



RIVER: A reliable inter-vehicular routing protocol for vehicular ad hoc networks

James Bernsen, D. Manivannan *

Department of Computer Science, University of Kentucky, Lexington, KY 40506, USA

ARTICLE INFO

Article history:

Received 19 December 2011

Received in revised form 21 August 2012

Accepted 27 August 2012

Available online 5 September 2012

Keywords:

VANETs

Vehicular ad hoc networks

Routing protocols for VANETs

Vehicle-to-vehicle communication

Reliability

ABSTRACT

Vehicular Ad hoc NETWORKs (VANETs), an emerging technology, would allow vehicles on roads 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 paper, 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 these 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.

© 2012 Elsevier B.V. All rights reserved.

1. Introduction

The vehicular ad hoc network (VANET) 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. Within this ever-changing network, messages must be passed from vehicle to vehicle in order to reach their intended destination. To participate in such a network, a routing protocol must direct these message transfers in an efficient manner to ensure robust data communication. Bernsen et al. [3], discuss various design factors of VANET protocols, sur-

veyed a number of VANET routing protocols, and presented an analysis of them.

As a special class of mobile ad hoc networks, VANETs have their own unique characteristics that distinguish them as a subset of this larger class. 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 routing messages along roads is typically necessary.

2. Motivation

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. Road characteristics such as traffic signals and stop signs affect the flow of traffic in urban areas, breaking any sufficiently dense

* Corresponding author. Tel.: +1 859 257 9234.

E-mail addresses: jrbtex@gmail.com (J. Bernsen), mani@cs.uky.edu (D. Manivannan).

URL: <http://www.cs.uky.edu/~manivann> (D. Manivannan).

streams of similar-velocity vehicles. Traffic density, measured in the number of vehicles per unit distance, has a large influence on road capacity and vehicle velocity. 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, and poor road conditions due to weather. 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.

The seminal VANET protocols such as GSR [13] and SAR [18] did not take traffic factors into account. A-STAR [17] utilized static traffic information from bus schedules. The designers 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. Other methods of traffic monitoring considered to be static approaches may 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 typical traffic patterns may 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. If distance between a node and its nearest neighbor is greater than the transmission ranges of both of them, it causes a network 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 Fig. 1 where vehicular traffic on a street is moving away from each other, thus partitioning the network.

Temporary gaps in network are common on most streets at frequent intervals. The use of static data alone cannot adapt to dynamically changing network gaps. A real-time approach is required, and some protocols have attempted this to varying degrees. STAR [8] 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 [15] 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 [6] measures message delivery delays to estimate traffic densities.

ACAR [19], like our protocol, uses a pre-loaded map. However, for determining the connectivity of a road segment, it uses a probabilistic approach. It divides the road segments into cells and clusters and collects the density of vehicles in these clusters and cells. Based on the density of the clusters and cells, it determines the probability of the connectivity of a road segment. It uses this connectivity to choose routes. When a node selects a next hop for forwarding a packet, it does not use a greedy approach in selecting the neighbor but uses the node with best quality of transmission based on the intuition that the farthest neighbor may have high packet error rate. VADD [20] uses a carry-and-forward approach to deal with disconnectivity on a road segment which can cause very large delay. To handle this delay, they propose different variations of the protocol.

Like the edges of a graph, 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, called *anchor points*. 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 since a single disconnected street segment will cause a strictly shortest-path routing to fail. 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.

These observations lead us to a position-based VANET routing protocol that utilizes real-time traffic information to generate a route that travels along a reliable path (a path which is less likely to contain network gaps), even if such a path is not the shortest path in a geographic sense. The rest of the paper is organized as follows: Section 3 introduces the basic idea behind our protocol. Section 4 presents the traffic monitoring component of the protocol. In Section 5, we present how our algorithm calculates the reliability of the edges in the street graph. In Section 6, we present our routing algorithm in detail. Section 7 contains the performance evaluation results and Section 8 concludes the paper.

3. Basic idea of the protocol

Reliable Inter-VEhicular Routing (RIVER) [2] is a position-based VANET routing protocol with an optimized greedy strategy. This protocol prefers transmitting messages using routes it deems to be reliable through its traffic monitoring components. This traffic monitoring happens in real-time by actively sending probe messages along

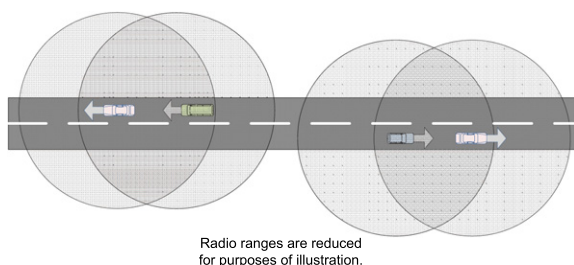


Fig. 1. Formation of a network gap.

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.

Like other greedy, position-based, VANET routing algorithms, RIVER is a geographic protocol that identifies neighboring nodes with beacon messages, has street-awareness, utilizes position-based routes, and forwards packets greedily towards its destination. 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. 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.

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. This real-time, active traffic monitoring allows the protocol to avoid routes along streets containing gaps in node coverage that would prevent message transmission. The protocol tries to avoid roads with network gap by calculating the reliability of routing paths. 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 Sections 4–6.

4. Traffic monitoring

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. Probe and routing packets carry two other forms of traffic information: the **known edge list** and **weighted routes**.

4.1. Active monitoring

In VANETs, beacon messages primarily serve 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 [8] and CAR [15]. However, a node can only detect beacons emanating from nodes within its radio range, and frequently, the reliable range of a radio may be less than the distance between street intersections.

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 one-hop broadcast messages.

Each node maintains a copy of the surrounding street layout in its street graph 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 (within 50 m), and it is forwarded greedily to intended next-hop recipients along the streets that are 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.

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 4.4.

4.2. Passive monitoring

Each node also monitors edge connectivity by passively snooping into routing packets 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 allows a node to determine the reliability of the edges traversed by a packet. 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 Fig. 2, suppose a node receives a routing packet from a distant node. 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.

In addition to gathering traffic data from packets that are directly received by a node, each node also eavesdrops on the radio transmissions between other nearby nodes. For example, probe and routing packets are forwarded to

a specific recipient at each hop. By default, other nodes within radio range of sender 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. In order to perform passive traffic monitoring, each node 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.

4.3. Weighted routes

Every RIVER routing packet contains a list of anchor points for the route, identified by their geolocation. Any two consecutive route anchor points in the list represent 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 which represents time 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 packet prior to forwarding it. The rules in Section 5.2 govern the processing of incoming reliability information and updating of outgoing reliability information.

4.4. Known edge list

Each node monitors beacon, probe, and routing messages, each of which contains a **known-edge list (KEL)**. The known-edge list identifies edges by their endpoint geolocations and communicates reliability information about each edge (e.g. the “z”-marked edges depicted in Fig. 2) along the path. Upon sending a RIVER packet, the sending node selects edges from its own 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.

5. 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. 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 deter-



Fig. 2. Data gained from passive monitoring of a routing packet.

mine if a particular street edge became reliable recently and share this information with other nodes.

5.1. Determining reliable paths

Each node in the RIVER model assigns a weight to every known 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 [5] to calculate what it considers the most reliable routing path. 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 (in terms of the number of edges in a path) is still taken into account. Dijkstra's least weight path algorithm finds a path with least-weight based on the sum of the weights of edges on the path. 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.

5.2. 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, shown in Table 1; other important data points calculated with respect to each edge in the street graph are shown in Table 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 does not simply write all of its known-edge information into every packet it sends. Edges whose reliabilities are unknown (and set to a default value) are not shared. From the remaining edges, a node 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 shared: information about an edge is shared only if this condition is true. Beyond that, 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.

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 a node has no reliability information for an 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 a node already has reliability information for the edge, then it compares the declared timestamp information with its own last updated timestamp and accepts the declared rating if the

Table 1

Edge data fields.

Field	Description
Packet received	Timestamp of the last time when this node received a packet that traversed this edge
Last marked	Timestamp of the last time when this node marked this edge as disconnected
Last declared	Timestamp when this node last accepted a declared reliability value for this edge
Last shared	Timestamp of the last time when this node shared this edge's reliability
Last probe sent	Timestamp of the last time when this node sent a departing probe along this edge
First probe sent	Timestamp of 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 2

Calculated edge data.

Data	Description
Reliability	Based on the information known about the edge, this may be a calculated value (Section 5.3) or equal to the static reliability value above
Shareability	Ranking that dictates how worthy an edge reliability value is to be shared
Last updated	Equal to the most recent value of the edge's packet received, last marked, and last declared timestamps

declared timestamp is more recent. After the declared value is accepted, the node sets the edge's last declared timestamp to be 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.

5.3. 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. 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 the reliability information maintained by a node 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. Edges with low values are preferred over edges with high values when generating a route.

When 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** (10) is used in the calculation. When a node does not receive a response to a probe message sent along an edge, 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** (2) is used in cases where no packet has ever been received along the edge.

In addition to the dynamically calculated values, there are some static values used in RIVER reliability ratings. If no packet has been received along an edge (and the node has not sent a probe along this edge to test it) for a while, a time out period known as the **reliability default** (10 s) eventually expires. 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 reachable 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.

6. Routing

At its most basic level, 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. 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 [1,4,10–12,14]. 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 lo-

Table 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)

cated 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 a routing packet consults the anchor path in the packet header 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 (e.g. IP header destination address [16]) 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 [9]. If the link layer cannot recognize that a link was established with the next hop (e.g. no link-layer acknowledgment from the next hop), then the forwarding node repeats the next-hop identification process and attempts to send the packet again.

As the packet is received at each hop, 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 5.2). Then, the node examines the current anchor point in the packet. When an anchor point has been reached or passed (Section 6.4), 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 more anchor points remain in the routing packet header, the next hop is chosen based on the current anchor point, otherwise the next hop is chosen greedily toward the geolocation of the destination 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 than itself, then the node tries to find a neighbor closer to the subsequent anchor point instead (explained further in Section 6.4).

6.1. 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 threshold is defined as 50% of the reliability default value.

Once the new anchor path is established in the routing header, the next edge in the proposed route is examined. The routing process drops the packet 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.

6.2. 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 as 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.)

6.3. 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 total 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. 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 has 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 later another node 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 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 [16]) 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.

6.4. Greedy optimizations

In the strictest sense, forwarding packets along an anchor route involves greedily forwarding toward each an-

chor 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 Fig. 3 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".

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 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 in the route, 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.

A similar scenario happens when node N_a is closer to the anchor point than node N_b , as in Fig. 4. Here, node N_a is the closest node to the anchor point but is still outside the "reached" range. In a typical greedy algorithm, node N_a would drop the packet. However, 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 will look for a neighbor nearest to the subsequent anchor point such that the neighbor is located on the street edge between the current

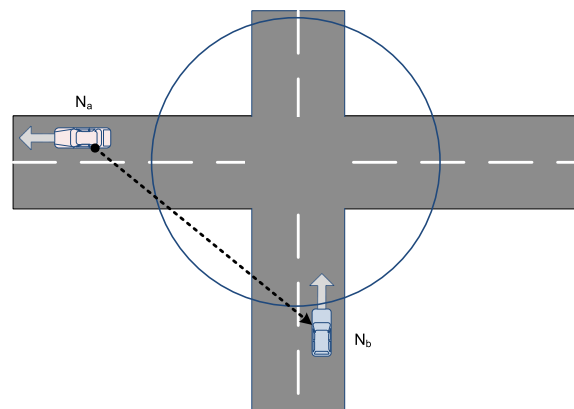


Fig. 3. Past anchor point, outside zone.

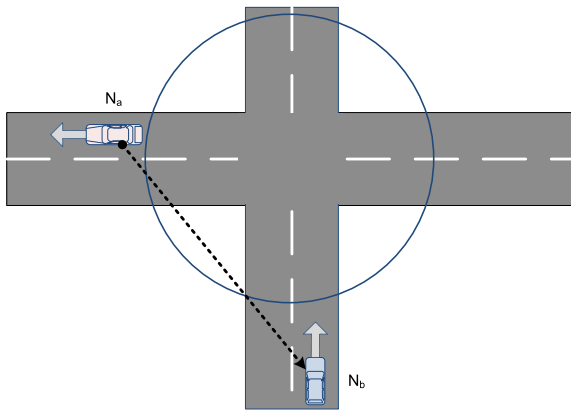


Fig. 4. Outside zone, no closer neighbor.

anchor point and the subsequent anchor point. Instead of dropping the packet, node N_a finds node N_b and forwards to that node. Node N_b detects that the packet has passed the anchor point and increments the AP pointer appropriately.

7. Performance evaluation

To evaluate RIVER, we simulated the protocol with the *ns-2* simulator [7] at version 2.33 using the CMU wireless extension, the default 802.11 bandwidth (2 Mbps) and default transmission ranges. Settings included “WirelessPhy” interface, the two-ray ground propagation model, omnidirectional antenna, and wireless channel configurations. (Our simulation source code is available upon request.)

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 km². This simulation area was populated with varying traffic densities of 100–300 vehicular network nodes distributed randomly. Vehicles traveled in both directions along each street, and vehicles turned left or right or continued ahead (with equal probability) 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. Vehicles travel at a constant speed and do not interact with one another. Each simulation iteration used the same movement pattern.

The connection pattern consisted of five sender/receiver pairs using constant bit rate (CBR) data bursts of 4 Kbps. That is, each sender transmitted a 512 byte packet every 8 s, 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 prevent the possibility of two or more nodes transmitting simultaneously. Each simulation ran for 20 s prior to any routing packet transmissions, and the simulation ran for 15 s following the final send, for a total of 200 s of simulation time (consistent with the simulation times for STAR [8]). For each simulation iteration, the sender/receiver node pairs were randomly selected.

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 [8], the GPSR routing protocol [9], and a shortest-path VANET routing algorithm. For all results, each simulation configuration was repeated for 20 iterations with a different seed at each iteration, and the statistical mean of the results from these iterations was calculated. For RIVER throughput measurements, standard deviation was about 7.3%. As expected, 95% of throughput values were within two standard deviations of the mean.

7.1. RIVER feature analysis

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

7.1.1. Route recalculation and recovery

We evaluated several RIVER protocol options for preventing and/or recovering from routing failures due to network gap as discussed in Section 6. The *recovery* option is a reactive mechanism that engages when a network gap is encountered, while the *recalculation* option is a proactive mechanism that evaluates a data packet's route at each successive node, making route changes based on local information. 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 the route.

Fig. 5 shows 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.

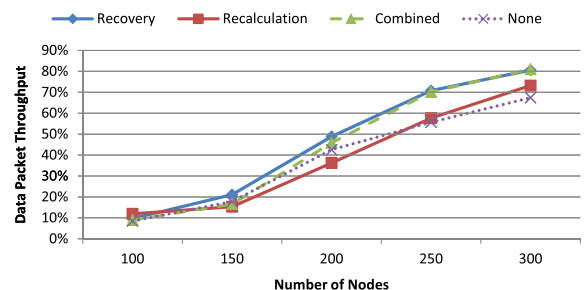


Fig. 5. Data packet throughput with recovery and recalculation strategies.

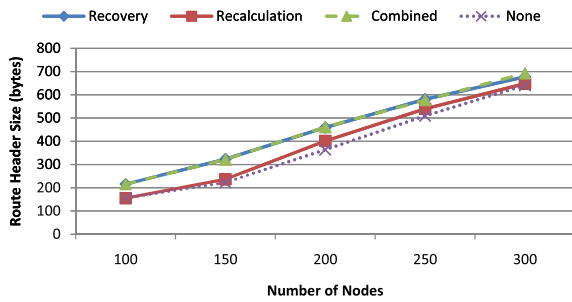


Fig. 6. Route header size with recovery and recalculation strategies.

Fig. 6 shows 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 Fig. 7, we find that the recalculation strategy requires the fewest number of hops per route (excluding the *None* strategy).

In Fig. 8, we observe that the recalculation strategy delivers route packets more quickly than either the recovery or combined strategies.

7.1.2. Reliability distribution

A novel 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

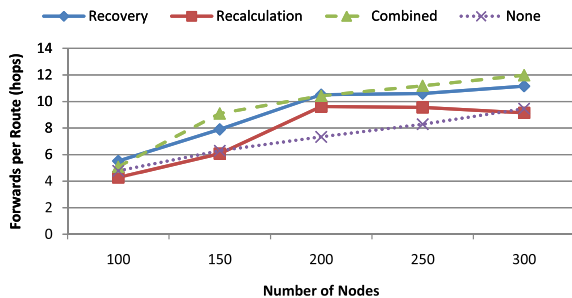


Fig. 7. Forwards per route with recovery and recalculation strategies.

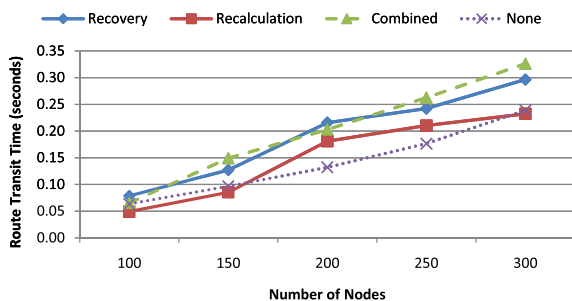


Fig. 8. Route header size with recovery and recalculation strategies.

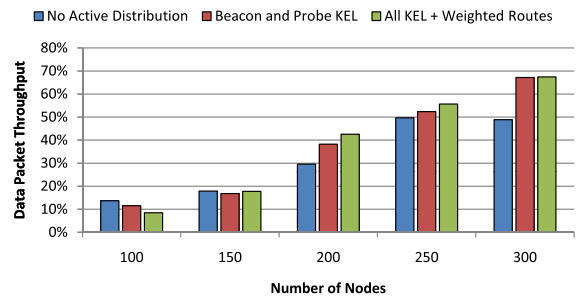


Fig. 9. Effect of active reliability distribution on data packet throughput.

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 Fig. 9, we observe that in average to dense traffic scenarios, active distribution of reliability information via the known-edge list and weighted route mechanisms have a positive effect on data packet throughput. In sparse traffic scenarios, there is a small negative effect.

In Fig. 10, 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 easily observed from this graph, the beacon and probe known-edge list mechanisms do marginally increase beacon and 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. In our simulations, we only encountered this problem when we artificially increased packet size to determine these effects.

7.1.3. 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.

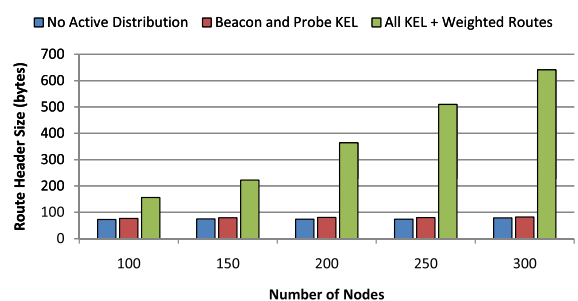


Fig. 10. Effect of active reliability distribution on route header size.

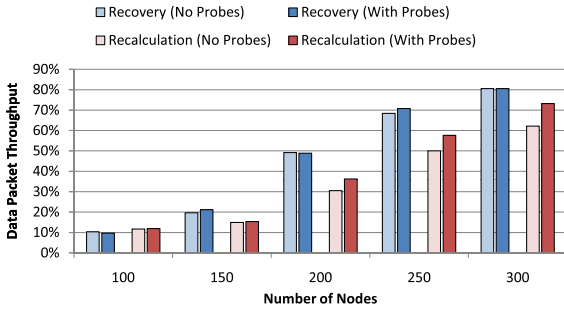


Fig. 11. Effect of probe messages on data packet throughput.

In Fig. 11, 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.

7.1.4. Optimized greedy strategy

As described in Section 6, if the algorithm forwarded packets toward each anchor point in a strictly greedy manner, then some packets would be dropped 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 6.4) that detects these scenarios and handles them appropriately. We compared RIVER using its strict greedy forwarding strategy and the optimized strategy.

In Fig. 12, 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.

7.2. 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 VA-

NETs [8], the GPSR geographic routing protocol [9], and a generic routing algorithm called Short-Path. Both GPSR and STAR 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 without worrying about the reliability of the path. 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 reliability distribution components.

Performance of RIVER was measured with its recovery strategy, optimized greedy forwarding, and with knowledge-lists on beacons, probes, and weighted routes.

From Fig. 13, we find that RIVER delivers packet throughput up to 75% better than Short-Path (45% better on average), up to 222% better than GPSR (102% better on average), and up to 39% better than STAR (8% better on average).

From Fig. 14, we see that RIVER incurs about twice as much delivery delay as Short-Path on average, up to seven times the delay of GPSR, and up to four times the delay of STAR. The reason for the increased route transit times becomes clearer as we examine the number of forwards required to for a data packet to reach its destination in RIVER (Fig. 15). On average, a RIVER data packet travels through about 40% more hops than Short-Path, 84% more hops than GPSR, and 45% more hops than STAR. This is

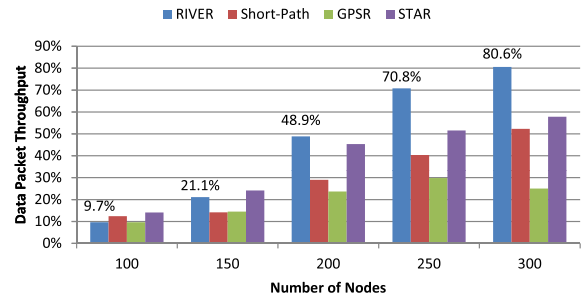


Fig. 13. Peer protocol data packet throughput comparison.

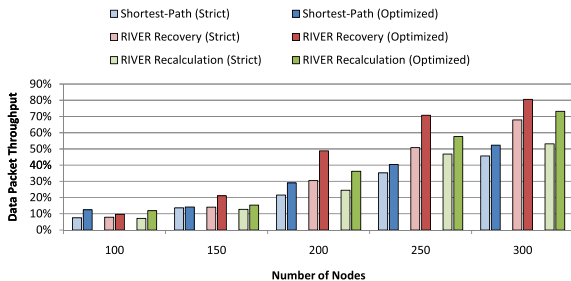


Fig. 12. Effect of optimized greedy strategy on data packet throughput.

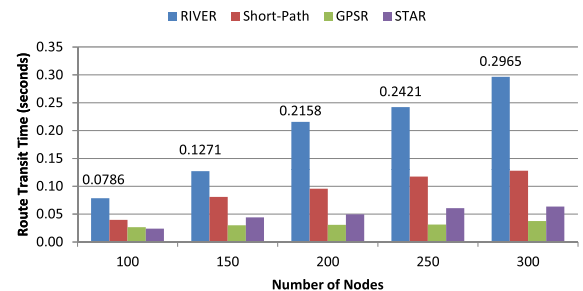


Fig. 14. Peer protocol route transit time comparison.

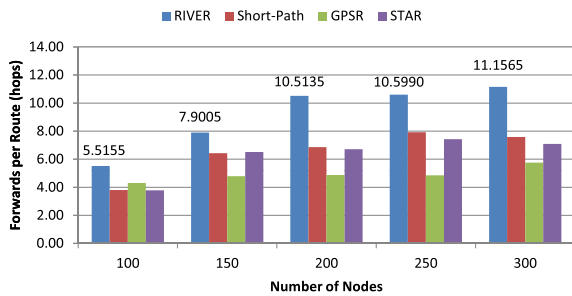


Fig. 15. Peer protocol forwards per route comparison.

due to the protocol's ability to deliver packets to farther destinations than its peer protocols.

8. Conclusions and future work

In this paper, we have proposed "Reliable Inter-Vehicular Routing" (RIVER), a routing protocol for VANETs based on estimated network reliability. RIVER takes advantage of real-time traffic monitoring using active and passive methods. The protocol is able to effectively distribute reliability data throughout the VANET using known edge lists and weighted routes.

In our simulation environment, we found that RIVER provides the highest throughput in most traffic densities when using its recovery strategy, but the recalculation strategy yields higher throughput in low traffic density with less overhead. We also found that RIVER's reliability distribution components perform best in average to high density scenarios. These components cause a significant increase in routing header size, which can be effectively negated by restricting reliability distribution to beacon and probe packets. We also learned that RIVER's optimized greedy forwarding strategy can significantly increase packet throughput with no known negative effects, and this strategy can be applied to 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. Performance under low-density traffic was not a focus point during the protocol's design, so this is an area where its performance could be enhanced. 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.

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 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.

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.

Acknowledgments

We thank the editor and the anonymous reviewers for their valuable comments which helped us greatly in improving the content and presentation of the paper. Thanks also to Brad Karp and Francesco Giudici for making source code available for their respective routing protocols, GPSR and STAR.

References

- [1] S. Basagni, I. Chlamtac, V. Syrotiu, 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.
- [2] J. Bernsen, A Reliability-Based Routing Protocol for Vehicular Ad-Hoc Networks, Master's thesis, University of Kentucky, 2011.
- [3] 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. <<http://linkinghub.elsevier.com/retrieve/pii/S1574119208000758>>.
- [4] 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.
- [5] E.W. Dijkstra, A note on two problems in connexion with graphs, *Numerische Mathematik* 1 (1959) 269–271. doi:<http://dx.doi.org/10.1007/BF0138639>.
- [6] 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.
- [7] K. Fall, K. Varadhan, The ns Manual (formerly ns Notes and Documentation), 2010 <http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf>.
- [8] F. Giudici, E. Pagani, Spatial and traffic-aware routing (STAR) for vehicular systems, in: Proceedings of High Performance Computing and Communications, Lecture Notes in Computer Science, Vol. 3726/2005, Springer, Berlin/Heidelberg, 2005, pp. 77–86.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] M. Mauve, A. Widmer, H. Hartenstein, A survey on position-based routing in mobile ad hoc networks, *Network, IEEE* 15 (6) (2002) 30–39.

- [15] V. Naumov, T. Gross, Connectivity-aware routing (CAR) in vehicular ad-hoc networks, in: Proceedings of the 26th IEEE International Conference on Computer Communications (INFOCOM 2007), IEEE, 2007.
- [16] J. Postel, Internet protocol, RFC 791 (Standard), updated by RFC 1349 (September 1981) <<http://www.ietf.org/rfc/rfc791.txt>>.
- [17] 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.
- [18] J. Tian, L. Han, K. Rothermel, C. Cseh, Spatially aware packet routing for mobile ad hoc inter-vehicle radio networks, in: Proceedings of the IEEE Intelligent Transportation Systems, 2003, vol. 2, IEEE, 2003.
- [19] Q. Yang, A. Lim, S. Li, J. Fang, P. Agrawal, ACAR: adaptive connectivity aware routing for vehicular ad hoc networks in city scenarios, *Mobile Networks and Applications* 15 (1) (2010) 36–60.
- [20] J. Zhao, G. Cao, VADD: vehicle-assisted data delivery in vehicular ad hoc networks, in: Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM 2006), 2006, pp. 1–12.



James Bernsen has an M.S degree in computer science at the University of Kentucky. His academic interests include distributed computing and mobile networking. He has worked as a software engineer and system analyst in the field of product lifecycle management (PLM) and currently manages the PLM team at Lexmark International, Inc.



Dr. D. Manivannan is currently an associate professor of Computer Science at University of Kentucky, Lexington, Kentucky, USA. He received an M.Sc degree in mathematics from University of Madras, Madras, India. He received M.S and PhD degrees in computer and information science from The Ohio State University, Columbus, Ohio, in 1993 and 1997 respectively. He published his research work in the following areas: fault-tolerance and synchronization in distributed systems, routing in wormhole networks, routing in ad hoc

networks, channel allocation in cellular networks, wireless personal area networks and sensor networks. Dr. Manivannan has published more than 50 articles in refereed International Journals (most of which were published by IEEE, ACM, Elsevier, and Springer) and International Conferences.

He is on the Editorial board of IEEE Transactions on Parallel and Distributed Systems, IEEE Communications Magazine, Information Sciences journal, Wireless Personal Communications journal, International Journal On Advances in Telecommunications, International Journal On Advances in Networks and Services and International Journal On Advances in Systems and Measurements. He served as a program chair for two International Conferences and served as program committee member for over 30 International Conferences. He also served as reviewer for more than 30 International Journals published by ACM, IEEE, Elsevier, Springer, Oxford University Press and others. He also served on several proposal review panels of US National Science Foundation and as external tenure reviewer for other universities.

He is a recipient of the Faculty CAREER Award from the US National Science Foundation. He is a senior member of the IEEE and a senior member of the ACM.