Event Dissemination in High-Mobility Ad-hoc Networks

Masataka Kan Rahul Pande Patrick Vinograd Hector Garcia-Molina
Stanford University
Computer Science Department
353 Serra Mall, Stanford CA 94305-9025
masa@db.stanford.edu, {rpande,patrickv,hector}@cs.stanford.edu

Abstract

We focus on effective event dissemination in vehicular environments. In particular, without relying on any fixed infrastructure, what is an efficient way to propagate an event amongst a collection of (fast) moving cars? In this paper we propose a general-purpose flooding-based event dissemination framework for such an environment, and describe two protocols that implement its broadcast mechanism. In particular, one of the schemes called SR is a new very simple algorithm that gives surprisingly good results. We present an evaluation framework for the analysis, and conduct a thorough evaluation of the algorithms.

1. Introduction

As wireless devices become more ubiquitous, automobiles will soon be able to communicate wirelessly with nearby cars. Currently, cars can already use the cellular infrastructure (e.g., with OnStar passengers can communicate with remote operators). However, a peer-to-peer approach, where cars communicate directly with other cars, may be more cost effective, and can be used in environments where emergency conditions or underdevelopment make relying on a fixed infrastructure impossible. With peer-to-peer communication, equipped cars could inform enabled neighboring vehicles of road conditions or emergency situations, while children could play interactive games with remote passengers. In addition, cars could pick up news (e.g., game scores, weather reports) as they passed certain base stations, and then relay the data to other vehicles.

Our paper focuses on how to effectively disseminate events in such a highly mobile vehicle environment. For example, given a weather report or news of an accident, what is an effective way to communicate it to the largest number of cars in the area using solely peer-to-peer communication? The broadcast primitive we study is also important because it can form the basis of a more general publish-subscribe system for a vehicular network.

The main constraint faced by an event dissemination scheme in such a setting is that the amount of inherent motility precludes the use of state-based routing techniques (such as those used by most wireless applications). The high flux makes it difficult to justify the use of even ad-hoc routing methods that assume limited mobility – there is not enough stability to maintain a route even for the short period of time required by algorithms such as AODV [9] and DSR [6].

In this paper, therefore, we study if a flooding-based protocol can be used to effectively disseminate events (messages) in a dynamic environment. The dynamism introduces several interesting aspects, not seen in traditional static flooding scenarios:

- In a static network where neighbors are fixed, a node might as well immediately retransmit a received event as soon as it gets it. However, in a dynamic network, it may be advantageous to have a car (node) delay retransmission, giving it a chance to travel to a region where the event has not been heard.
- Similarly, in a static network, once a node re-transmits an event, there is no sense in re-transmitting later on. However, in a dynamic environment where the topology changes constantly, the same car can re-transmit the same event multiple times, since each time it may be heard by new neighbors.
- In a dynamic environment, cars may frequently depart (or turn off) and arrive. This churn makes it harder to achieve high coverage. It also makes it more challenging to properly define "coverage," since we do not want to penalize a protocol for not delivering a message to cars that have no way of receiving an event.

To handle the dynamic flooding, in this paper we present two algorithms. The first is a variation of the traditional TTL scheme (time-to-live), where each event is retransmitted a fixed number of times. In our case, we delay each transmission to optimize coverage. The second scheme is a new algorithm we call SR (send-receive). The SR algorithm is extremely simple: each car can re-transmit an event a fixed number of times, independent of how many times the event has been retransmitted. In spite of its simplicity, SR performs surprisingly well, giving better coverage with less overhead than TTL in many scenarios. Furthermore, selecting the right SR threshold (number of times a car retransmits an event) is often simpler than selecting the right TTL.

In summary, the main contributions of this paper are as follows:

- We present two tunable dissemination algorithms (Delay TTL and SR) for a highly dynamic vehicular network.
- We present a detailed simulation model for studying the algorithms. While the model is relatively simple, it captures many of the key factors such as car density, road layout and transmission radius.
- We propose metrics, such as coverage, overhead and latency, that are specifically targeted to a highly dynamic vehicular network.
- We present a thorough experimental evaluation and comparison of the algorithms.

2. Related Work

The large body of work related to mobile and ad hoc networking is obviously relevant to our work. Much of this work still assumes a lesser degree of mobility than we do; routing algorithms such as AODV [9] and DSR [6] still rely on the ability to establish a route, even if that route is not long-lived. There is also work being done related to wireless communications between moving automobiles and fixed base stations[8]. This work indicates that the physical layer allows fast-moving network entities to communicate successfully.

Epidemic and gossip algorithms are used in a variety of settings, such as database replication [4], news dissemination, and multicast routing. Gossip typically refers to one entity picking another entity at random, and the two exchanging any information not known by both, thus simulating the spread of gossip in a social network. Such randomness could be added to the protocols we present in this paper.

Publish-Subscribe systems dynamically route events from the source (publisher) to any party that is interested (subscriber), where interest is determined by either the meta-data or content of an event. In most cases these systems work by establishing a structure of intermediary entities, or brokers, between event sources and destinations. Much of the work related to mobility in publish-subscribe systems [5] refers to the ”access-point” model of mobility where an endpoint is mobile but the majority of brokers remain fixed. In a highly mobile environment like the one we study, a publish-subscribe mechanism would most likely use broadcast primitives like the ones we study in this paper.

Regional Alert systems are the closet in flavor to what we are trying to create. These protocols rely on peer-to-peer communication to inform vehicles in the neighborhood of a source about the event being broadcasted [11, 3, 1]. Unlike our work though, these schemes assume that event information is location-sensitive and so concentrate on ensuring that all peers that come within a certain distance from the source are informed of the event in a timely manner; they do not focus on the general issue of efficient event dissemination to all peers.

3. Event Dissemination

As described in the introduction, our environment comprises a collection of moving automobiles that communicate with each other without relying on any fixed infrastructure. We assume that the region that these peers can travel in is of a finite (but arbitrary) size; new peers, however, will be able to enter and existing peers be able to leave this area along its boundary. Since cars are constantly moving, any cached state based routing information will quickly become stale.

In order to propagate a message the sender has to rely on peer-to-peer transmissions, i.e., the sender will broadcast a message to some/all of its neighbors which in turn, will rebroadcast it to some/all of their neighbors . . . until the intended recipient is eventually reached. We do not allow cars to speed up/ slow down/ change paths in order to hasten/ slow the spread of messages. We assume that each car has a small broadcast footprint relative to size of the region in which it can move – so it is unlikely that a sender will be able to reach its intended recipient with a single broadcast.

As the cars are independent entities, we allow a peer to have access to local information (such as its previously sent messages, what time it got a particular message, whether there are any peers within its communication radius, . . .), but not to global information (such as how many peers have received copies of a message, the location of other peers, . . .). For this paper we assume that each peer does *not* have the means to determine its own location. There is a large body of work in ad-hoc vehicular networks that does posit that cars are able to determine their position [11, 3, 1], generally via GPS. The algorithms we present here could be extended to exploit location information in an analogous fashion.

In our model designated sources periodically generate *events* – atomic pieces of information (e.g. a report that an accident has occurred) – that must be disseminated to as many peers in the system as possible. Each event will be broadcasted by its source only once, i.e., after the original broadcast, the only way an event can spread through the system is if the peers that receive it retransmit it efficiently. Since messages carry events, in this paper we use the terms interchangeably.

Except for Section 5.6, in this paper we focus on the dissemination of a single event, in order to understand clearly how the broadcast primitive works. However, in Section 5.6 we briefly study a multiple event scenario, where the source originates a sequence of related events. For example, the source could be transmitting partial results of a baseball game, so that the newer events supersede the older ones.

The objective of the dissemination algorithm is to have an event reach as many peers as possible (high coverage), as quickly as possible (low latency), with as little overhead as possible. In Section 4.1 we define these metrics more precisely. For now, we simply note that in many cases it is impossible to achieve high coverage, regardless of the algorithm. For example, if car density is very low, the propagation of an event may "die out" because there are no nearby peers. Similarly, it may be impossible to reach peers that only show up for brief periods of time. Even though coverage may be low, we still want an algorithm that can do as well as possible, given the circumstances.

3.1. Event Dissemination Algorithm

The high rate of motility inherent in the environment and the requirement that the maximum number of cars be informed of a broadcasted event (i.e. the coverage be high) forces the event dissemination mechanism to rely on a flooding-based approach. The characteristics of a vehicle network, however, makes flooding-based communication a much richer area of study than it is in a fixed network.

As mentioned earlier, when flooding in a fixed network there is little incentive to delay rebroadcasting an event (other than perhaps to avoid collisions on a shared medium); similarly, there is no sense in forwarding an event multiple times, since the topology is unlikely to change. In contrast consider a vehicle network: When a peer receives an event, it can retransmit it immediately, but it is fairly likely that many of its current neighbors received the same initial broadcast. Instead, the peer can wait for some period of time during which it is likely that it will encounter new neighbors who are have not seen the event. After the initial rebroadcast, the peer continues to move around and potentially encounter new neighbors, so there is a good chance that if it receives a copy of an event it has previously transmitted, forwarding it again will be useful.

We next present our dissemination algorithm, which is based on this intuition. The algorithm uses three functions, which as we discuss below can be implemented in different ways to achieve different effects: (i) a *delay* function, which is used by a peer to pick an interval to wait before transmitting; (ii) a *suppression* function which is used to decide whether to forward the message at all, or to drop it for the time being (if it receives another copy of the message, it might decide to forward it later); and (iii) a broadcast function which transmits the event.

Each time a peer receives an event e (message), it goes through the following process. (Note that this description is conceptual; an actual implementation may do things differently.)

1. Use the suppression function to decide if e (or perhaps a copy of e in the pending queue) is discarded.

2. If e was not discarded, it is added to the queue of pending events and its delay d is calculated using the delay function.

3. Wait for d seconds.

4. Use the broadcast function to transmit e, and remove e from the queue of pending events. (In some cases, we may add e to a cache of already transmitted events, see below).

3.1.1 Suppression, Delay and Broadcast Functions

For the suppression mechanism we consider two options:

- TTL (HOP COUNT): Each event is created with some initial TTL value. When a peer receives an event e, it decrements the TTL by 1, and if the TTL reaches 0, the event is dropped.

  If TTL is non-zero and e is added to the queue, we check if any copy of the event, e' is found in the pending queue. If so, we discard e' (leaving e in). (When two copies of an event are received in close proximity, it means there are multiple cars in the area. In this case, we have found it is best to only consider the copy that arrived the latest.)

- SR (Send/Receive) Count: This is a new suppression scheme that we created specifically for this protocol. Each peer maintains a local cache of previously transmitted events, and how many times it has broadcast each. Once this local count has reached some threshold value, the event is dropped, as are any future instances of same event. The rationale here is that if a peer receives an event several times, it is likely that most of the peers around it have already received the event.

After an event e is added to the pending queue (its count was below the threshold), we check for duplicates (as we did for TTL). That is, we remove any copies e', leaving only the latest copy e in the pending queue.

The SR scheme does require a cache of previously broadcast events (unlike TTL). However, we do not expect the cache to be large: Entries can be purged after an event is expected to be fully disseminated. Furthermore, each entry only need consist of a hash of the event, to distinguish it with high probability from other events.

A variety of delay functions can be used, although in this paper we only experiment with the first one:

- FIXED(n,m): Upon receiving a message, the peer waits for a period selected uniformly from the interval $[n, m]$ time units.
- VARIABLE(n,m): The delay interval grows larger for events with larger SR count (or smaller TTL). The intuition is that peers with a larger SR count (or smaller TTL) may have received several transmissions from nearby and so should wait longer before retransmission.
- INVERSE(n,m): The delay interval grows shorter for events with larger SR count (or smaller TTL). In this case peers with a low SR count (or high TTL) are allowed to move further away from the source before rebroadcasting.

The broadcast function can implement a variety of optimizations. For example, if a collision occurs during transmission, the event can be re-transmitted soon after. Similarly, if we can tell that no cars are within range, we can postpone the transmission briefly until cars appear. For our evaluation, we assume that both of these optimizations are in place.

4. Modeling a Vehicle Network

In order to test and parameterize our event dissemination scheme, we use a simplified simulation model of automobiles moving around in physical space, as well as for the wireless communication and data exchange. To be tractable, the model must be simple, but it must still capture the essential features of the environment. We believe that the model we will present makes the right tradeoffs: it allows us to study the interesting aspects of the algorithms, while avoiding the details that do not affect *relative* performance. (The omitted details may affect performance overall, but we believe do not shed light of the differences between approaches.)

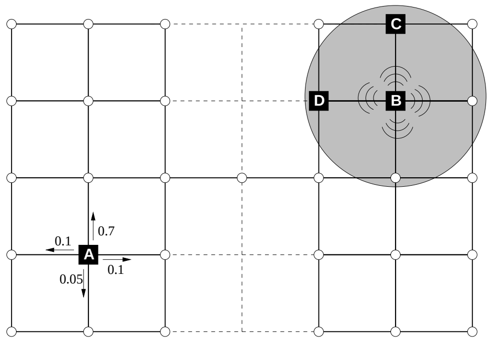

Figure 1. Our model.

We consider the area in which the cars move around, or *map*, to be a regular two dimensional grid (see Figure 1). The size and scale of the grid can be adjusted. Each intersection of the grid is a *location* (representing some stretch of road in real space) where zero or more cars may be present at a given time. Cars may move along the edges between locations according to one of the mobility models described below. We can eliminate some edges (dotted lines in Figure 1), thus creating potentially interesting road layouts that we wish to study.

We use a biased random mobility model to describe the motion of cars on the map. Cars move from location to location at regular intervals. A car is more likely to move in the same direction that it moved during the last interval, but has some fixed probability of turning right or left, reversing, or not moving at all. (For example, Peer A in Figure 1 moves up with probability 0.7, moves left or right with probability 0.1, moves down with probability 0.05, and stays in place with probability 0.05.) In our present configuration all cars (independently) chose whether to move/not to move at the same time intervals.

When a car reaches the edge of a map, it can turn and remain on the map or carry on and exit the simulation. Once a car leaves it cannot reenter the simulation at some later time. As we want the density of cars on the map at any given time to be fairly constant, we model departures by having an entry onto the map scheduled for some random interval after a car leaves the map. This simulates an approximately steady state level of automobile traffic, which seems like a reasonable assumption over the time intervals we are interested in.

When a peer transmits a message, it is heard by all the peers at neighboring locations out to some radius (barring collisions). (For example, in Figure 1, Peer B has a communication radius of 1, so Peers C and D hear its broadcast.) The transmission has a duration directly proportional to the number of bytes in the message. At a given location, transmissions from two or more peers might overlap in time, in which case a collision results and neither transmission is received by any of the peers at that location. For handling collisions, we initially use a basic CSMA MAC protocol; a peer that wants to transmit will do so if it detects that the channel is clear. This obviously makes the system susceptible to collisions due to the hidden terminal problem. A more sophisticated MAC such as CSMA/CA[12] (or one of the many other alternatives [7, 2, 10]) could be used, but we do not believe it will make a significant difference in our results.

As mentioned earlier, in this paper we study the dissemination of a single event. (Multiple events are studied briefly in Section 5.6.) Each new event is broadcast by a source only once – to ensure that it is not lost, the source makes this initial transmission only when there is at least one peer in its vicinity (it should be noted though, that event generators can also be event recipients – so even though a source can generate an event only once, it can forward it more than once – potentially one for each time it is notified about it from its neighbors). Peers will then relay the event to others by sending additional messages containing the event. In our results graphs, each data point represents the average of 10 experiments, where one event is disseminated in each experiment.

4.1. Metrics

The efficacy of each dissemination scheme was gauged on the basis its of the coverage, message overhead and to some extent its latency.

Event Coverage. Since peers can exit the map during a simulation, we weight each peer by the length of time that it spends on the map. Weighting is necessary as our experiments have indicated that on average a peer stays in the map for only $\frac{1}{3}$ of the total simulation time, with a large chunk of the vehicles in the simulation leaving the map after only a few moves. If the peers are not weighted therefore we would expect the algorithm to achieve a high coverage only if it can notify nearly all peers about an event – even though many of them are on the map for only a few time units. This is clearly unrealistic.

Under the weighting scheme, if peers with a high weight hears news of an event, the coverage is increased by a greater amount than if a peer with a low weight is informed of the event. The intuition behind this is that the protocol has no excuse for not notifying peers with high weights (because they were always on the map), but can still be “good” and miss peers with low weights simply because they were barely present.

More formally, let the weight of $Peer_i$ for $Event_j$ be $w_{i,j}$:

$$w_{i,j} = \frac{\|\text{Time } Peer_i \text{ is in simulation}\|_j}{\|\text{Total simulation time}\|_j}$$

$$\begin{aligned} Cov_j &= \frac{\sum_i w_{i,j} * 1_j\{Peer_i\}}{\sum_i w_{i,j}} \\ \text{Coverage} &= \frac{\sum_j Cov_j}{\text{Number of events}} \end{aligned}$$

where

$$\|\text{Time period T}\|_j = \text{Number of time units of T that occur after } Event_j \text{ is generated}$$

$$1_j\{Peer_i\} = \begin{cases} 1 & Peer_i \text{ heard about } Event_j \\ 0 & \text{otherwise} \end{cases}$$

Event Number of Transmissions. The Event Number of Transmission (or simply the Number of Transmissions/Overhead) is the total number of messages broadcasted for the one source event throughout the simulation. We do not weight the peers in calculating this metric as even if a peer that is on the map for a short time receives notification of an event and rebroadcasts it, we would want to count its transmissions towards the overhead the same way as we count the transmissions made by a peer that was in the simulation for a long time.

Event Latency. We define event latency to be the average number of time units required by a peer to first hear about a generated event. More formally:

$$\begin{aligned} Lat_j &= \frac{\sum_i t_{i,j}}{\text{Number of peers notified of } Event_j} \\ \text{Latency} &= \frac{\sum_j Lat_j}{\text{Number of events}} \end{aligned}$$

where

$$t_{i,j} = \begin{cases} \text{Time taken for } Peer_i \text{ to be first notified about } Event_j \\ 0 \text{ (if } Peer_i \text{ is not notified about } Event_j) \end{cases}$$

It should be noted that peers are not weighted in the calculation of this metric. This is because weighting is only required to account for the fact that all peers do not have equal amount of time to hear about an event; only peers that have heard about the event are included in the latency calculation though, so there is no bias to adjust for.

4.2 Simulation Framework

We have developed a discrete event simulator that implements the above model. There are a number of experimental parameters that we can vary: Scenario parameters refer

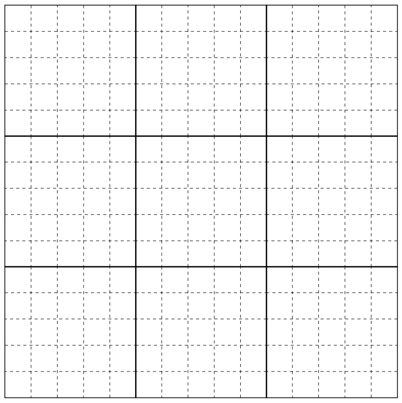

Figure 2. City Block map layout

to those variables that can be used to describe the environment in which the simulation was conducted. They are considered to be out of the control of the event dissemination mechanism. Algorithm parameters, on the hand, comprise of the variables which directly affect the functioning of the flooding protocol and can be tuned by it. In this section we describe the scenario parameters; algorithm parameters (delay start, delay end, TTL, and SR threshold) were discussed in Section 3.

- **Map size** Some mechanisms are more effective at propagating events over wide areas, so at times we vary the size of the map. We describe the map size either in terms of the number of reachable locations (where a car might be present), or in terms of overall dimensions (say, a 20x20 grid), depending on which factor is relevant to the discussion. By default our map is a 41x41 grid in 'city block' layout (refer next paragraph).
- **Map layout** The choice of map layout primarily affects the rate of peer circulation, which may have bearing on the choice of delay factor, for instance. By default we use a 'city block' layout, illustrated in Figure 2, which is a large grid simulating the regular layout of streets in, for example, downtown Manhattan, where each block is five map locations per side. Cars are allowed to enter and leave the simulation on the boundaries of the map where the streets intersect the roads. Any other map layouts that we used are described where relevant.
- **Car density** This parameter is expressed in cars per road location. Since a map layout might include unreachable locations (corresponding to areas without roads), we only consider the number of locations where a car can be present. By default, we set the car density to 1.
- **Simulation time** This is the total amount of time that the simulation is run for measured in ticks (1000 ticks = 1 second). This parameters plays an important role in determining the efficacy of certain time sensitive suppression schemes e.g. TTL. Unless otherwise mentioned, all experiments in this paper had a simulation time of 150,000 ticks. To give a clearer picture of the scale, the maximum speed of a car in our model is 1 grid square every 1000 ticks.
- **Communications radius** The range of radio communications; by default we use a range of 2 map squares, meaning the message is visible to any car at a location within a Manhattan distance of 2 from the sender.
- **Communications bandwidth** We model limited communication bandwidth by varying how long a message takes to transmit. Given a bandwidth B bits/ticks, and a message of b bits, a transmission will use the wireless medium for b/B ticks. As by default the message size used was 10240 bits, and the bandwidth was set to 1000 bits/tick, unless otherwise mentioned it should be assumed that each message used the transmission medium for approx 10 ticks.

Name	**Default Value**
Map Size	41x41 grid
Map layout	city-block layout
Car Density	1.0
Simulation time	150,000 ticks
Communication radius	2 map squares
Communication bandwidth	1000 bits/tick
Number of events	1
Event generation frequency	5000 ticks

Table 1. Scenario Parameters

The default values of the scenario parameters used in the experiments are summarized in Table 1. It is important to note that the absolute values for the parameters are not as important as their relationship to other parameters. Also, note that we vary many of the default values in our experiments.

5. Results

We begin by evaluating the two suppression schemes (TTL and SR) under different fixed delay intervals. After obtaining baseline measurements, we compare the coverage generated and the overhead requirements of TTL and SR. After establishing that both suppression schemes work well in general, we conclude by varying the scenario parameters and gauging their effect on the efficacy of these

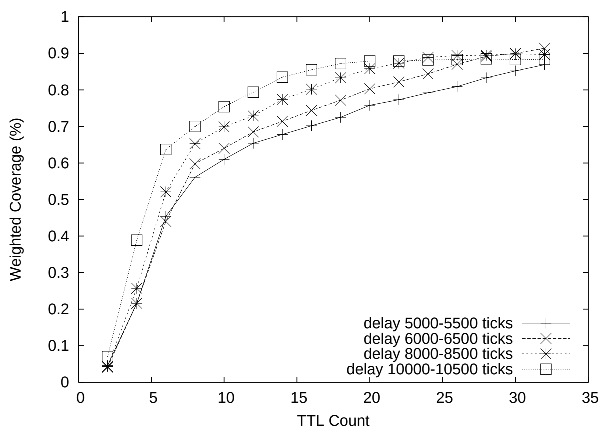

Figure 3. TTL vs. coverage

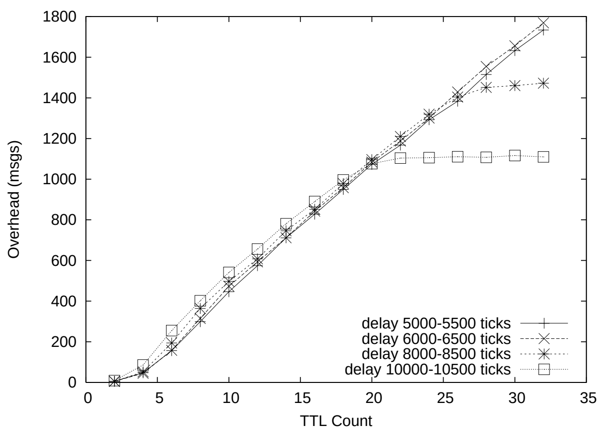

Figure 4. TTL vs. overhead

suppression schemes. Unless otherwise mentioned, the scenario parameters for each experiment are configured to their default values (summarized in Table 1).

5.1. Coverage and Overhead using TTL

The relationship between the suppression count, the delay interval, the coverage, the overhead and the latency for TTL is captured by Figures 3, 4 and 5.

Figure 3 plots the effect of varying the delay start time and the TTL count on coverage. The graph shows that as the TTL count increases the coverage also grows, with algorithms with high TTL counts markedly outperforming those with low counts (e.g., at a delay interval of 8000 -8500 ticks the coverage is 10% when the TTL count is 2, but around 90% when the TTL count is 32). The direct relationship between the TTL count and coverage is in line with our expectations – as the degree of suppression is decreased an event is forwarded a greater number of times. This will increase the probability that a new peer will hear at least one of the transmissions and thus increase the number of peers notified of the event.

The graph also shows that varying the delay significantly affects the coverage, e.g., at a TTL of 16 increasing the delay from around 5000 ticks to 10000 ticks improves the coverage from 65% to 85% – a 30% jump. The reason for this is that long delays gives recipients more time to move away from the location where they originally received the broadcast. This increases the likelihood of their retransmissions reaching previously unnotified neighbors. The fact that delay can affect the coverage in this environment should be contrasted with static systems where delaying a transmission plays a almost no role in improving the efficacy of a broadcast

Figure 4 is an analog of Figure 3 with the y-axis now giving the number of transmissions generated by the broadcasted event. The graph indicates that the while the overhead increases with the TTL count, it is virtually invariant to the amount of delay (the different saturation points at high TTL counts are artifacts of the simulation running for a finite time i.e. algorithms with high delays do not have enough time to forward an event the provided TTL count times). This is consistent with our model of TTL: reducing the amount of suppression will obviously increase the number of messages, but as the suppression is not dependant on the number of peers that have been informed of the event, the overhead will not depend on factors that can affect coverage (such as the delay). As we will see shortly for SR, the near independence between the overhead and the coverage is not always the case.

The above two graphs imply, that for TTL, increasing the delay tends to lead to favorable results – the coverage tends to increase whilst the overhead remains unchanged. Unfortunately, there is a tradeoff in making the delay too large: As the next figure will show, if the delay is incremented by too large an amount peers will be notified of events very slowly – even if the TTL count is high.

Figure 5 plots the effect of varying the TTL count on the latency. The plot shows that as the TTL count is increased a peer tends to hear about the broadcasted event faster, with TTL configurations with short delays providing lower latencies than those with higher delays. This makes intuitive sense: higher TTL counts result in a more messages being broadcasted while shorter delay intervals ensure that these messages are broadcasted more immediately – algorithms having both high counts and low delays will therefore ensure that its peers will hear about events faster than those with high delays and low counts. The initial hump in the graph appears to be an exception to the observation that latency increases with TTL count – it occurs because at low TTL values messages do not spread far from the source (the coverage is very low); so even though the peers that hear of

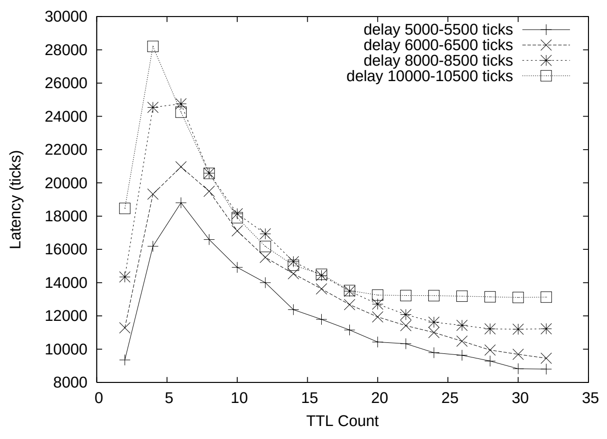

Figure 5. TTL vs. latency

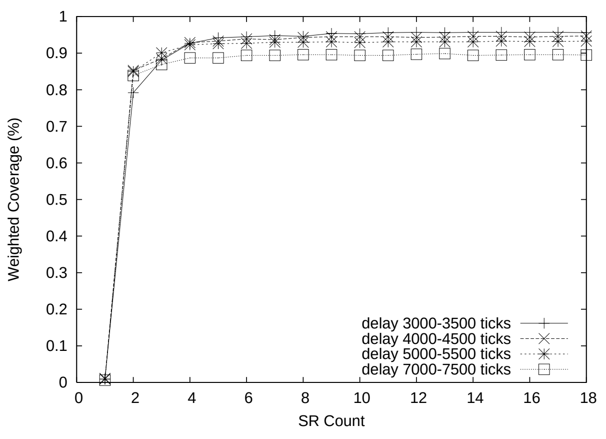

Figure 6. SR vs. coverage

the event will do so quickly, their population is so small the latency measure is not really meaningful.

It should be noted that latency for each delay interval tend to saturate at high TTL counts – this implies that even though incrementing the TTL count helps to reduce latency, the main determinant of this metric is the delay interval. As these saturation values ranges from twice (for delay of [5000,5500] ticks) to 1.5 times the delay (delay [10000,10500] ticks), on average it will take 1.5 - 2 hops for an event to reach a peer for the first time (i.e. if most peers first heard about the event from their initial set of neighbors, the latency would be very close to the delay interval). The fact that latencies for higher delay intervals are smaller multiples of the delays also implies that high delays are more efficient at spreading events – even though they are slower, they require fewer broadcasts to notify peers – than shorter delays.

A discerning reader may note that in the graphs above we varied the delay start time (from 5000 to 10000 ticks), but not the delay increment – it was fixed in all simulations to 500 ticks. This is because our experiments indicated that the size of the delay interval had little impact on the coverage and number of transmissions – its main purpose (analogous to static systems) is to reduce the number of collisions between neighboring peers.

5.2. Coverage and Overhead using SR

Analogous to the experiments in the previous section, we now study the effect of varying the suppression count and delay start on the coverage, overhead and latency using the SR protocol. Figure 6 plots the relationship between the coverage and SR count at different delay intervals, Figure 7 graphs this for the number of transmissions metric, while Figure 8 plots this for the latency.

Figure 6 indicates that, like the TTL protocol, as the SR count is increased the coverage also increases, but unlike TTL the coverage for SR quickly saturates to a (high – around 90%) constant value. Also unlike TTL, in this scheme lower delays in general lead to better coverages than higher delays (95% for the lowest delay as opposed to 90% for the highest delay).

The reason behind these disparities lies in the fundamental difference between SR and TTL. In TTL a count of 2, say, means that each message can be broadcasted a maximum of two times, in SR on the other hand, a count of 2 means that each peer can forward the message two times. This implies that for the same suppression count an SR scheme will always broadcast more messages than TTL. The large number of messages that even a small SR count can generate implies that the coverage will saturate relatively quickly for the SR suppression protocol.

The better coverage for the SR protocol at lower rather than higher delay's is a result of the large number of messages that SR generates. As peers enter and leave the map frequently, if one waits for too long before retransmitting many peers would have left the map and therefore have no chance of being informed of the broadcasted event. When they are sufficient messages, therefore, lower delays will lead to a better coverage as they will reduce the chance of a peer entering and leaving the map within the same delay interval. This effect was not present in TTL as fewer messages are generated, and therefore even though more peers leave the grid at higher delays it is still more efficient to ration out the broadcasts by waiting to maximize the number of new peers encountered. More differences between the SR and TTL suppression scheme are studied in the next section.

Figure 7 captures the effect of varying the SR count on the number of transmissions at different delay intervals. We immediately notice that unlike TTL, in the SR protocol, the message overhead is strongly correlated with the delay – a

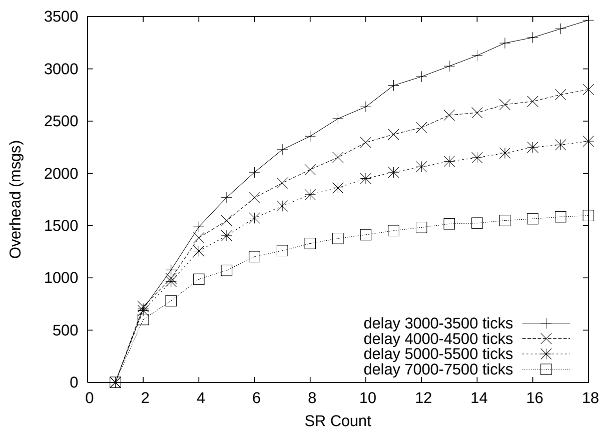

Figure 7. SR vs. overhead

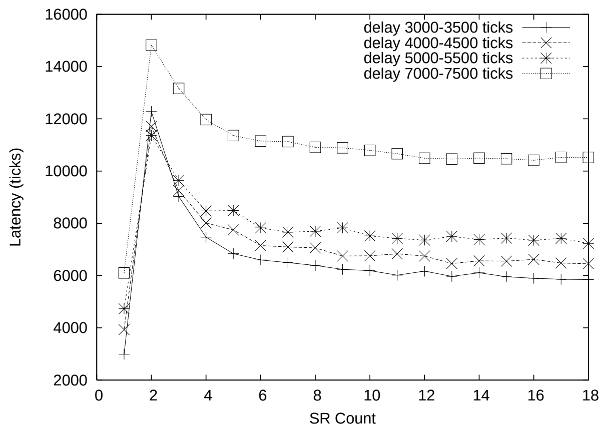

Figure 8. SR vs. latency

delay of 3000 ticks leads to an overhead of 3700 messages (at an SR count of 18), more than double the overhead of 1700 messages at a delay of 7000 ticks (for the same SR count). This is because, unlike TTL, the number of messages generated in SR is highly dependant on the coverage: If the SR count is k and m peers are notified about the event, then the message overhead will be around $k * m$ – each peer will make approx k forwards, and they are m peers (this is only a very rough estimate as it does not account the fact that a peer may not receive a message k times – to then re-forward it k times). Factors that increase coverage (like low delays), will therefore, in this protocol, also noticeably increase the number of transmissions. The converse will hold for factors that decrease the coverage (such as high delays).

Figure 8 plots the relationship between the SR count and the latency. Analogous to TTL, we notice that that the latency again tends to decrease as the SR count is increased (the initial hump is because at low SR counts only a few peers hear of an event – so even though these peers will be notified of the event quickly, their number is so small the latency measure is bereft of any quantitative meaning). Also, similar to TTL we notice that the delay interval is the main factor determining the latency – at high SR counts the latency tends to saturate to values 1.5-2 times the delay. If we compare TTL and SR side by side though (graph not shown) we notice that SR has a slightly lower latency than TTL (e.g., for a delay interval of 5000-5500 ticks SR tends to converge to a latency of 7500 ticks while TTL converges to a latency of 8500 ticks) – this is mainly because SR as tends to broadcast more messages than TTL, peers can hear about an event faster.

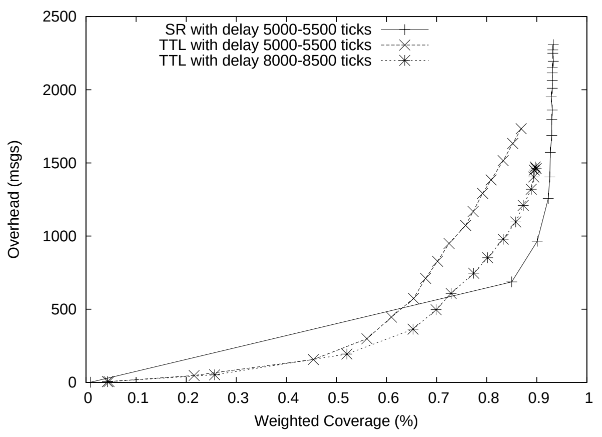

Figure 9. TTL vs. SR

5.3. Comparison of SR vs. TTL

Now that we understand the effect of algorithm parameters on the TTL and SR schemes, we would like to compare them side-by-side. The sections following this one study the effect of varying different scenario parameters on the TTL and SR suppression schemes; we start, however, by comparing the TTL and SR protocols in the default setting used in the previous experiments. We do not include latency comparisons for any of these experiments as, as mentioned previously, our main focus is the coverage and overhead (that being said SR tends to equal or have a lower latency than TTL in the presented environments).

Each point in Figure 9 gives the coverage and overhead for a given SR or TTL count. Ideally we would like as many data points as possible in the lower left hand quadrant of the figure – simulations in this position will have low overhead and high coverage. To compare TTL and SR, we selected the delay setting that made each scheme perform best. That is, we used delays of [5000, 5500] for SR, and of [8000, 8500] for TTL. These configurations results in a high coverage with a reasonable overhead for each suppression scheme (refer previous section). The additional TTL contour was configured with a delay of [5000, 5500 ticks] –

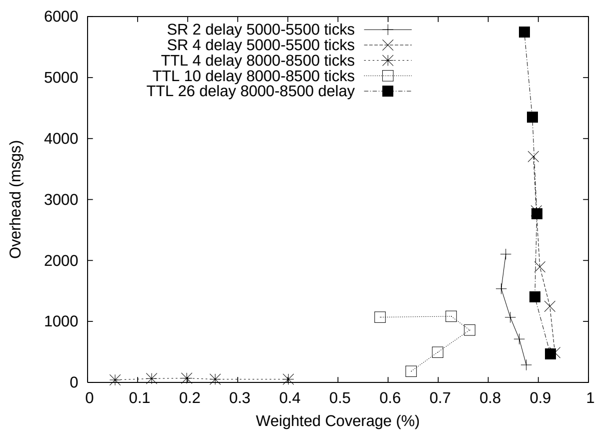

Figure 10. Effect of map size

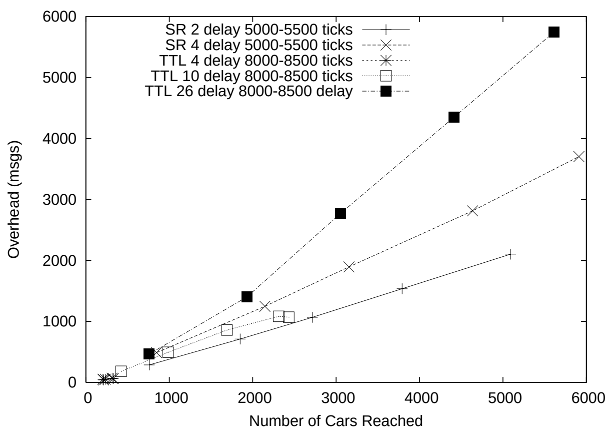

Figure 11. Cars reached

this was to facilitate direct comparison with the SR scheme.

As we travel the SR plot from left to right (Figure 9), the SR count increases from 1 to 18. Similarly, as we follow the TTL plot from left to right the TTL count increases from 2 to 32 (in increments of 2). The graph shows that even though on average each TTL contour has fewer broadcasts than SR, the best coverage achievable by SR exceeds that of TTL (95% for SR vs. 90% for TTL). In addition, it is possible to parameterize the SR algorithm such that it gives a higher coverage and lower overhead than that of any TTL configuration (e.g., SR = 3 has a coverage of 92% and only a 1000 message overhead – far superior to any TTL parameterization). The results indicate that even though SR in general can be configured to greatly outperform TTL, TTL may be preferable in settings where a very low overhead is required and a medium level of coverage (50% - 70%) is acceptable.

Note that SR tends to saturate very quickly (coverage wise). This implies that the SR protocol is very useful in situations when the optimal algorithm parameters are not known in advance: The coverage for even a bad choice will be (most of the time) close to optimal – the overhead may be high though. TTL, on the other hand, does not level-off as fast as SR – it is maybe relevant therefore in environments where a finer level of control is required on the coverage/overhead tradeoff.

5.4. Effect of Map size on SR and TTL

We now vary different scenario parameters and analyze their effect on the SR and the TTL suppression schemes. We do this as we are interested in evaluating the differences between the TTL scheme and the SR protocol in disparate environments. We start by modifying the size of the map.

Figure 10 graphs the effect of varying the map size on the SR and TTL protocols' coverage and number of transmissions. The bottom-most point of each plot gives the coverage and overhead for a map of size 21 x 21. The other extreme for each plot represents the coverage and number of transmissions for a 101 x 101 grid. Each point in between corresponds to a map 20 units longer than the preceding one (i.e. the next map after the 21 x 21 grid is one of length 41). Both SR schemes used (with SR counts of 2 and 4) were parameterized with a delay interval of 5000 to 5500 ticks; the three TTL schemes used (TTL count 4, 10 and 26) were configured with a delay interval of 8000 to 8500 ticks (i.e. both protocols had favorable delay intervals).

The graph shows that the coverage for SR is largely invariant to the map size, the coverage for low TTL counts, on the other hand, is very sensitive to the size of the map (we also expect TTL 26 to also be affected by the map size when the maps used are bigger than those graphed). The number of transmissions for both protocols increase with the map size – this is simply the effect of larger maps having more cars (as the car density is held constant, and also because larger maps have more entry/exit points than smaller maps so they will have more cars over the course of the simulation), and therefore more retransmissions, than smaller maps than anything else. In summary, SR is a much more robust scheme since it can be used, with good results, when the map size is not known in advance.

To analyze the trends more closely, we now plot in Figure 11 the number of cars reached (i.e. the coverage not normalized by the total number of peers in the simulation) versus the overhead for the same map sizes. (As we move from left to right along each line, the map size increases from a 21x21 grid to a 101x101 grid.) Both figures indicate that even though TTL can be configured to alert as many cars as SR, the cost of the TTL scheme (in terms of overhead) will be much larger. The reason for this is that as TTL fixes the number of times an event can be rebroadcast it in essence defines a hop radius around the event outside

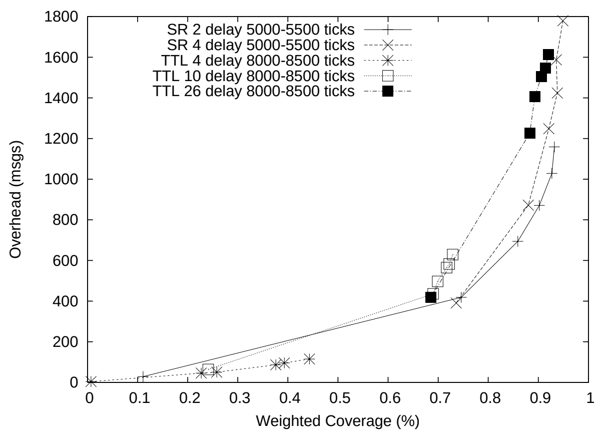

Figure 12. Effect of car density

which the peers cannot receive notification because the TTL count has expired. For larger maps to have a good coverage (i.e. more peers to hear about the event) we therefore have to make the TTL count very large. This in turn leads to a large overhead due to redundant transmissions (i.e. transmissions to peers who already know about the event). The SR scheme does not suffer from this drawback as we only limit the number of times a car can forward an event, and not the number of times the event in general can be retransmitted. This will allow even small SR counts to efficiently allow for a good coverage.

5.5. Effect of Car Density on SR and TTL

The previous sections have indicated that SR is efficient because it encourages each car to broadcast received events many times. As long as the number of cars is 'reasonable' (as in the previous sections) therefore, SR will do well. We would, however, like to design a scheme where the dissemination mechanism gives good performance results even in areas of low traffic. We therefore test the efficacy of our suppression schemes under different car densities – as the map size is fixed, this directly corresponds to varying the average number of cars present in the simulation at any one time. It should be noted that as the TTL count is suppressed on an event basis (and not on a peer basis like SR), we would not expect it to be greatly disadvantaged by varying the number of cars in the system (in fact with fewer cars we would expect TTL to give a better coverage as there will be fewer broadcasts close to the source so notification of the event can spread further out).

Each line in Figure 12 corresponds to the coverage/overhead for a given SR or TTL scheme at a certain density. As before as we move along each contour from left to right the car density increases – the leftmost point corresponds to a car density of $\frac{1}{3}$, the rightmost point on each line a car density $\frac{7}{3}$, with increments of $\frac{1}{3}$. As before, the SR schemes are parameterized with a delay interval of 5000 to 5500 ticks, and the TTL schemes with a delay interval of 8000 to 8500 ticks i.e. both protocols are configured with delay intervals that lead to favorable results.

The plot indicates that, barring the lowest density of $\frac{1}{3}$, SR results in a better coverage than TTL. Moreover, even for TTL configurations that give good results in low densities (e.g., TTL counts of 10 and 26), SR (for all but a density of $\frac{1}{3}$) requires a lower overhead to give the same or better coverage. The graph, however, also shows that while SR tends to outperform TTL, it is hard to throttle coverage wise . TTL therefore may be useful in varying density conditions where the course-grain nature of the SR protocol is not appropriate.

5.6 Related Events

In this section we briefly study the performance of the dissemination schemes when a sequence of events is transmitted by the source. In particular, we assume that the events in the sequence are related, so the recipient may not need to get all of them, just one or just a few. For example, the source may send out a sequence of alerts regarding a particular accident, so it may not be essential to receive all related events.

There are many ways to model and evaluate such a scenario; here we discuss only one option. We refer to the sequence of events as a *topic*, and we assume that a car need only get one of the events on the topic. Thus, we modify our coverage metric accordingly:

$$\begin{aligned} w'_i &= \frac{\text{Time } Peer_i \text{ is in simulation}}{\text{Total simulation time}} \\ \text{Topic Coverage} &= \frac{\sum_i w'_i * 1'\{Peer_i\}}{\sum_i w'_i} \end{aligned}$$

where

$$1'\{Peer_i\} = \begin{cases} 1 & Peer_i \text{ heard about at least} \\ & \text{one event in the topic} \\ 0 & \text{otherwise} \end{cases}$$

The TTL dissemination scheme treats each event from a topic as independent, since it is hard to implement a "combined TTL" across events. However, the overhead of the SR scheme can be reduced by treating events on the same topic as copies, but this change would impact coverage.

To keep our comparison simple, here we compare the unmodified algorithms as applied to topic dissemination. In particular, the source transmits an event every 5000 ticks, for a total of 10 events, and the cars disseminate each event independently.

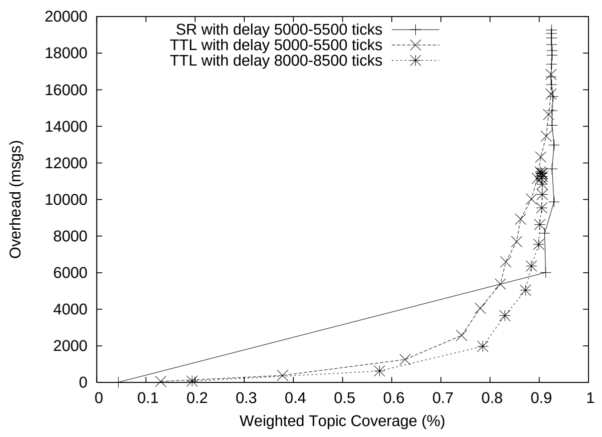

Figure 13. Topics

Figure 13 shows the topic coverage (as defined above) versus the total overhead (for all events), for the TTL and SR schemes. (All other parameters are as before.) As we move from left to right the SR count increases from 1 to 18 and the TTL count increases from 2 to 32 (in increments of 2).

The graph indicates that even with topic dissemination, the SR protocol outperforms TTL, i.e., the SR plot is always to the left of the TTL contours (implying that its coverage is always higher than TTL for the same overhead). In this particular scenario, TTL (with a count of about 6) may be preferable to SR: even though the coverage will be lower (around 80% as compared to 90%) the overhead will be significantly lower (around 2000 messages).

Comparing Figure 13 to Figure 9, we see that coverage is not markedly improved over the single event scenario, even though the overhead is significantly higher. This is because in our scenario, the single event broadcast reaches most cars that can be reached. However, note that coverage improves faster as SR (and TTL) increase. For example, when the SR threshold is 2 in Figure 9, coverage is about 0.87. In the same situation, the topic coverage in Figure 13 is 0.92.

6. Conclusion

Although we did not show results for a traditional dissemination algorithm, it is easy to see that it would have much less coverage than our mobile dissemination algorithms. Our mobile dissemination algorithms take advantage of the car's motion and changing topology, to carry events further and to more cars. Our results show that our new SR algorithm performs significantly better that a TTL-based scheme, while still being very simple to implement. Since flooding-based dissemination is a key function for highly dynamic networks, we believe that the SR algorithm can play a significant role in such an environment. In addition, the framework and evaluation metrics we have introduced can be useful for studying other algorithms in this dynamic environment.

References

[1] A. Bachir and A. Benslimane. A multicast protocol in ad hoc networks inter-vehicle geocast. In *Vehicular Technology Conference*, Spring 2003.

[2] V. Bharghavan, A. J. Demers, S. Shenker, and L. Zhang. MACAW: A media access protocol for wireless LAN's. In *SIGCOMM*, pages 212–225, 1994.

[3] L. Briesemeister and G. Hommel. Role-based multicast in highly mobile but sparsely connected ad hoc networks. In *International Symposium on Mobile Ad Hoc Networking and Computing*, pages 45–50. IEEE Press, 2000.

[4] A. Demers, D. Greene, C. Hauser, W. Irish, and J. Larson. Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12. ACM Press, 1987.

[5] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile enviroment. In *Second ACM international workshop on Data engineering for wireless and mobile access*, pages 27–34. ACM Press, 2001.

[6] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In *Computer Communications Review – Proceedings of SIGCOMM '96*, August 1996.

[7] P. Karn. MACA - a new channel access method for packet radio. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 134–140, 1990.

[8] K. Murauyama, M. Tanizaki, and S. Shimada. Evaluation of ip handover for dsrc network, 2002.

[9] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the Second IEEE Workshop in Mobile Computing*, 1999.

[10] S. Singh and C. Raghavendra. PAMAS: Power aware multi-access protocol with signalling for ad hoc networks, 1999.

[11] Q. Sun and H. Garcia-Molina. Using ad-hoc inter-vehicle networks for regional alerts, 2005.

[12] J. Weinmiller, H. Woesner, and A. Wolisz. Analyzing and improving the IEEE 802.11-MAC protocol for wireless LANs.