H. Füßler, H. Hartenstein, M. Mauve, A Reactive Location Service for Mobile Ad Hoc Networks, Department of Computer Science, University of Mannheim, Tech. Rep. TR-02-014.

[47] W. Kieß, H. Füßler, J. Widmer, M. Mauve, Hierarchical Location Service for Mobile Ad-Hoc Networks, ACM SIGMOBILE Mobile Computing and Communications Review 8 (4) (2004) 47–58.

[48] M. Kihl, M. Sichitiu, T. Ekeroth, M. Rozenberg, Reliable Geographical Multicast Routing in Vehicular Ad-Hoc Networks, in: Proceedings of International Conference on Wired/Wireless Internet Communications, vol. 4517/2007 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2007, pp. 315–325.

[49] G. Korkmaz, E. Ekici, F. Özgüner, Ümit Özgüner, Urban Multi-Hop Broadcast Protocol for Inter-Vehicle Communication Systems, in: Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks (VANET '04), ACM, New York, NY, USA, 2004.

[50] L. Lamport, Ti Clocks, and the Ordering of Events in a Distributed System, Commun. ACM 21 (1978) 558–565.
URL http://doi.acm.org/10.1145/359545.359563

[51] J. LeBrun, C.-N. Chuah, D. Ghosal, M. Zhang, Knowledge-Based Opportunistic Forwarding in Vehicular Wireless Ad Hoc Networks, in: Proceedings of the 61st IEEE Vehicular Technology Conference, 2005 (VTC 2005-Spring), vol. 4, IEEE, 2005.

[52] I. Leontiadis, C. Mascolo, GeOpps: Geographical Opportunistic Routing for Vehicular Networks, in: Proceedings of the IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks, 2007, IEEE, 2007.

[53] H. Li, M. Singhal, A Scalable Routing Protocol for Ad Hoc Networks, in: Proceedings of IEEE 61st Vehicular Technology Conference (VTC2005), 2005.

[54] H. Li, M. Singhal, An Anchor-Based Routing Protocol with Cell ID Management System for Ad Hoc Networks, in: Proceedings of 14th International Conference on Computer Communications and Networks, 2005.

[55] J. Li, J. Jannotti, D. De Couto, D. Karger, R. Morris, A Scalable Location Service for Geographic Ad Hoc Routing, in: Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, ACM, 2000.

[56] C. Lochert, H. Hartenstein, J. Tian, H. Füßler, D. Hermann, M. Mauve, A Routing Strategy for Vehicular Ad Hoc Networks in City Environments, in: Proceedings of the IEEE Intelligent Vehicles Symposium, 2003.

[57] C. Lochert, M. Mauve, H. Füßler, H. Hartenstein, Geographic Routing in City Scenarios, ACM SIGMOBILE Mobile Computing and Communications Review 9 (1) (2005) 69–72.

[58] M. Mauve, A. Widmer, H. Hartenstein, A Survey on Position-Based Routing in Mobile Ad Hoc Networks, Network, IEEE 15 (6) (2002) 30–39.

[59] M. Mauve, J. Widmer, H. Hartenstein, A Survey on Position-Based Routing in Mobile Ad Hoc Networks, IEEE Network 15(6) (2001) 30–39.

[60] Z. Mo, H. Zhu, K. Makki, N. Pissinou, MURU: A Multi-Hop Routing Protocol for Urban Vehicular Ad Hoc Networks, in: Proceedings of the Third Annual International Conference on Mobile and Ubiquitous Systems - Workshops, IEEE, 2006.

[61] V. Namboodiri, L. Gao, Prediction-Based Routing for Vehicular Ad Hoc Networks, in: IEEE Transactions on Vehicular Technology, vol. 56(4),2, IEEE, 2007.

[62] V. Naumov, R. Baumann, T. Gross, An Evaluation of Inter-Vehicle Ad Hoc Networks Based on Realistic Vehicular Traces, in: Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '06), ACM, New York, NY, USA, 2006.

[63] V. Naumov, T. Gross, Connectivity-Aware Routing (CAR) in Vehicular Adhoc Networks, in: Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM 2007), IEEE, 2007.

[64] Z. Niu, W. Yao, Q. Ni, Y. Song, DeReQ: A QoS Routing Algorithm for Multimedia Communications in Vehicular Ad Hoc Networks, in: Proceedings of the 2007 International Conference on Wireless Communications and Mobile Computing (IWCMC '07) , ACM, New York, NY, USA, 2007.

[65] OmniAir Consortium, DSRC for the Public Sector: Transportation Facility Operators, http://www.omniair.org/benefits/forpublicsector.html (2008).

[66] V. D. Park, M. S. Corson, A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks, in: Proceedings of the INFOCOM, 1997.

[67] C. Perkins, P. Bhagwat, Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers, in: Proceedings of the Conference on Communications Architectures, Protocols and Applications, London, United Kingdom, 1994.

[68] C. E. Perkins, E. M. Royer, Ad Hoc On-Demand Distance Vector Routing, in: Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, 1999.

[69] J. Postel, Internet Protocol, RFC 791 (Standard), updated by RFC 1349 (Sep. 1981).
URL http://www.ietf.org/rfc/rfc791.txt

[70] H. Pucha, S. M. Das, Y. C. Hu, The Performance Impact of Traffic Patterns on Routing Protocols in Mobile Ad Hoc Networks, in: Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '04) , ACM, New York, NY, USA, 2004.

[71] M. Raya, J.-P. Hubaux, The Security of Vehicular Ad Hoc Networks, in: Proceedings of the 3rd ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN '05) , ACM, New York, NY, USA, 2005.

[72] A. Roy, S. K. Das, QM2RP: a QoS-based Mobile Multicast Routing Protocol using Multiobjective Genetic Algorithm, Wireless Networks 10 (3) (2004) 271–86.

[73] A. K. Saha, D. B. Johnson, Modeling Mobility for Vehicular Ad-hoc Networks, in: Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks (VANET '04), ACM, New York, NY, USA, 2004.

[74] B.-C. Seet, G. Liu, B.-S. Lee, C.-H. Foh, K.-J. Wong, K.-K. Lee, A-STAR: A Mobile Ad Hoc Routing Strategy for Metropolis Vehicular Communications, in: Proceedings of the third International IFIP-TC6 Networking Conference, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications (NETWORKING 2004), vol. 3042/2004 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2004, pp. 989–999.

[75] M. Singhal, V. C. Giruka, A Self-Healing On-Demand Geographic Path Routing Protocol for Mobile Ad-hoc Networks, Ad Hoc Networks 5 (7) (2007) 1113–28.

[76] P. Sinha, R. Sivakumar, V. Bharghavan, CEDAR: a Core-Extraction Distributed Ad Hoc Routing Algorithm, in: Proceedings of the INFOCOM, 1999.

[77] A. Srinivas, E. Modiano, Minimum Energy Disjoint Path Routing in Wireless Ad Hoc Networks, in: Proceedings of the 9th Annual International Conference on Mobile Computing and Networking, San Diego, CA, USA, 2003.

[78] E. T. Clausen, E. P. Jacquet, Optimized Link State Routing Protocol, in: Request for Comments# 3626, 2003.

[79] J. Tian, L. Han, K. Rothermel, C. Cseh, Spatially Aware Packet Routing for Mobile Ad Hoc Inter-Vehicle Radio Networks, in: Proceeding of the IEEE Intelligent Transportation Systems, 2003. , vol. 2, IEEE, 2003.

[80] M. Torrent-Moreno, P. Santi, H. Hartenstein, Fair Sharing of Bandwidth in VANETs, in: Proceedings of the 2nd ACM International Workshop on Vehicular Ad Hoc Networks (VANET '05) , ACM, New York, NY, USA, 2005.

[81] USDOT, Intelligent Transportation Systems Standards Fact Sheet: ASTM E2213-03 - Standard Specification for Telecommunications and Information Exchange Between Roadside and Vehicle Systems - 5

GHz Band Dedicated Short Range Communications (DSRC) Medium Access Control (MAC) and Physical Layer (PHY) Specifications, http://www.standards.its.dot.gov/fact_sheet.asp?f=66, fact Sheet ASTM E2213-03 (Jan 2006).

[82] P.-J. Wan, K. M. Alzoubi, O. Frieder, Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks, Mobile Network Applications 9 (2) (2004) 141–149.

[83] H. F. Wedde, S. Lehnhoff, B. van Bonn, Highly Dynamic and Scalable VANET Routing for Avoiding Traffic Congestions, in: Proceedings of the Fourth ACM International Workshop on Vehicular Ad Hoc Networks (VANET '07), ACM, New York, NY, USA, 2007.

[84] N. Wisitpongphan, F. Bai, P. Mudalige, V. Sadekar, O. Tonguz, Routing in Sparse Vehicular Ad Hoc Wireless Networks, IEEE Journal on Selected Areas in Communications 25 (8) (2007) 1538–1556.

[85] H. Wu, R. Fujimoto, R. Guensler, M. Hunter, MDDV: A Mobility-Centric Data Dissemination Algorithm for Vehicular Networks, in: Proceedings of the 1st ACM International Workshop on Vehicular Ad Hoc Networks (VANET '04), ACM, New York, NY, USA, 2004.

[86] J. Wu, F. Dai, M. Gao, I. Stojmenovic, On Calculating Power-Aware Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks, Journal of Communications and Networks 5 (2) (2002) 169–178.

[87] J. Zhao, G. Cao, VADD: Vehicle-Assisted Data Delivery in Vehicular Ad Hoc Networks, Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006) (2006) 1–12.

Vita

1. Background
    - Name: James Bernsen
    - Date of Birth: October 16, 1976
    - Place of Birth: Corpus Christi, Texas, USA
2. Academic Degrees
    - Bachelor of Science in Computer Science: McMurry University in Abilene, Texas, USA, 1999.
3. Professional Experience
    - PLM Production Support Lead, Lexmark International, Inc, 2006-present
    - PLM Programmer/Admin, Lexmark International, Inc, 2003-2006
    - CAD Applications Programmer, Lexmark International, Inc, 2000-2003
    - Software Engineer, Honeywell Defense Avionics Systems, 1999-2000
4. Publications
    - James Bernsen and D. Manivannan. "Unicast Routing Protocols for Vehicular Ad Hoc Networks: A Critical Comparison and Classification". Pervasive and Mobile Computing, 5(1):1-18, February 2009, Elsevier.