RobustGeo: a Disruption-Tolerant Geo-routing Protocol

Ruolin Fan, Yu-Ting Yu*, and Mario Gerla
Department of Computer Science,
University of California, Los Angeles, CA
Email: {ruolinfan, gerla}@cs.ucla.edu
* Qualcomm Research, Bridgewater, NJ
Email: yutingy@qti.qualcomm.com

***Abstract*—While geo-routing is a promising routing algorithm in urban vehicular ad-hoc networks (VANETs), there is still much work to be done for it to become truly usable for such environments. One of the biggest obstacles to this goal is the intermittent nature of VANETs due to mobility. Traditional geo-routing algorithms do not perform well in these conditions because they drop packets whenever they cannot find an immediate forwarder for the packet. In this paper, we propose RobustGeo, a routing protocol that combines the simplicity and efficiency of the greedy forwarding technique in geo-routing algorithms, with the robustness of delay tolerant networks in the face of disruptions in network connectivity. When there exists a good connection between the source and destination, RobustGeo can route the packet like the traditional geo-routing algorithms, and when the network faces disruptions, RobustGeo relies upon vehicle mobility and packet replication to explore multiple geo-route paths and quickly recover the packet back to greedy forwarding. We show that for a highly intermittent scenario, RobustGeo has a delivery ratio of over 20% (compared to a pure geo-routing protocol's delivery ratio of 0), and it reduces delay by over 40% as compared to a more pure delay-tolerant routing solution.**

I. Introduction

Location-based routing algorithms are a class of routing algorithms that use locations of forwarding nodes to route packets. Unlike the more traditional link-state (LS) or distance-vector (DV) algorithms that rely on each node having some overarching information on the network topology [4], location-based routing generally only require each node to have information on its immediate neighbors. This is a desirable property to have because it obviates the need for nodes to periodically flood the network with routing-related messages. Because of their scalability in this respect, these algorithms offer a very promising solution to the difficult problem of routing in vehicular ad-hoc networks (VANETs), where the network topology changes rapidly and bandwidth is limited [21].

One of the first and the most well-known of these routing protocols is the Greedy Perimeter Stateless Routing (GPSR) protocol [7], which sets the foundation for almost all subsequent location-based routing algorithms. In GPSR, each node in the network keeps a list of its immediate neighbors and their locations. Packets are thus routed based on *greedy forwarding* under normal circumstances, where the forwarding node transmits packets to the neighbor who is the closest geographically to the packets' destination. When no node closer to the destination than the current one can be found, the packet is said to have encountered a *local maximum* and must be routed around the perimeter via *perimeter forwarding*, which requires careful mapping of the nodes to form planar graphs so that routing loops can be avoided.

While the greedy forwarding method proposed by GPSR offers a great solution to routing in VANETs, the perimeter routing approach as a solution to the local maximum problem is inadequate [8], as the highly mercurial network topology common to VANETs quickly outdates any planar graphs that may have been built from the nodes, much like routing tables in the DV and LS algorithms. GPCR (Greedy Perimeter Coordinator Routing) [13], another location-based routing protocol that focuses more on urban VANETs, takes advantage of the urban layout that naturally forms a planar graph for its *recovery strategy* from a local maximum, and proposes the *restricted* greedy forwarding method, which prioritizes forwarding packets to urban cross-streets for better routing decision making.

To the end of avoiding local maximums and getting out of them in urban VANETs, many proposals are made [10] [11] [15] [6]. However, all of them make the assumption that despite mobility and the urban landscape, the network is always connected. Unfortunately, this is not the case for VANETs in reality, where **the network is often partitioned or disrupted due to mobility, giving no instant end to end routes**. According to [20], network connections in VANETs are highly dependent upon inter-vehicle spacing, which is never constant, and that on freeways network disconnectivity generally heals on the order of seconds. In such situations, conventional geo-routing algorithms will cause large numbers of packets to be dropped. To address this problem, we propose RobustGeo, a location-based routing algorithm that takes the *store and forward* and *replication* approaches of delay-tolerant networks to explore the possibility of multiple paths to overcome such disruptions in VANET connectivity.

Although delay-tolerant networks (DTNs) were originally proposed for networks that experience very frequent and long periods of disconnectivity [5], the DTN methods, with modification, can be used to deal with the shorter intermittencies in geo-routing with great benefits. One such method is the Data MULE (Mobile Ubiquitous LAN Extension) [16] that exploits node mobility to deliver packets. While this used by itself

translates to unacceptable delay lengths in more conventional networks, we can, in the same spirit, rely on node mobility to recover from a disconnected scenario by saving the packet and actively looking for greedy forwarding routes at the same time. Another important aspect of DTNs is packet *replication*, as seen in Epidemic Routing [19] and Spray and Wait [18]. The drawback to these methods are mainly in bandwidth overhead when too many packet replications occur. However, by using similar methods while limiting the number of replications, we can increase the packet delivery ratio and minimize delay.

With the combination of traditional geo-routing schemes and DTN approaches, RobustGeo indeed offers a more *robust* solution for geo-routing in urban VANETs. When the network connectivity is good, RobustGeo simply acts like any other state-of-the-art location-based routing algorithm. When the connectivity is intermittent, RobustGeo is not only able to withstand these disruptions and continue forwarding packets when the network heals, but also make use of the mobility of nearby neighboring nodes to even increase the healing rate (in the perspective of the packet stored by the node).

The rest of this paper is organized as follows. Section II describes the general design of RobustGeo, while Section III analyzes the potential effects and consequences of packet replication. In Section IV, we evaluate RobustGeo by comparing it with three other geo-routing and DTN approaches, and conclude with Section V.

II. Design

RobustGeo combines the quick speed of geo-routing with the robustness of delay-tolerant networking. Under normal circumstances, greedy forwarding is used to quickly route packets to its destination. When, unavoidably, a local maximum situation arises, it can use a good recovery algorithm like the one suggested in [11] to route around the perimeter while at the same time safeguard against complete disconnectivity by employing the delay-tolerant networking routing approach. In this approach, the nodes keep the packets in their disruption tolerant queue (DTQ), where the packets are opportunistically routed as available greedy forwarding neighbors are found. Periodically, nodes broadcast the packets inside their DTQ to their one-hop neighbors to explore other possible paths and increase the chances of finding a geo-routed path toward the destination.

A. Geo-routing

The geo-routing component of RobustGeo is compatible with all types of greedy-forwarding mechanism based on the one proposed in [7]. Generally, each node keeps a list of its immediate neighbors and looks for the neighbor geographically closest to the packet destination. Following these neighbors, the packets are forwarded greedily step by step until it reaches its destination. To combat challenges posed by urban grid layouts and high urban error rates, restrictive forwarding approaches as described in GPCR [13], GpsrJ+ [10], and TO-GO [11] can be employed.

Likewise, RobustGeo can utilize any of the previously existing solutions for perimeter forwarding to attempt local maxima recovery.

B. Disruption-tolerance

The DTN component of RobustGeo exploits the mobile nature of VANETS. In such a highly mobile environment, it is beneficial to not give up on a local maximum node so quickly since topology can evolve quite rapidly. In many cases, a local maximum at one moment in time may very well no longer be a local maximum in the next simply because of mobility. Therefore unlike conventional geo-routing algorithms, when the perimeter forwarding phase is entered, RobustGeo routes a *replica* of the packet around the perimeter rather than the original packet itself. It then pushes the original packet into the node's DTQ, essentially turning the node into an "intelligent" data MULE [16] that looks for greedy forwarding neighbors and at the same time further packet replication by periodically broadcasting to its one-hop neighbors. In this way, RobustGeo recovers from a local maximum situation by entering DTN mode in addition to perimeter forwarding.

1) Perimeter forwarding with packet replication: When a node encounters a local maximum, it sends out a packet replica for perimeter forwarding, stores the original in its DTQ, and continues to actively try and look for nodes that are closer to the destination than the local maximum. If found, the node has successfully recovered greedy forwarding via DTN means.

Meanwhile, as the replicated packet is routed around the perimeter it records each node it visits as breadcrumbs. If a greedy path is recovered, a receipt can be sent back to the originator using the breadcrumb path so that the originator can remove the packet from its DTQ and abort the broadcasting count-down time. Otherwise, the perimeter-routed packet can continue to be routed by the right-hand rule until either a greedy route is found, or ends up back at the originator or reaches its TTL. In both of the latter cases the packet is discarded. Note that packet replication due to perimeter forwarding happens only once at the original local maximum node. This prevents the packet from replicating out of control as it is forwarded around the perimeter.

If the original packet and its replicas both manage to find a greedy forwarding path, two cases should be considered. In the first case, as illustrated in Figure 1, multiple different paths to the destination are found. This arguably improves the scenario, since having multiple paths to the destination causes the intermittent network to be more robust. In the second case (Figure 2), both packets are forwarded greedily onto the same path. While on the surface this may seem problematic, it is not a great cause for concern since in the unlikely event that this happens, the receiving node can simply drop one of the duplicates and continue to forward the packet onwards. However, this does mean that the packets should be uniquely identified so that identical packets can be easily discovered. A way to realize this is to use a large sequence number in combination with the sender's identifier (like an IP address).

The decision to replicate the packet when doing perimeter forwarding is due to the fact that in urban scenarios, the local maximum situation often occurs because of complete network partitioning. When this is the case, no amount of perimeter forwarding can bring the packet to its destination. Fortunately, these situations are generally not permanent because of mobile nature of VANETs. An example of such an occurrence is when there are gaps in vehicular traffic caused by something as

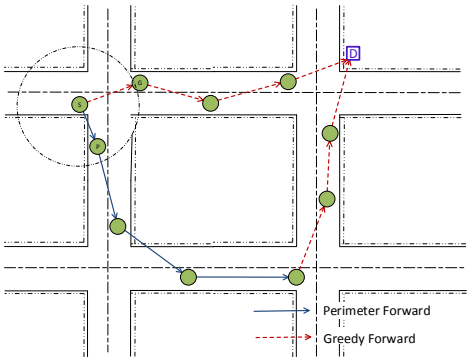


Figure 1: Greedy forwarding and perimeter forwarding finding two different paths. The greedy forwarding recipient node G moves into the source node S's range after S has perimeter forwarded the packet to node P.

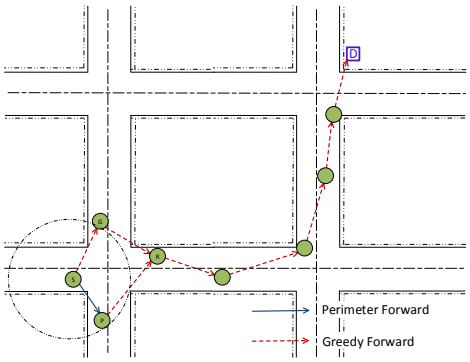


Figure 2: Greedy forwarding and perimeter forwarding finding the same path. Once again, the greedy forwarding recipient node G moves into the source node S's range a moment after perimeter routing is done by S. The two identical packets both pass through node R and the latter one is dropped

simple as a long red light. Traditional perimeter forwarding solutions in these situations will cause the packet to be dropped when it either reaches its TTL or end up back at its originator [7] [1]. With RobustGeo, however, because the local maximum node keeps the perimeter forwarded packets, connectivity that is reestablished a little later can allow greedy forwarding to resume. Although this will cause the network to experience delays, the receiver is still able to receive the message eventually rather than experience complete disconnectivity.

2) Single-hop broadcasting to explore multiple paths: Packets that are not replicas of any previous broadcasts in the DTQ are scheduled to be one-hop broadcasted periodically to explore multiple paths, which potentially increases the delivery rate and decreases latency. Although the node is still continuing to look for greedy routes for the packet, once the packet was broadcasted, it essentially switches into "full DTN mode", relying more on node mobility to recover the next greedy routing path.

When a node receives a broadcasted packet, it first makes sure that it does not already have the packet in its DTQ. Then it adds the packet into its DTQ, where the node can repeatedly attempt to greedy forward it.

The single-hop broadcasting technique is beneficial in situations where node density is sparser, taking advantage of the replication effects in delay-tolerant networking [22]. In RobustGeo, we choose to replicate the packets using single-hop broadcast because it has a high replication-to-transmission ratio in that a packet can reach all the neighboring nodes with just a single transmission.

Since packet replication increases bandwidth usage, it is important to consider the length of the period in between broadcasts. Set too long, the node can lose opportunities to send to neighbors who are good data MULE candidates; set too short, the network would be saturated with replicated packets. With this in mind, we configure the period to be 6 seconds, which we believe is short enough to not miss most of the potential neighbors, yet long enough that it does not overload the network (see Section III). We believe that most of the potential neighbors would not be missed because vehicles rarely travel faster than 20m/s (about 45mph) in an urban area. Taking incoming traffic into account, the relative speed can be up to 40m/s. Therefore 6 seconds is equivalent to 240m, still within most broadcasting ranges. Additionally, we set this as an adjustable parameter depending on needs. For example, if the target scenario is a very sparse network, then it would be beneficial to make this period shorter. In all, due to the relatively long broadcasting period, momentary interruptions in the network will not trigger one-hop broadcasts.

To further limit the number of packet replicas, a packet that was once received via broadcasting can never be broadcasted again. This is easily done by marking a single bit in the packet header and checking that bit whenever a packet is about to be put into broadcast mode. Finally, when the first copy of the packet, be it a replica or the original, reaches its destination, the destination can send out an *active receipt* [2] to clean up all of the packet replicas that are still in the network.

3) Scheduling: When a local maximum situation occurs, the node stores the packet into its DTQ, and packets bound for other destinations with available next-hop nodes continue to be forwarded normally. Every time the node receives a new beacon, be it from any of its existing neighbors or from a new neighbor, the node would check its DTQ to see if any of the packets can now be routed.

The DTQ is similar in function to a normal FIFO queue, in that packets which are enqueued earlier generally get dequeued earlier as well. However, the DTQ is not necessarily dequeued from the head all the time. Whenever a node receives a new beacon from its neighbor, potentially marking a new route, RobustGeo attempts to dequeue from the DTQ by attempting to greedily route to the destination every packet in the queue, beginning with the packet at the head. When a route is found for a packet, that packet is dequeued to be sent out. This speeds up average packet delivery time and keeps the queue length short.

To prevent a large number of packets in the DTQ from suddenly flooding the network, RobustGeo allows only one packet to be sent out for a set period of time. For example, it

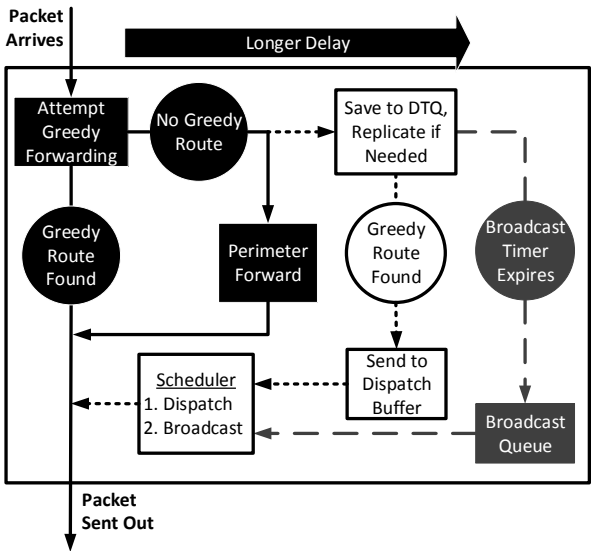


Figure 3: RobustGeo processing a packet. Normal packets use all paths as necessary; Broadcasted packet replicas do not traverse the long-dashed grey path; perimeter forwarded packet replicas only stay on the solid black path

may be decided that the backlogged packets should only take up about 1Mbps of the overall bandwidth, so a maximum of 1 packet can be sent out every millisecond. However, since the beaconing period is much longer than that, a dispatch buffer is introduced to manage the send rate. Therefore, the DTQ dequeues all eligible packets into the dispatch buffer up to a maximum of $send\ rate * beaconing\ period$ packets each beaconing period. Then at every $\frac{1}{send\ rate}$ period, RobustGeo sends out a packet from the dispatch buffer.

Packets marked for one-hop broadcasting are not sent out immediately either. Rather, RobustGeo adds the packet into a separate broadcasting queue and continues to look through its DTQ for other packets to be greedily forwarded. Additionally, the packet marked for broadcasting still remains in the DTQ for more future attempts at greedy forwarding recovery. If the recovery attempt is successful, the packet is removed from both the DTQ and the broadcasting queue. On the other hand, if the packet was broadcasted, it is removed from the broadcast queue only but kept in the DTQ, with the broadcasting timer reset to begin a new period of broadcasting.

Having two queues with different priorities, we employ our dispatch buffer to enforce these priorities. The broadcasting packets are never added to the dispatch buffer. Instead, the dequeuing method for the dispatch buffer dequeues a packet from the broadcast queue only if the dispatch buffer is empty. As a result, packets are only broadcasted when there are no more geo-routed packets to be sent out in the beaconing period. Figure 3 gives an overall view of the components of RobustGeo working together inside a node upon a packet arrival.

III. Analysis

Packet replication is a tricky parameter in ad hoc networks. In a lossy and especially delay tolerant environment, replication can provide substantial benefits in that it increases the delivery rate and decreases delay [22]. Unfortunately, replication is a double-edged sword, as it also introduces bandwidth overhead into the network. Therefore as a protocol that stresses replication, it is important to see just how much overhead replication brings into the network.

In RobustGeo, packets are only replicated when a local maximum is reached. When the node is attempting to route the packet via perimeter routing, it introduces a single packet replica. If the recovery time is long, the packet can be replicated many times due to periodic broadcasting. The overall number of packets replicated as the result of a single local maximum is therefore

$$rep_1 = mn\lfloor\frac{t}{\pi}\rfloor + 1, \tag{1}$$

Where m is the number of packets in the network that experience local maximum recovery exactly once, t is the average local maximum recovery time in seconds; π is the broadcasting period of each node, and n is the number of neighbors that receive the broadcasted packet for the first time.

However, not every replicated packet ends up finding a greedy route. Packets that do not find alternate paths are ultimately dropped as they time out, and perimeter-routed replicas that find a greedy route cancels out its original. For this reason, we assign probabilities to the replicated packets to indicate that they actually remain in the network.

Suppose that each broadcasted packet finds an alternative route with probability P_b, and the perimeter-routed packet finds an alternative route without being able to inform its originator with probability P_p, then the number of replicated packets produced from a single local maximum recovery with these probabilities take into account is

$$rep_1 = mn\lfloor\frac{t}{\pi}\rfloor P_b + P_p. \tag{2}$$

Multiplying P_b to the first term takes into account that the only replicated packets that remain are those who are forwarded to neighbors that eventually find a greedy geo-forwarded route. The value 1 from Equation 1 is replaced with P_p to take into account that the packet replicated from perimeter routing ends up remaining in the network at the probability P_p.

If the distance between the sender and receiver is long, a packet may experience multiple cases of local maximum. Each time it happens, more packets are replicated. In RobustGeo, no packet replicas from broadcasting can be broadcasted again, but they may still replicate via perimeter- routing. We now calculate the number of replicas produced from the second time that m packets encounter local maximum recovery:

$$rep_2 = mn\lfloor\frac{t}{\pi}\rfloor P_b + P_p(mn\lfloor\frac{t}{\pi}\rfloor P_b) + P_p + P_p^2. \tag{3}$$

The first term, $mn\lfloor\frac{t}{\pi}\rfloor P_b$, is the number of packet produced from broadcasting the original packet. The second term, $P_p(mn\lfloor\frac{t}{\pi}\rfloor P_b)$, is the probabilistic number of packets produced from broadcasting the perimeter-routed packet. P_p is the replica from perimeter-routing the current original packet,

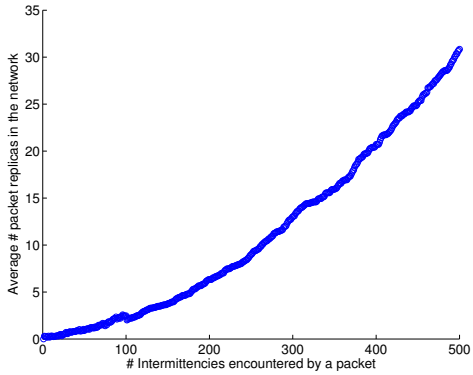


Figure 4: The average number of packet replicas in the network vs. the number of intermittencies encountered by a packet, where the intermittency length t is modeled as a Poisson random variable with $\lambda = 3$. All cases of intermittencies are independent of one another.

and P_p^2 is the replica from perimeter-routing the replicated packet produced from the first time that perimeter-routing was performed.

Since P_p is a probability with value between 0 and 1, as it gets higher powers it becomes more negligible and is disregarded. Further, each time the packet is perimeter-routed, a new *broadcastable* packet is replicated from perimeter routing with probability P_p. The next time that the local maximum is encountered, all of these replicated packets are eligible for broadcasting. Therefore, the total number of replica packets in the network as m original packets are routed through K local maximums is approximately

$$rep \approx \sum_{k=1}^{K} mn\lfloor\frac{t}{\pi}\rfloor P_b(1+(k-1)P_p) + P_p. \qquad (4)$$

Figure 4 shows the number of replicas a single packet accumulates as it is routed in the network. It describes an urban scenario with short but frequent intermittencies. In this scenario, the recovery time is assumed to take 3 seconds on average, and the broadcasting period is 6 seconds. We also assume that the node has on average 10 new neighbors to broadcast to each time that it broadcasts. We also set P_p to be 0.01, a very low number because we assume that for the majority of times, the local maximum occurs due to brief network partitions and so it fails. For a similar reason, we set P_b to be 0.2 because we assume that most of the paths found through broadcasting are not unique. Because the broadcast rate is a floor function, we assume that the length of recovery is roughly a Poisson distribution with $\lambda = 3$[1] (for the 3 seconds of average recovery time) so that the results are not just summations of P_p.

We see that the number of packet replicates increase slowly with the number of intermittencies it encounters. At 100 intermittencies, the average number of replicas is less than 5 per packet. Thus packet replication is manageable.

[1]we can use a discrete pmf in this case since the output values we are concerned with are discrete

IV. Evaluation

We evaluate RobustGeo's performance by simulating three scenarios using the NS3 simulator, with the following parameters:

- PhyMode: OFDM 36Mbps
- Propagation Delay Model: Constant Speed
- Propagation Loss Model: Friis
- WiFi Standard: 802.11a
- Wifi Mac Type: Adhoc Wifi Mac
- Transport Protocol: UDP
- Application: CBR Client-Server Pair (20Kbps)
- Simulation Time: 200s

We begin with an artificially set-up scenario mainly to showcase the major features of RobustGeo, followed by two progressively more realistic scenarios, first with a realistically simulated mobility model running on an actual map of Washington D.C, then with an actual trace of taxi cabs in the San Francisco area. We believe that these three scenarios give a clear view of the superiority of RobustGeo over the other three competing routing algorithms.

To highlight the effectiveness of RobustGeo in intermittent situations, we treat all cases of local maximum in the simulation as network partitioning by not allowing perimeter routing in any of the algorithms we evaluate. For each scenario, we compare RobustGeo against *pure geo-routing*, which simply drops the packet that it cannot immediately route, *geo-routing with a DTQ* that improves delay tolerance but without packet replication, similar in behavior to GeoDTN+Nav [1], and *geo-routing with controlled flooding*, where nodes can only pass along the packet to 5 unique neighbors before dropping the packet.

We base our comparisons on the following metrics: packet delivery rate, average delay, and overall traffic per packet received. While packet delivery rates and average delay are obvious metrics to measure, we believe that the overall traffic per packet received is equally important because packet replication is a major feature of RobustGeo. Finally, using data gathered from the San Francisco cab trace, we analyze the number of intermittencies packets generally encounter in the network, as well as the number of times each packet is broadcasted, to further cement our claim that RobustGeo is scalable in terms of packet replication.

We begin by testing a synthetic scenario as illustrated in Figure 5. This 5-part system consists of groups of three static nodes and two groups of mobile nodes, with a total of 11 nodes. The two clusters of static nodes on the left form a stable connection, while the single group of static nodes on the right relies on the two mobile groups to receive messages. As the mobile nodes move around there are short intervals they form a bridge of network connection between the source and the destination. Most of the time, however, the network on the left side is completely partitioned off from the right. We believe that this is a scenario best fit for RobustGeo since the source is able to reach its stably connected neighbor via

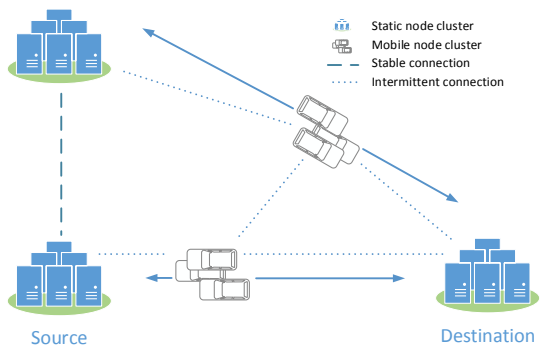


Figure 5: 5 groups of nodes in an intermittent situation; arrows denote node mobility, and dotted lines denote network connection

periodic broadcasting and thus make use of both routes as each become available. Such a path cannot be realized with perimeter routing because when the mobile node on the slant is connected to the static relay, it is not connected with anything else, so all of the perimeter routed packets would be dropped.

We repeat our simulation 10 times with different seeds for each of the routing algorithms in this scenario. In figure 6, we see that as expected, the pure geo-routing method performs poorly because the sender can only transmit to the receiver when the two mobile clusters line up to form the transmission bridge. For the other three schemes, while RobustGeo performs marginally better than the rest in terms of packet delivery, it generated the most packets. We attribute this to the small number of nodes in the network. Controlled flooding only sends packets to another node if the packet is brand new to the node. With only 11 nodes in a highly partitioned system, there is not as much chance for sending packets in such a fashion. RobustGeo, on the other hand, broadcasts periodically regardless of who is around.

We also simulated a more realistic environment by downloading a 2000 m by 2000 m map of the Washington, DC area made available by the US Census Bureau's TIGER database, and simulating mobility on the map using the Intelligent Driver Model with Intersection Management by VanetMobisim [3]. This model is complete with intersections and stop light rules to simulate realistic vehicular traffic. We generate 4 scenarios by varying the number of nodes between 50 and 125. In each scenario, we randomly place a static node on the map as the destination (server) and choose a random mobile node as the source (client).

As Figure 7 shows, in the very sparse scenario of only 50 nodes, pure geo-routing completely breaks down and has a packet delivery ratio of 0. Having the DTQ and using RobustGeo marginally increases the delivery ratio, but it is still under 10%. As the nodes become more dense, the protocols generally trend upwards in their packet delivery ratios. Something quite unexpected is that geo-routing + controlled flooding did considerably worse than pure geo-routing when the number of nodes increased to 125. We attribute this to the fact that each time a node cannot find a greedy forwarding path for a packet, the packet is sent out at least 5 times by that single node alone and multiplied by each recipient 5 more times. This generates a large amount of traffic, taking up available bandwidth and causing interference for nearby nodes. This suspicion is confirmed by the traffic line plot in Figure 7. In all, from the VanetMobisim scenarios we see that RobustGeo is indeed the most robust of all four routing schemes compared, having the highest delivery ratio for every situation. In terms of delay, geo-routing + DTQ did better in the 75-node case, but RobustGeo did better in all other cases. Additionally, we see that as predicted in Section III, the replication overhead of RobustGeo is a manageable one, since it has only a slightly higher traffic to packet received ratio (about 10% or less) than the geo-routing + DTQ scheme that does not have replication.

Finally, we used real traces of an extremely intermittent network to further compare the four geo-routing schemes. In this scenario, we use actual mobility traces of taxi cabs in San Francisco downloaded from Crawdad [14]. We run the simulation in a 5700m by 6600m area with 116 nodes moving for 200 seconds and again place a static node on the map to be the receiver. In this scenario, the nodes are extremely sparse because only cab movements are recorded in the tracefile. This means that the network's period of disconnection is likely longer than its connectivity.

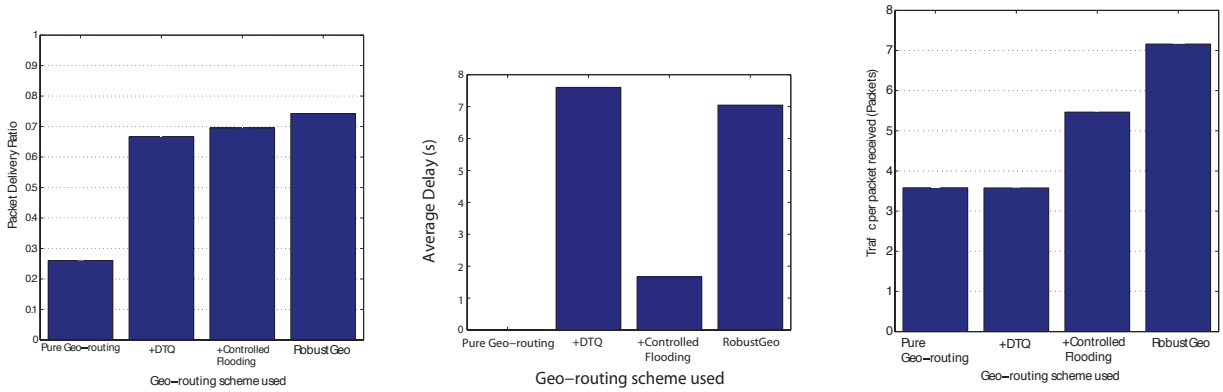
As Figure 8 shows, the pure geo-routing approach completely breaks down, with a receiving ratio of 0. Geo-routing + DTQ and RobustGeo does much better, with RobustGeo almost achieving a 30% packet delivery ratio. Once again, the geo-routing + controlled flooding approach fails, with a less than 1% delivery rate. We again attribute this to its high delay time. This means that had the simulation time been longer, the geo-routing + controlled flooding approach can likely a more acceptable packet delivery rate at the cost of even higher delay times. We choose to show the traffic per packet delivered metric for only pure geo-routing + DTQ and RobustGeo because the other two approaches yields incredibly high numbers (infinity and about 2500). As expected, RobustGeo generated 58% more traffic per packet delivered than geo-routing + DTQ. At the same time, it also has a 36% higher delivery ratio with a very slightly lower average delay. We believe that this is a good tradeoff as packet delivery ratios should be prioritized over bandwidth conservation.

Since the San Francisco cab scenario is very realistic and at the same time gives an extremely intermittent network, we use it to analyze disconnectivity and broadcast frequencies. Table I shows that in the given environment, the majority of packets experience disconnectivity 3 or fewer times, with only 8 packets encountering more than 4 intermittencies. Meanwhile, Figure 9 shows that about 80% of packets get broadcasted 6 times or fewer. This confirms the fact that packet replication will not spiral out of control.

We also postulate that the majority of packets experiencing higher frequencies of broadcast do not end up at the destination. If true, this can be used as a good metric to decide how long the packet should remain in a node's DTQ before being dropped.

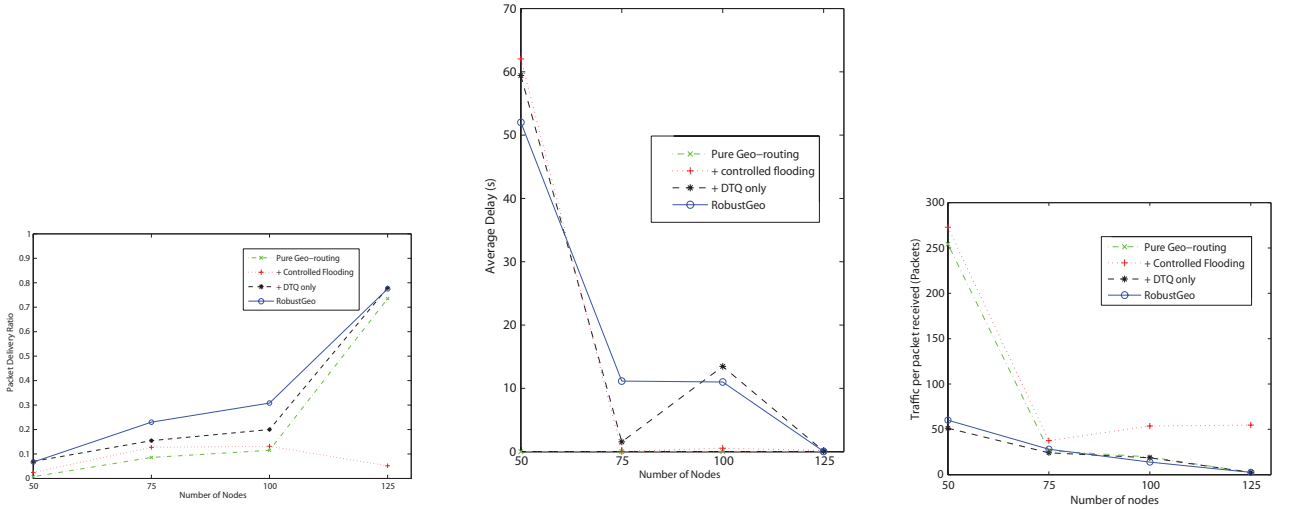
V. Conclusion and Future Work

In this paper we presented another location-based routing algorithm, RobustGeo, that is designed to withstand net-



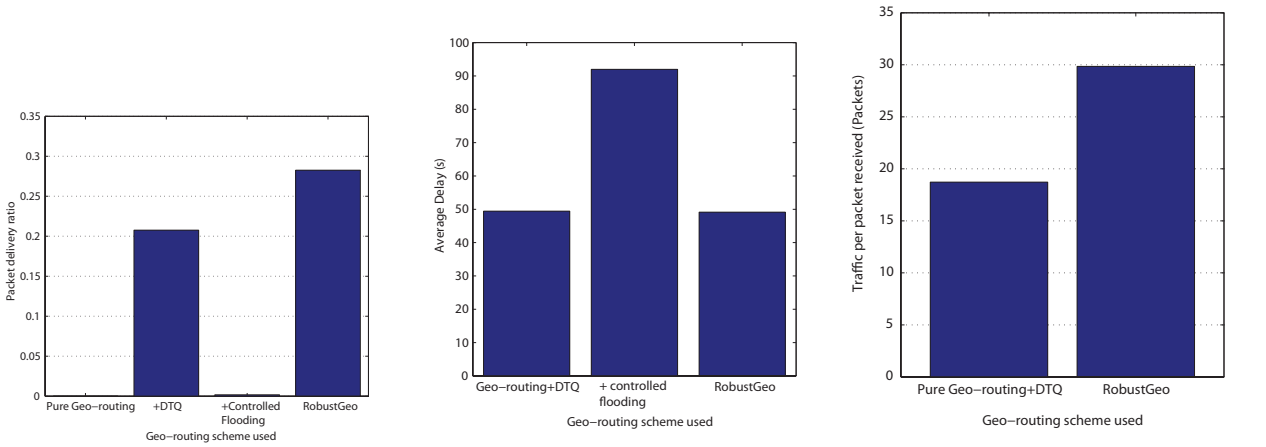
(a) Packet delivery ratios (b) Average delay (seconds) (c) Total traffic generated per packet received

Figure 6: Results for the 5-group artificial scenario



(a) Packet delivery ratios (b) Average delay (seconds) (c) Overall traffic generated per packet received

Figure 7: Results for the realistically simulated mobility in Washington, DC using VanetMobisim and TIGER maps



(a) Packet delivery ratios (b) Average delay (seconds) (c) Overall traffic generated per packet received

Figure 8: Results for the real life San Francisco taxicab simulation scenario

# intermittencies	# packets
1	1434
2	1063
3	981
4	214
5	3
6	5

Table I: The number of packets that encounter each frequency of intermittencies in the SF Taxicab scenario

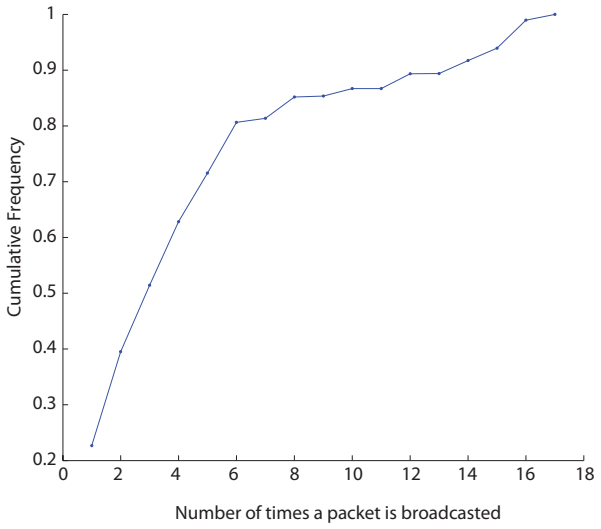


Figure 9: The cdf of packet broadcast rate in the SF Taxicab scenario

work intermittencies common to urban VANETs. RobustGeo achieves this result by taking advantage of both the store-and- forward and packet replication strategies found in delay tolerant networks. With RobustGeo, we were willing to make the tradeoff of introducing extra bandwidth overhead into the network with packet replications in favor of more reliable delivery. We showed in both Sections III and IV that the tradeoff is a viable one. As future work, we would like to analyze the relationship between a packet's frequency of broadcasts and its journey completion likelihood to get a better idea of how long the packet should remain in the DTQ before timing out and being dropped. This can be very helpful in RobustGeo as it helps reduce node *storage overhead* in addition to bandwidth overhead. In all, RobustGeo is a hybrid solution that allows for connections in both reliable and intermittent networks, which are attributes of VANETs.

REFERENCES

[1] P.-C. Cheng, K. C. Lee, M. Gerla, and J. Härri. Geodtn+nav: Geographic dtn routing with navigator prediction for urban vehicular environments. *Mob. Netw. Appl.*, 15(1):61–82, Feb. 2010.

[2] K. A. Harras and K. C. Almeroth. Transport layer issues in delay tolerant mobile networks. In *NETWORKING 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*, pages 463–475. Springer, 2006.

[3] J. Härri, F. Filali, C. Bonnet, and M. Fiore. Vanetmobisim: Generating realistic mobility patterns for vanets. In *Proceedings of the 3rd International Workshop on Vehicular Ad Hoc Networks*, VANET '06, pages 96–97, New York, NY, USA, 2006. ACM.

[4] A. Huhtonen. Comparing aodv and olsr routing protocols. *Telecommunications Software and Multimedia*, pages 1–9, 2004.

[5] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 145–158, New York, NY, USA, 2004. ACM.

[6] M. Jerbi, S.-M. Senouci, R. Meraihi, and Y. Ghamri-Doudane. An improved vehicular ad hoc routing protocol for city environments. In *Communications, 2007. ICC'07. IEEE International Conference on*, pages 3972–3979. IEEE, 2007.

[7] B. Karp and H.-T. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254. ACM, 2000.

[8] K. C. Lee. *Geographic routing in vehicular ad hoc networks*. University of California at Los Angeles, 2010.

[9] K. C. Lee, P.-C. Cheng, J.-T. Weng, L.-C. Tung, and M. Gerla. Vclcr: a practical geographic routing protocol in urban scenarios. *UCLA Computer Science Department, Tech. Rep. TR080009*, 2008.

[10] K. C. Lee, J. Haerri, U. Lee, and M. Gerla. Enhanced perimeter routing for geographic forwarding protocols in urban vehicular scenarios. In *Globecom Workshops, 2007 IEEE*, pages 1–10. IEEE, 2007.

[11] K. C. Lee, U. Lee, and M. Gerla. To-go: Topology-assist geo-opportunistic routing in urban vehicular grids. In *Wireless On-Demand Network Systems and Services, 2009. WONS 2009. Sixth International Conference on*, pages 11–18. IEEE, 2009.

[12] I. Leontiadis and C. Mascolo. Geopps: Geographical opportunistic routing for vehicular networks. In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, pages 1–6. IEEE, 2007.

[13] C. Lochert, M. Mauve, H. Füßler, and H. Hartenstein. Geographic routing in city scenarios. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(1):69–72, 2005.

[14] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. CRAWDAD data set epfl/mobility (v. 2009-02-24). Downloaded from http://crawdad.org/epfl/mobility/, Feb. 2009.

[15] B.-C. Seet, G. Liu, B.-S. Lee, C.-H. Foh, K.-J. Wong, and K.-K. Lee. A-star: A mobile ad hoc routing strategy for metropolis vehicular communications. In *NETWORKING 2004. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications*, pages 989–999. Springer, 2004.

[16] R. C. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: Modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks*, 1(2):215–233, 2003.

[17] V. N. Soares, J. J. Rodrigues, and F. Farahmand. Geospray: A geographic routing protocol for vehicular delay-tolerant networks. *Information Fusion*, 15:102–113, 2014.

[18] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 252–259. ACM, 2005.

[19] A. Vahdat, D. Becker, et al. Epidemic routing for partially connected ad hoc networks. Technical report, Technical Report CS-200006, Duke University, 2000.

[20] N. Wisitpongphan, F. Bai, P. Mudalige, and O. Tonguz. On the routing problem in disconnected vehicular ad-hoc networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 2291–2295, May 2007.

[21] S. Yousefi, M. S. Mousavi, and M. Fathy. Vehicular ad hoc networks (vanets): challenges and perspectives. In *ITS Telecommunications Proceedings, 2006 6th International Conference on*, pages 761–766. IEEE, 2006.

[22] Z. Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges. *Communications Surveys & Tutorials, IEEE*, 8(1):24–37, 2006.