# Prioritized Gossip in Vehicular Networks*

Alejandro Cornejo
MIT CSAIL
32 Vassar St
Cambridge MA 02139
acornejo@csail.mit.edu

Calvin Newport
MIT CSAIL
32 Vassar St
Cambridge MA 02139
cnewport@csail.mit.edu

## ABSTRACT

We present a method for using real world mobility traces to identify tractable theoretical models for the study of distributed algorithms in mobile networks. We validate the method by deriving a vehicular ad hoc network model from a large corpus of position data generated by Boston-area taxicabs. Unlike previous work, our model does not assume global connectivity or eventual stability; it instead assumes only that some subset of processes are connected through *transient paths* (e.g., paths that exist over time). We use this model to study the problem of prioritized gossip, in which processes attempt to disseminate messages of different priority. Specifically, we present CabChat, a distributed prioritized gossip algorithm that leverages an interesting connection to the classic *Tower of Hanoi* problem to schedule the broadcast of packets of different priorities. Whereas previous studies of gossip leverage strong connectivity or stabilization assumptions to prove the time complexity of global termination, in our model, with its weak assumptions, we instead analyze CabChat with respect to its ability to deliver a high proportion of high priority messages over the transient paths that happen to exist in a given execution.

## Categories and Subject Descriptors

C.2.4 [**Computer Communication Networks**]: Distributed Systems; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

## General Terms

Theory, Algorithms

## Keywords

Radio Networks, Gossip, Vehicular Networking

*This work was supported in part by AFOSR (Award Number FA9550-08-1-0159), NSF (Award Numbers CNS-0715397 and CCF-0726514) and Mobile Mesh Networks (Ford-MIT Alliance Agreement January 2008).


*DIALM-POMC'10,* September 16, 2010, Cambridge, MA, USA.


## 1. INTRODUCTION

A difficulty in studying distributed algorithms for mobile networks is defining realistic mobility. A common solution to this difficulty is to use position traces from real mobile network deployments. In recent work, for example, Liu et al. [13] use traces of San Francisco-area taxicabs to study the performance of their *VMesh* strategy for local information storage, and Sarafijanovic-Djukic et al. [16] use traces from cabs in Warsaw to study an *island hopping* strategy for routing. In these two papers, as in most other data-driven analyses of mobile network algorithms, the position traces are used to support *simulation studies*. In this paper, by contrast, we propose using traces to derive models suitable for generating *theoretical results*.

**Restricted Dynamic Graphs.** Specifically, we begin with the *dynamic graph* model of [12], which describes the connectivity of processes in a mobile network as a graph in which the edge set can change arbitrarily from round to round. (An edge between two nodes in a given round indicates the ability for the associated processes to communicate in that round.) We then propose using position traces from real mobile networks to identify properties of these graphs that arise in practice. These properties define a restricted dynamic graph model. The goal is to identify properties that allow a theoretician to generate better algorithms and bounds than what is possible with the unrestricted dynamic graph model, while at the same time maintaining the results' applicability in practice.

For example: imagine that the study of buses traveling on a fixed bus route reveals a small amount of connectivity at any one time step, but a high probability that an arbitrary pair of buses will eventually be connected (e.g., as they pass each other on the route). This might inspire the dynamic graph property that a given pair of nodes are expected to be connected in the graph, for at least $x$ rounds every $T$ rounds, with high probability, where $x$ is a small constant and $T$ is a large constant, both derived from the bus traces. When analyzing a distributed algorithm to be deployed on buses, this data-derived property can be used to tame the otherwise arbitrary edge changes in the graph. A result proved in this model is likely to hold in the real world network from which the property was derived.

**A Data-Derived Dynamic Graph Property.** To validate the usefulness of this modeling approach, we study vehicular ad hoc networks (VANETs) comprised of taxicabs in an urban setting. The source of our experimental observations is a large corpus of position traces gathered from GPS-equipped embedded computers deployed by the MIT CarTel

project [6] in a fleet of taxicabs in the Boston metropolitan area. We present a framework for combining these traces into VANET networks of varying size (in this study, we examine networks of 25 and 100 vehicles), and varying traffic conditions (we examine morning and evening rush hour, and the midday lull).

We next examine the properties of the connectivity graphs induced by these networks, first showing that there is never global connectivity: in our 100 vehicle networks, for example, the largest connected component observed in any round contained no more than 35 vehicles. What we instead observe is a large amount of *transient connectivity* between vehicles—e.g., paths over time—with over a quarter of the vehicles in the 100 vehicle networks having transient connectivity to a majority of the network. We also observe moderately stable pairwise links, with a median link duration of 16 seconds. We combine these observations of transient connectivity and pairwise link stability into what we call the *$\ell$-stable transient path* property, which describes a transient path between two vehicles such that each hop in the path exists for at least $\ell$ consecutive rounds.[1] In this paper, when we analyze the performance of distributed algorithms, we do so in the dynamic graph model under the assumption that such paths exist; e.g., *given an $\ell$-stable path between nodes $u$ and $v$ in the dynamic graph, starting at round $r$, we prove the following performance result...*

**The Prioritized Gossip Problem.** With our data-driven model defined, we turn our attention to solving a specific problem. A commonly-cited use for vehicular networks is the dissemination of timely information between vehicles [4, 22], for example: an observation about traffic, the location of a road-side access point, or an accident alert. This problem can be cast as a form of gossip, in which vehicles occasionally generate *messages* of different priorities that need to be disseminated. Presumably, an accident alert, for example, would have higher priority than an observation of a traffic jam. We refer to this problem as *prioritized gossip*, and we study it, in this paper, in the context of the dynamic graph model with the assumption of $\ell$-stable transient paths.

A challenge for gossip in our setting is the lack of strong connectivity assumptions. In contrast to previous work [12], we do not assume global connectivity (or even that every pair has transient connectivity), and this prevents us from proving the time complexity of global termination (e.g., *the gossip problem terminates in $O(n^2)$ rounds*). Instead, we only assume that *some* pairs are connected by an $\ell$-stable transient path, for varying $\ell$ values, leaving the designer of prioritized gossip algorithms to prove their algorithms leverage such paths, when they arise, to deliver as much high priority information as possible. Such results are weaker than those guaranteeing global termination, but because they make less connectivity assumptions they are applicable in a wider variety of practical settings.

Another challenge of solving gossip in our model is the presence of priorities. Without priorities, it is sufficient for processes to work through their message queue in round robin order, broadcasting a new message in each round: this behavior guarantees that over any $\ell$-stable transient path, $\ell$ different messages are delivered. Priorities, however, complicate this approach, as we not only desire to send *unique* messages, but we also want to send *high priority* messages. Imagine, for example, a process $u$ with an $\ell$-stable transient path to $v$, and a message queue of size much larger than $\ell$. The round robin approach might lead $u$ to deliver $\ell$ low priority messages during the $\ell$ rounds it participates in the path. A good prioritized gossip aglorithm, therefore, must be careful in how it schedules its messages for broadcast.

**The $t$-Latency Metric.** To capture the effectiveness of a given gossip algorithm's priority scheduling scheme, we introduce the *$t$-latency* metric, which upper bounds the number of rounds required for a process to broadcast its $t$ highest priority messages, over all rounds in which it has at least $t$ messages, over all executions. An algorithm that guarantees a small $t$-latency with respect to $t$, for all $t$ values, will deliver a high proportion of high priority messages at each hop of an $\ell$-stable transient path. Our main performance theorems, summarized below, will leverage $t$-latency results proved with respect to our algorithm, CabChat, to bound the amount of high priority information the algorithm guarantees to be sent over a given $\ell$-stable transient path.

**Our Results.** We present CabChat, a distributed prioritized gossip algorithm that leverages properties of a slight variation of the *binary carry sequence* [1], which also describes an optimal solution the classic *Towers of Hanoi* problem [18, 11].[2] To aid the proof of our main performance theorems, we start by bounding the algorithm's $t$-latency. In the general case, we show that the algorithm guarantees a $t$-latency of $2^{t+1}$, for all $t$, and in the case where the $t$ highest priority values span only $k < t$ distinct priorities, it guarantees a $t$-latency of $(t-k+2)2^{k+1}$. Using these results, we prove **two main performance theorems**:

(1) If at round $r$ process $u$ knows $t$ messages of priority at most $p$ (assume smaller values are higher priority), and there is an $\ell$-stable transient path from $u$ to $v$ starting at $r$ and ending at $r'$, then $\Omega(\min(\log(\ell), t))$ messages of priority at most $p$ eventually reach $v$ by $r'$. This result indicates that as the bandwidth available on a path grows (i.e., as $\ell$ increases), so does the total amount of high priority information guaranteed to be delivered over this path (i.e., $u$'s $\log(\ell)$ highest priority values).

(2) If in addition to the assumptions of the first performance result, the $t$ highest priority messages at process $u$ span a constant number of priorities (for example, if $u$ has a collection of $t$ accident alerts, all sharing the same high priority), then $\Omega(\min(\ell, t))$ messages of this priority eventually reach $v$ by $r'$. Notice, because $\ell$ messages is the maximum number that can be communicated over an $\ell$-stable transient path, this second result indicates that CabChat behaves optimally when delivering many messages from a small number of priorities. This result is important as in practice we would like the very highest priority messages to take precedence over other communication.

## *Related Work.*

Though the global properties of dynamic graphs—i.e., graphs with edge sets that can change over time—have been studied from a complexity perspective for many years (see [17] for

[1]Notice, capturing the stability of the hops is important as it bounds the total amount of information that can flow through the path. An $\ell$-stable path between $u$ and $v$ allows $u$ to transmit $\ell$ messages to $v$, assuming a rate of one message per round.

[2]This versatile sequence has also been used to identify *Hamiltonian paths* in hypercube graphs and generate *binary reflected codes*, also known as Gray codes.)

a good overview), in the last decade, models based on such graphs have been increasingly used to study the performance of distributed algorithms. This direction gained momentum with the *stabilizing dynamic graph* model—c.f., [7, 20, 21, 14]—which describes device connectivity as a dynamic graph with an edge set that can change arbitrarily from round to round. Most results in this model assume that changes to edge set eventually stop, and therefore prove properties with respect to these stabilization points.

To avoid the assumption of stabilizing connectivity, the authors in [12] introduce the *non-stabilizing dynamic graph* model (which we refer to in this paper as simply the *dynamic graph model*). In this model, the edge set never stops changing, but some properties on the graph are assumed to hold. In [12], for example, the authors assume that a connected backbone exists in every round. In our work, we use position traces from real mobile network deployments to identify suitable connectivity properties which hold in practice. (As mentioned, for example, we found that in VANETs comprised of Boston-area taxicabs, the global connectivity assumption of [12] never holds.) We are not the first to use data from mobile networks to derive theory models. Chaintreau et al. [3], for example, use network logs to derive a distribution on the time before two processes meet in a mobile network; they then use this distribution to prove performance theorems.

The gossip problem, of course, has been studied in numerous models (see [5] for a survey of classical results and [9, 8, 2] for a sampling of more recent work). The results most relevant to ours come from the aforementioned study by Kuhn et al. [12], which examines all-to-all gossip in a dynamic graph under the assumption of global connectivity. This strong assumption allows them to prove the time complexity of global termination. As mentioned above, due to weak connectivity assumptions of our model, we cannot prove results regarding global termination. Instead, we study the amount of information, and its priority, that is delivered through transient paths that happen to exist in an execution. Though the notion of a transient path has been studied previously—c.f., [10], which calls such paths *time-respecting*, and analyzes the computational complexity of finding such paths in a dynamic graph—our work is the first, that we know of, to combine the notion with stability and generate performance results for distributed gossip. Accordingly, we have no points of comparison for our results.

### *Road Map.*

We continue, in Section 2, by analyzing the connectivity of real vehicular networks. In Section 3, we use these observations to help define our formal model. Then, in Section 4 we define the prioritized gossip problem, present CabChat, our distributed prioritized gossip algorithm, and prove a pair of performance theorems. We conclude in Section 5 with a discussion of future work.

## 2. BEHAVIOR OF REAL WORLD VEHICULAR NETWORKS

We begin by studying the connectivity properties of vehicular networks comprised of taxicabs in the Boston area. In Section 3, we use these observations when formalizing our mobile network model. The source of our data is MIT's CarTel [6] project, which over the past four years has maintained GPS-equipped embedded computers in a fleet of 20 - 40 taxicabs that service the Boston metropolitan area. These computers report the cabs' changing GPS coordinates to a central server, which uses a rigorous *map-matching* process [19] to transform the raw data into error-corrected position traces. For the purposes of this study, we developed a framework for combining traces from different days while preserving the traffic conditions of a specific hour of a specific day. (In vehicular network research, the traffic conditions are considered an important feature of the network behavior, making it bad practice to combine traces from different days without normalization of traffic conditions.) This framework allows us to study large networks (i.e., networks larger than the number of cabs from which we have data) while maintaining consistent traffic behavior.

In more detail, we studied 15 different traffic conditions—5 drawn from morning rush hour, 5 from midday, and 5 from evening rush hour—and for each traffic condition we studied two network sizes—25 vehicles and 100 vehicles—for a total of 30 different *experiment scenarios*. In each scenario, we divided time into *rounds* of length 1 second, and for each round calculated the communication graph of the network at that round by using a simple 100 meter threshold to determine whether a given pair of vehicles is connected.[3] The result was 30 evolving communication graphs, representing a variety of traffic conditions and network sizes. We analyzed these graphs to identify commonly occurring connectivity properties in real world vehicular networks. We summarize the results of this analysis below.

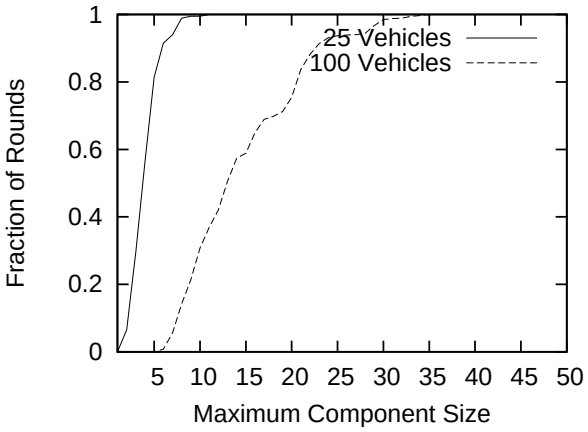


**Figure 1: Cumulative distribution function plot of the maximum component sizes for both the 100 vehicle and 25 vehicle graphs.**

### *Global Connectivity.*

A common assumption in the study of mobile networks is that the communication graph is connected; e.g., [12]. To test this hypothesis we calculated for each round, over all 30 experiment scenarios, the largest connected component

[3]The choice of the 100 meter threshold was derived from a series of vehicle tests conducted in the Boston-area using dashboard-mounted wireless laptops communicating using 802.11g [15]. Although the distance threshold is only a rough approximation of the real network connectivity, it is sufficient for studying high-level connectivity traits.

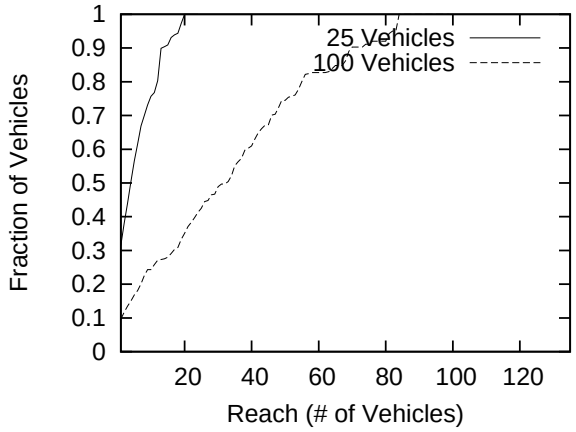


**Figure 2: Cumulative distribution function plot of the *reach* over all vehicles for both the 100 vehicle and 25 vehicle graphs.**

in the communication graph at that round. In Figure 1 we plot the cumulative distribution function (CDF) of these maximum component sizes, split by network size. (In this CDF, a point $f(x)$ describes the fraction of rounds with a maximum component size less than or equal to $x$.) Notice, in the 100 vehicle graphs the maximum component is always of size less than 35, and in the 25 vehicle graphs, the maximum component is always of size less than or equal to 10. Put another way, the network is *never* connected, though small connected components are common. This yields the following observation:

**Observation #1:** The networks are *never* connected but usually *do* contain connected components of non-trivial size.

### *Transient Connectivity.*

The lack of global connectivity does not rule out the existence of *transient connectivity*; that is, paths that exist over time. For example, imagine a dynamic graph defined over two rounds with three nodes $a$, $b$ and $c$, and the following time varying edge set: In the first round, $a$ is connected to $b$, and in the second round $b$ is connected $c$. In this example, $a$ is transiently connected to $c$, even though there is no path from $a$ and $c$ present in either of the two rounds for which the graph is defined.

To measure transient connectivity we use the *reach* metric. The *reach* of a vehicle $a$ in a given graph is the total number of vehicles—excluding $a$—to which $a$ is transiently connected. For each graph we calculated the reach of each vehicle. In Figure 2 we plot the cumulative distribution function of these reach values, split by network size. (In this CDF, a point $f(x)$ describes the fraction of vehicles with a reach value less than or equal to $x$.) The plot reveals significant transient connectivity. In the 100 vehicle experiments, for example, around 60% of the vehicles had a reach of at least 20 vehicles, around a fourth of the vehicles had a reach of at least 50 vehicles, and around 10% were *super-connectors*, with a reach including at least 75% of the vehicles. We summarize these results as follows:

**Observation #2:** Most vehicles have transient connectivity to a non-trivial fraction of the network, while a non-trivial fraction of the vehicles have transient connectivity to most of the network.

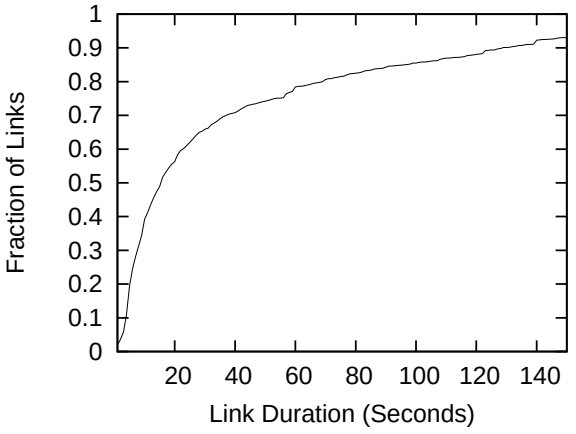


**Figure 3: Cumulative distribution function plot of link duration over all links observed in the graphs.**

### *Link Stability.*

We conclude our analysis by looking at the stability of pairwise links. In Figure 3 we plot the cumulative distribution function of the link durations of all links observed in our networks. (In this CDF, a point $f(x)$ describes the fraction of links with a duration less than or equal to $x$.) Notice that fleeting links are rare—less than 2% of the links are of the minimum length of 1 second—and long links are common—around 20% of links have a duration of at least a minute. The median link duration was 16 seconds. We summarize these results as follows:

**Observation #3:** Extremely short-lived links are rare.

## 3. MODEL

In this paper, we use the dynamic graph model of [12], combined with a connectivity property inspired by the experimental analysis of Section 2. Formally, the dynamic graph model describes a synchronous radio broadcast network with a *dynamic graph* $G = (V, E)$, where $V$ is a static set of nodes, and $E : \mathbb{N} \to \{\{u, v\} | u, v \in V\}$ is a function mapping each round number $r \in \mathbb{N}$ to a set of undirected edges $E(r)$, describing the connectivity in the round. (In this paper we assume the natural numbers $\mathbb{N}$ start from 1). Notice, this model is worst-case, in that it allows the connectivity to change, perhaps significantly, from round to round.

An *algorithm* $\mathcal{A}$ is a collection of $|V|$ *processes*, one for each node in the graph. In this paper, when discussing an algorithm, we use the notation *process* $u$, for some $u \in V$, to reference the process associated with node $u$.

An execution of a given algorithm in a given dynamic graph proceeds as follows. In each round $r$, each process $u$ receives a set (potentially empty) of input values. (The type of these values depends on the problem being studied.) Next, each process chooses a message to broadcast (if any). Each process then receives all messages sent during that round from a neighbor in $E(r)$. That is, communication is reliable, and each process can only broadcast a single message per round. We assume each process has a unique

identifier but no advance knowledge of the size of the network or their neighbors in a given round.

As described in [12], to prove useful results in the dynamic graph model it is often necessary to assume some additional properties on the graph. In [12], for example, the authors assume the graph is connected in every round. In this paper, we use the analysis of real vehicular networks, described in Section 2, to identify such a property. Specifically, we know from Observation 1 that the graph is not globally connected. However, from Observation 2 we know that many nodes have transient paths, and Observation 3 tells us that the individual hops of these transient paths are likely to be stable. We combine these observations into the concept of an *$\ell$-stable transient path*, which is a path that exists over time (but not necessarily all at once), such that each hop is stable for $\ell$ consecutive rounds.

We now formally define a *transient path*:

**Definition 1** (Transient Path)**.** Given a dynamic graph, a transient path in the graph at round $r$ from node $u$ to $v$ is defined by a sequence of rounds $r_1 = r, \ldots, \ldots, r_m = r + d$ and a sequence of edges $e_1 = (u, u_1), e_2 = (u_1, u_2), \ldots, e_m = (u_{m-1}, v)$ such that: **i)** $\forall i \in [1, m-1]$, $r_i < r_{i+1}$, and **ii)** $\forall i \in [1, m]$, edge $e_i$ exists in the graph at round $r_i$. Here $m$ is the *length* of the path and $d$ is its *duration*.

Notice that the duration of a path can be greater than its length (but not the other way around), as arbitrarily long intervals of time can exist between hops.

We now define what it means for a transient path to be *$\ell$-stable*:

**Definition 2** ($\ell$-Stable Transient Path)**.** Given a dynamic graph, a transient path in the graph described by the round sequence $r_1, \ldots, r_m$ and the edge sequence $e_1, \ldots, e_m$ is $\ell$-stable if **i)** $\forall i \in [1, m]$, the edge $e_i$ exists in the graph throughout the interval $[r_i, r_i + \ell - 1]$, and **ii)** $\forall i \in [1, m-1]$, $r_{i+1} \geq r_i + \ell$.

In other words, a transient path is $\ell$-stable if each hop exists for $\ell$ consecutive rounds, and no two hops' round intervals overlap.

# 4. PRIORITIZED GOSSIP

The prioritized gossip problem requires processes to disseminate messages of various priorities, giving precedence to messages of higher priority. As mentioned in the introduction, due to the weak connectivity assumptions in our model, we have no notion of *solving* the prioritized gossip problem in the traditional sense of completing all-to-all message exchange. We instead turn our attention to how well an algorithm takes advantage of the connectivity that happens to exist in an execution to deliver messages of high priority.

In more detail, our main performance theorems bound the number of messages, and their priority, delivered over the $\ell$-stable transient paths that exist in a give execution. To aid the proof of these results, we introduce the *$t$-latency* metric (formally defined later in this section). This metric upper bounds the time required for a process to broadcast its $t$ highest priority messages. A core difficulty of prioritized gossip in our model is the need to maximize both the total number of unique messages *and their priority*, sent over an $\ell$-stable path. The $t$-latency metric captures an algorithm's performance in terms of this goal. For example, an optimal algorithm, would guarantee a $t$-latency of $t$, for all $t$. This would ensure that given an $\ell$-stable path between some $u$ and $v$, starting at round $r$, $v$ would learn the $\ell$-highest priority values known to $u$ at $r$ (or $\ell$ values of the same or higher priority).[4] Unfortunately, no such optimal algorithm exists. To see why, notice that to satisfy the property for $t = 1$, processes must always send their single highest priority message, but this generates an infinite $t$-latency for all $t > 1$. (A straightforward extension of this argument demonstrates the impossibility of guaranteeing a $t$-latency of $ct$, for all $t$ and a positive constant $c$.)

The CabChat algorithm presented in this paper, by contrast, guarantees a $t$-latency of $O(2^t)$, for all $t$. For small $t$, this value is small, ensuring that high priority messages are disseminated on $\ell$-stable paths defined with a small $\ell$ value. For large $t$, this value is large but bounded, ensuring that as bandwidth increases on a path (i.e., $\ell$ gets larger), so does the amount of unique messages that will be delivered. We also show that in the special case where the $t$ highest priority values span only a constant number of priorities (e.g., if a process has learned a large number of high priority alerts) the algorithm approximates the optimal throughput by achieving a $t$-latency of $O(t)$.

From these two $t$-latency results we then derive our two main performance theorems. From the former result, we can prove that given an $\ell$-stable path between $u$ and $v$, $u$ will deliver its $\Omega(\min(\log(\ell), t))$ highest priority messages to $v$ in time proportional to the duration of the path (Theorem 4.6). Using the latter $t$-latency result, we improve this bound to $\Omega(\min(\ell, t))$, under the assumption that the $t$ highest priority messages at $u$ span no more than a constant number of priorities (Theorem 4.7).

Below, we start by formally defining the prioritized gossip problem and the $t$-latency metric. We then introduce the CabChat algorithm and analyze its $t$-latency. We conclude by using these latency results to prove a pair of performance theorems.

## 4.1 Problem Description

The prioritized gossip problem assumes the following. At the beginning of each round, each process produces a set of messages (perhaps empty) to disseminate. Each message is labeled with a *priority* value from $\mathbb{N}$. We use $\rho(m)$ to denote the priority of message $m$. The lower the priority value, the higher the message's priority, where 1 is the highest possible priority. Without loss of generality, we assume all messages are unique.[5] In each round, each process $u$ can choose a single message, from among those it knows (which were either generated by process $u$, or were received in an earlier round), to broadcast to its neighbors.

**The $t$-Latency Metric.** To characterize the scheduling behavior of a prioritized gossip algorithm, we introduce the *$t$-latency* metric. Informally speaking, the $t$-latency captures the maximum delay incurred by a process when sending its $t$ highest priority messages. Formally:

[4]This follows from a simple induction and the observation that at each hop $(w, w')$ along the transient path, $w$ has sufficient time to send its $\ell$ highest priority messages to $w'$.

[5]This can be accomplished, for example, by having each process append its id and the current round number to each message it generates. In the case of more than one message is generated in the same round, processes can also append a sequence number.

**Definition 3** ($t$-latency)**.** Fix an execution of an algorithm $\mathcal{A}$ in some dynamic graph. We say $\mathcal{A}$ has $t$-latency $T$ for process $u$ at round $r$ (where $t, r, T \in \mathbb{N}$) if it holds that:

> If process $u$ knows at least $t$ messages at round $r$, and the $t$ messages of smallest priority value have priority at most $p$, then by round $r+T$ process $u$ broadcasts at least $t$ messages of priority at most $p$.

For simplicity, when we omit $u$ and $r$, the property is assumed to hold for all processes and rounds over all executions and over all dynamic graphs.

## 4.2 CabChat Algorithm

In this section we present CabChat, a deterministic distributed prioritized gossip algorithm. The CabChat algorithm uses a priority scheduling function based upon the *binary carry sequence* [1]. This sequence has many useful properties, one of them being that if cast as a schedule, it schedules every positive integer $i$ every $2^i$ rounds. It also guarantees that whenever integer $i$ is scheduled, in a vicinity of size $2^{i-2}$ on either side, it contains all positive integers $j < i$. These two properties will be useful when proving our performance results. Before continuing with the description of CabChat, we describe formally this sequence and its properties.

### *Binary Carry Sequence.*

The priority scheduling function, *schedule*, used by CabChat is exactly the binary carry sequence plus 1. Formally, for every round $r \in \mathbb{N}$:

$$schedule(r) = \max\{i \in \mathbb{N} : r \mod 2^{i-1} = 0\}.$$

The following is a well known fact regarding the binary carry sequence that carries over to our scheduling function.

**Fact 1.** The function *schedule* returns each value $i \in \mathbb{N}$ every $2^i$ rounds.

This sequence also has an interesting connection to the Tower of Hanoi problem [11, 18]. Specifically, given three rods and $n$ disks labeled in descending size from 1 to $n$, the sequence $S_n$ (defined below) gives an optimal solution to the three rod Tower of Hanoi problem. It is well know that the sequence $S_n$ can be defined recursively as follows:

$$S_1 = 1$$
$$S_n = S_{n-1}, n, S_{n-1}$$

Perhaps surprisingly, it was shown that the first $2^n - 1$ numbers of the binary carry sequence plus one (which is exactly our schedule) equals $S_n$. The following fact of the scheduling function follows as a straightforward consequence of the recursive definition of $S_n$:

**Fact 2.** If $i = schedule(r)$ then it holds that:
**i)** $\forall j < i\ \exists r' \in [r - 2^{i-2}, r)$ such that $j = schedule(r')$, and
**ii)** $\forall j < i\ \exists r' \in (r, r + 2^{i-2}]$ such that $j = schedule(r')$.

With our scheduling function and its properties defined, we now describe the CabChat algorithm which leverages the properties of this schedule.

### *Algorithm Description.*

We begin with a high-level description of the CabChat algorithm. The detailed pseudo-code is presented in Algorithm 1. Each process maintains an initially empty linked list where each node in the list has a non-empty message queue. Let $Q_i$ denote the $i^{\text{th}}$ queue in the list. Queues are only manipulated using the SAMPLE and PUSH operations. Specifically, executing SAMPLE($Q$) returns the element at the front of queue $Q$, while at the same time popping the element and pushing it at the back of the queue. Executing PUSH($Q, m$) pushes message $m$ into the front of queue $Q$. Informally speaking, a queue acts as a ring buffer which samples most recently added messages, hence implementing a round robin schedule of sorts. At the beginning of round $r$, let $i = schedule(r)$, if the list has at least $i$ queues, a process retrieves queue $Q_i$ and broadcasts the message returned by the operation SAMPLE($Q_i$). If the list does not contain an $i^{\text{th}}$ queue, the process broadcasts nothing. After broadcasting a message, a process gets a set (possibly empty) of *incoming* messages which were either received from neighboring processes or generated by the process itself. For each message $m \in incoming$, if there exists a queue $Q$ such that $\rho(Q) = \rho(m)$, the process executes PUSH($Q, m$). Otherwise a new queue is created with message $m$, and this queue is subsequently inserted in the correct place in the list as to maintain the priority ordering. Notice, that at round $r = schedule(i)$ this algorithm does not necessarily broadcast a message with *priority* $i$. Instead it broadcasts a message with the $i^{\text{th}}$ *highest priority* from among the messages a process knows, but the actually priority value can be arbitrarily large. This is an important distinction, as our results are stated in terms of the highest priorities a process knows, not specific priority values.

**Algorithm 1** CabChat

```
1: list ← ∅
2: for r = 1, 2, ... do
3:     i ← schedule(r)
4:     if |list| ≥ i then
5:         broadcast SAMPLE(Q_i)
6:     end if
7:     incoming ← messages received and generated at round r
8:     for m ∈ incoming do
9:         if ∃Q ∈ list such that ρ(Q) = ρ(m) then
10:            PUSH(Q, m)
11:        else
12:            insert new queue with message m in list respecting the ordering
13:        end if
14:    end for
15: end for
```

## 4.3 Correctness Proof

At the conclusion of this section we prove two main performance theorems. The first, Theorem 4.6, says that if a process $u$ knows $t$ messages, and an $\ell$-stable transient path exists from $u$ to $v$, then $v$ will receive $u$'s $\Omega(min(\log(\ell), t))$ highest priority messages (or a collection of messages of similar or better priority) in time proportional to the duration of the path. The second result, Theorem 4.7, says that in the special case when the number of priorities spanned by the

$t$ messages with best priority at $u$ is a constant, the bound improves to $\Omega(min(\ell, t))$.

To prove these results, we first establish some useful invariants that follow from the way queues are manipulated by our algorithm. Aided by these invariants we then prove that regardless of the number of arriving messages at every round, and the priority distribution of these messages, CabChat has a $t$-latency of $2^{t+1}$ for every $t$. We also show that CabChat has a $t$-latency of $(t-k+2)2^{k+1}$ if the $t$ messages with smallest priority value span $k$ distinct priorities. We use these $t$-latency results to derive the main performance theorems summarized above. In the following we annotate the results as they are presented in an attempt to motivate their necessity for establishing our main theorems.

To simplify notation, we assume that the lemmas, corollaries, and theorems that follow are all proved with respect to a fixed execution of CabChat on a fixed dynamic graph.

### *Proof Details.*

Given a queue $Q$, we say a value is sampled in an interval, if during that interval the SAMPLE($Q$) operation returned that value. In the following lemma we prove a useful consequence of the way the queues are sampled.

**Lemma 4.1** (Round Robin)**.** *Fix a queue $Q$ and a time interval $[t_1, t_2]$. The queue can be expressed as the concatenation of queues $F$ and $T$ ($Q = F \cdot T$), such that every element in $F$ was not sampled during the interval $[t_1, t_2]$ and every element in $T$ was sampled during the interval $[t_1, t_2]$.*

*Proof.* Let $S$ be the set of operations performed on $Q$ during the interval $[t_1, t_2]$. We proceed by induction on $|S|$.

**Base Case:** If $|S| = 0$ then no operations were performed during the interval, and the statement holds with the partition $F = Q$ and $T = \varnothing$.

**Inductive Step:** Let $S = S' \cup \{op\}$ where $op \in \{\text{PUSH}, \text{SAMPLE}\}$. By our inductive hypothesis, after applying the operations in $S'$ the statement holds for some partition $F'$ and $T'$. If $op = \text{PUSH}(Q, m)$ then after applying $op$ the statement holds for $F = F' \cup \{m\}$ and $T = T'$. If $op = \text{SAMPLE}(Q)$ then we consider two cases depending on $F'$. If $|F'| = 0$ then the after applying $op$ the statement holds for $F = F'$ and $T = T'$. Otherwise, we let $F' = \{m\} \cup F$, and after applying $op$ the statement holds with $F$ and $T = T' \cup m$. □

The following straightforward corollary follows directly.

**Corollary 1.** *Fix a queue $Q$ and a round interval $[r, r']$. If by the end of the interval the message at the front of $Q$ has been sampled previously in the interval, then all messages in the queue $Q$ were sampled in the interval $[r, r']$.*

We say an execution is *stable* for process $u$ at a round $r$, if at and after round $r$ the incoming set at $u$ is empty. It is easy to show that, as a consequence of fact 1 and Corollary 1, in an execution which is stable for process $u$ at round $r$, CabChat has a $t$-latency of $2^t$. Unfortunately, this is not the case for general executions. To see why, consider an interval of the schedule of length $2^t$ where element $t$ is scheduled at the end of the interval (the fact that such intervals exist follows from fact 1). Assume process $u$ starts at round $r$ with $t$ (or more) messages, each with different priority, where the smallest $t$ messages have priority at most $p$. To satisfy a $t$-latency of $T$ process $u$ needs to send $t$ messages of priority at most $p$ by round $r + T$. Let the algorithm run for $2^t - 1$ rounds without receiving or generating any new messages. At the end of round $2^t - 1$ process $u$ has sent $t - 1$ messages with priority $p$ or less. Now generate an incoming message whose priority is such that it is inserted at position $t - 1$, thereby displacing the queue which was at position $t - 1$ (and had already been sampled) to position $t$. At round $2^t$ the algorithm will sample the queue at position $t$ and no new message will be sent. This already proves that $T > 2^t$. In fact, it is possible to extend this execution to show that $T \geq 3 \cdot 2^{t-1} - 1$.

The question remains of exactly *how much* incoming messages can hurt the $t$-latency of our algorithm. In the rest of this section we answer this question by showing that CabChat has a $t$-latency of at most $2^{t+1}$. In other words, the impact of instability in the queues is a factor of 2 in the $t$-latency. Before proving this we need some intermediate results. Informally speaking, the next lemma shows that if a queue is scheduled to be sampled in the future, either it is sampled or a newly inserted queue will be sampled "soon" instead.

**Lemma 4.2.** *Fix a round $r$, let $i = schedule(r')$ for some $r' \geq r$, and let $Q$ be the queue that occupies the $i^{th}$ position at round $r$. Then either queue $Q$ is sampled at round $r'$, or a queue $Q'$, such that $\rho(Q') < \rho(Q)$ and $Q'$ did not exist at round $r$, is sampled in the interval $[r', r' + 2^{i-1}]$.*

*Proof.* Fix a round $r$, let $i = schedule(r')$ for some $r' \geq r$, and let $Q$ be the queue that occupies the $i^{\text{th}}$ position at round $r$. We proceed by induction on $i$.

**Base case:** If $i = 1$ then at round $r'$ the first queue is sampled, and it will contain either the same queue that existed at round $r$ (queue $Q$) or a newly inserted queue with better priority.

**Inductive step:** If at round $r'$ either queue $Q$ or a newly inserted queue that occupies the $i^{\text{th}}$ position is sampled, the statement holds. Hence suppose a new queue $Q'$ was inserted at the $j^{\text{th}}$ position, where $j < i$ (and hence $\rho(Q') < \rho(Q)$). Fact 2 implies there exists some round $r''$ where $j = schedule(r'')$ and $r' < r'' \leq r' + 2^{i-2}$.

Finally the inductive hypothesis implies queue $Q'$, or a newly inserted queue, is sampled by round $r'' + 2^{j-1} \leq r'' + 2^{i-2} \leq r' + 2^{i-2} + 2^{i-2} \leq r' + 2^{i-1}$. □

In some ways, the following lemma is the complementary opposite of the previous one (and the proofs are in fact, very similar). Informally it shows that if element $i$ is scheduled to be sampled at round $r$, then all smaller elements (that have been in the execution for the relevant period) are sampled "soon" before round $r$.

**Lemma 4.3.** *Fix a round $r$, let $i = schedule(r)$ and let $Q$ be a queue that exists since round $r - 2^{i-1}$ and occupies the $j^{th}$ position at round $r$ where $j < i$. Then queue $Q$ is sampled in the interval $[r - 2^{i-1}, r]$.*

*Proof.* Fix a round $r$, let $i = schedule(r)$ and let $Q$ be a queue that exists since round $r - 2^{i-1}$ and occupies the $j^{\text{th}}$ position at round $r$ where $j < i$. We proceed by induction on $j$.

**Base case:** Let $i = 1$, since the first queue is sampled once every two rounds, it was not sampled at round $r$ (because $schedule(r) > 1$), and it existed the round $r - 1$, it has

to be that it was sampled at round $r-1$ and the statement holds.

**Inductive step:** Fact 2 implies there exists some round $r'$ where $j = schedule(r')$ and $r - 2^{i-2} \leq r' < r$. If at round $r'$ queue $Q$ still occupies the $j^{\text{th}}$ position the statement holds. Hence suppose at round $r'$ queue $Q$ occupies the $k^{\text{th}}$ position where $k < j$.

Finally the inductive hypothesis implies queue $Q$ is sampled in the interval $[r' - 2^{j-1}, r']$, and since $j \leq i-1$ this implies queue $Q$ is sampled in the interval $[r - 2^{i-2} - 2^{i-2}, r] = [r - 2^{i-1}, r]$. □

Equipped with the previous two lemmas, we are now ready to bound the $t$-latency of CabChat.

**Theorem 4.4.** *For every $t \in \mathbb{N}$,* CabChat *has a $t$-latency of $2^{t+1}$.*

*Proof.* Assume process $u$ knows $t$ messages at round $r$, and the smallest $t$ messages have priority at most $p$. We proceed by induction on $t$.

**Base case:** Suppose at round $r$ process $u$ has no messages ($t = 0$), then the $t$-latency statement is vacuously true.

**Inductive step:** By the inductive hypothesis, by round $r + 2^t$ process $u$ has sent $t-1$ messages with priority at most $p$. Let $Q_i$ be the first queue at round $r + 2^t$ such that $\rho(Q_i) \leq p$ and there is an unsampled message at the front of queue $Q_i$. We proceed by cases depending on $Q_i$.

- If $Q_i$ is undefined, then all queues with priority at most $p$ have an element at the front of the queue which has already been sampled by $u$. Therefore, by Corollary 1 all messages with priority at most $p$ have already been sent by $u$, and since by assumption there are at least $t$ such messages the theorem follows.
- If $i > t$ then at least $i - 1 \geq t$ messages of priority at most $p$ have already been sent by $u$, and the theorem holds.
- If $i \leq t$ then at round $r + 2^t$ each of the first $i-1$ queues contain a single sampled message of priority at most $p$. By fact 1 the $i^{\text{th}}$ element of the list was sampled at some round $r' \leq r + 2^i$ but no message of priority less than $p$ was sent. Therefore, it follows that at round $r'$ the first $i - 1 < t$ queues contained $t$ messages of priority at most $p$, and hence at least one queue $j < i$ contained more than two messages. If $r' > r + 2^{i-1}$ then Lemma 4.3 would imply the queue $j$ with two messages gets sampled between $[r, r']$, which in turn would contradict that $i \leq t$. Therefore, it must be that $r' \leq r + 2^{i-1}$, let $r'' = r' + 2^i \leq r + 2^i + 2^{i-1}$. However, in this case by Fact 1 we know that for round $r'' > r'$ $i = schedule(r'')$ and by Lemma 4.2 either queue $Q_i$ gets sampled at round $r''$ or a newly inserted queue $Q'$ with better priority gets sampled before round $r'' + 2^{i-1} \leq r + 2^i + 2^{i-1} + 2^{i-1} = r + 2^{i+1} \leq r + 2^{t+1}$ and the theorem holds.

□

Notice that the $t$-latency result shown by Theorem 4.4 considers the worst case where the $t$ messages of lowest priority value span $t$ distinct priorities. However, one would expect that if the $t$ best messages span a small number of priorities (say a constant), the time required to disseminate them should be linear in the number of messages. This intuition is captured formally by the next theorem.

**Theorem 4.5.** *Fix positive integers $t$, $k$, $r$ and process $u$ If at every round in the interval $[r, r + (t - k + 2)2^{k+1}]$ the $t$ messages with smallest priority value known by process $u$ span at most $k$ distinct priorities, then* CabChat *has $t$-latency of $(t - k + 2)2^{k+1}$ for process $u$ at round $r$.*

*Proof.* Fix positive integers $t$, $k$, $r$ and process $u$, and assume at every round in the interval $[r, r + (t - k + 2)2^{k+1}]$ the $t$ smallest messages known by process $u$ span at most $k$ distinct priorities. We proceed by induction on $k$.

**Base case:** If $k = 0$ then it must be that $t = 0$, and the $t$-latency statement is vacuously true.

**Inductive step:** Let $t'$ be the largest integer such that: 1. Node $u$ knows at least $t'$ messages of priority at most $p$ at the beginning of round $r$. 2. Throughout the interval $[r, r + t2^{k-1}]$ the $t'$ smallest messages known by process $u$ span at most $k-1$ distinct priorities.

If $t' \geq t$ then by inductive hypothesis by round $r + (t - k + 3)2^k < r + (t - k + 2)2^{k+1}$ process $u$ sends at least $t' \geq t$ messages with priority at most $p$, and the theorem holds. If $t' < t$, then by inductive hypothesis by round $r' \leq r + (t - k + 3)2^k$ process $u$ sends $t'$ messages with priority at most $p$. By assumption, the remaining $t - t'$ unsent messages will remain in the first $k$ queues of the list until round $r + (t - k + 2)2^{k+1}$. Moreover, from Lemma 4.1 all the unsent messages are always at the top of the queues.

Therefore, by Fact 1 after an additional $(t - t')2^k$ rounds, the $t - t'$ missing messages would have been sent. Also observe that $t' \geq k - 1$, and hence $t - t' \leq t - k + 1$. Therefore, by round $r' + (t - k + 1)2^k = r + (t - k + 3)2^k + (t - k + 1)2^k \leq r + (t - k + 2)2^{k+1}$ the missing messages have been sent and the theorem holds. □

Observe that Theorem 4.5 gives tighter results than Theorem 4.4 when the number of priorities, $k$, spanned by the $t$ smallest messages, is less than $t - 1$. For the important case where a process has $t$ messages of the highest possible priority (say, accident alerts), CabChat sends all $t$ in only $4(t + 1)$ rounds.

### *Main Performance Theorems.*

We now state our main results, which bound the number of messages, and their priority, disseminated along $\ell$-stable transient paths. Informally, if there exists an $\ell$-stable transient path from $u$ to $v$, and $\ell$ is large enough (say $\ell \geq 2^{t+1}$), then, assuming that $u$ has $t$ messages of priority at most $p$, Theorem 4.4 can be applied inductively along the path to show that $v$ eventually receives $t$ messages of priority at most $p$. On the other hand, if $\ell$ is not large enough, the largest number of messages that can be successfully delivered to $v$ is logarithmic in $\ell$. In more detail: $\log \ell - 1$, is the largest value of $t$ for which $\ell \geq 2^{t+1}$, as required by Theorem 4.4. This is captured by the following theorem statement:

**Theorem 4.6.** *Suppose at round $r$ there exists an $\ell$-stable transient path from $u$ to $v$ of duration $d$. Fix positive integers $t$ and $p$ and assume that at round $r$ process $u$ knows $t$ messages with priority at most $p$. Then at least $\min(\log(\ell) - 1, t)$ messages of priority at most $p$ reach $v$ by round $r + d$.*

For the special case when the $t$ highest priority messages span only a constant number of priorities, this result can be improved to $\Omega(\min(\ell, t))$ by using Theorem 4.5 instead of Theorem 4.4. The intuition is similar to the previous case: consider an $\ell$-stable transient path from $u$ to $v$, where every node in this path has its $t$ best messages (with priority at

most $p$) span no more than $k$ distinct priorities. If $\ell$ is large enough (say $\ell \geq (t-k+2)2^{k+1}$) and $u$ has $t$ messages of priority at most $p$, then Theorem 4.5 can be applied inductively along the path to show that $v$ eventually receives $t$ messages of priority at most $p$. On the other hand, if $\ell$ is not large enough, we can leverage the assumption that $k$ is constant to show that the number of messages of priority at most $p$ delivered to $v$ is linear in $\ell$.

**Theorem 4.7.** *Suppose at round $r$ there exists an $\ell$-stable transient path from $u$ to $v$ of duration $d$. Fix positive integers $t$ and $p$ and assume that at round $r$ process $u$ knows $t$ messages with priority at most $p$. Furthermore, assume that throughout the interval $[r, r+d]$ no process in the path has its $t$ smallest messages of priority at most $p$ span more than $k$ distinct priorities for some constant $k$. Then at least $\Omega(\min(\ell, t))$ messages of priority at most $p$ reach $v$ by round $r+d$.*

It is possible to state a stronger version of Theorem 4.7 (at the expense of a more longer theorem statement) that applies the restriction on priorities at each hop only to the rounds where that hop is involved in the transient path. We omit this extension for the sake of clarity.

# 5. CONCLUSIONS

In this paper, we advocate the use of real world mobility traces to identify properties for the dynamic graph model. We validate this approach by identifying the $\ell$-stable transient path property from the study of real vehicular traces. We then presented CabChat, a prioritized gossip algorithm with strong performance guarantees in the dynamic graph model when the identified property holds. Much interesting work remains in this vehicular model. We note, for example, that a *proper* prioritized gossip protocol should maintain the invariant that for all $t$, its $t$-latency is less than or equal to its $t+1$-latency. With this restriction established, we can prove that the $t$-latency bound of $2^{t+1}$ for CabChat is within a factor of 2 of the optimal solution for proper protocols. Beyond the study of prioritized gossip, other problems are interesting in this setting. For example, disseminating information to a small number of highly connected processes (representing, perhaps, vehicles with Internet access), or learning information about nearby geographic locations. Beyond vehicular networks, our general approach to connecting theory and practice is applicable to any mobile network setting in which real mobility traces are available. As mobile computing becomes more prevalent—e.g., with the growth in smart phone usage—such data sets will become increasingly common.

# 6. ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers of this paper for their constructive criticism, as well as Nancy Lynch for her insightful comments on multiple drafts of this paper.

# 7. REFERENCES

[1] K. Atanassov. On the 37th and the 38th Smarandache Problems. *Notes on Number Theory and Discrete Mathematics*, 5:83–85, 1999.

[2] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized Gossip Algorithms. *IEEE Transactions on Information Theory*, 52(6):2508–2530, 2006.

[3] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott. Pocket Switched Networks: Real-World Mobility and Its Consequences for Opportunistic Forwarding. Technical report, University of Cambridge, Computer Lab, Technical Report UCAM-CL-TR-617, 2005.

[4] Stefan Dietzel, Boto Bako, Elmar Schoch, and Frank Kargl. A Fuzzy Logic Based Approach for Structure-Free Aggregation in Vehicular Ad-Hoc Networks. In *Proceedings of the International Workshop on Vehicular Internetworking*, 2009.

[5] S.M. Hedetniemi, S.T. Hedetniemi, and A.L. Liestman. A Survey of Gossiping and Broadcasting in Communication Networks. *Networks*, 18(4):319–349, 1988.

[6] Bret Hull, Vladimir Bychkovsky, Yang Zhang, Kevin Chen, Michel Goraczko, Allen Miu, Eugene Shih, Hari Balakrishnan, and Samuel Madden. CarTel: a Distributed Mobile Sensor Computing System. In *Proceedings of the International Conference on Embedded Networked Sensor Systems*, 2006.

[7] R. Ingram, T. Radeva, P. Shields, J.E. Walter, and J.L. Welch. An Asynchronous Leader Election Algorithm for Dynamic Networks without Perfect Clocks. In *Proceedings of the International Symposium on Parallel and Distributed Processing*, 2009.

[8] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-Based Aggregation in Large Dynamic Networks. *ACM Transactions on Computer Systems*, 23(3):219–252, 2005.

[9] D. Kempe and J. Kleinberg. Protocols and Impossibility Results for Gossip-Based Communication Mechanisms. In *Proceedings of the Symposium on the Foundations of Computer Science*, 2002.

[10] D. Kempe, J. Kleinberg, and A. Kumar. Connectivity and Inference Problems for Temporal Networks. In *Proceedings of the Symposium on Theory of Computing*, 2000.

[11] M. Kraitchik. *Mathematical Recreations*, chapter The Tower of Hanoi, pages 91–93. W. W. Norton, 1942.

[12] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. Distributed Computation in Dynamic Networks. In *Proceedings of the Symposium on Theory of Computing*, 2010.

[13] B. Liu, B. Khorashadi, D. Ghosal, C.N. Chuah, and M. Zhang. Assessing the VANETâĂŹs Local Information Storage Capability under Different Traffic Mobility. In *Proceedings of the Conference on Computer Commnications*, 2010.

[14] N. Malpani, J. L. Welch, and N. Vaidya. Leader Election Algorithms for Mobile Ad Hoc Networks. In *Proceedings of the International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 2000.

[15] Calvin Newport, Lenin Ravindranath, Hari Balakrishnan, and Samuel Madden. Properties of Vehicle-to-Vehicle Communication. Unpublished Manuscript, 2009.

[16] N. Sarafijanovic-Djukic, M. Pidrkowski, and M. Grossglauser. Island Hopping: Efficient Mobility-Assisted Forwarding in Partitioned Networks.

In *Proceedings of the Conference on Sensor, Mesh, and Ad Hoc Communications and Networks*, 2006.

[17] C. Scheideler. Models and Techniques for Communication in Dynamic Networks. In *Proceedings of the International Symposium on Theoretical Aspects of Computer Science*, 2002.

[18] P. H Schoute. De Ringen van Brahma. *Eigen Haard*, 22:274–276, 1884.

[19] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. VTrack: Accurate, Energy-Aware Road Traffic Delay Estimation Using Mobile Phones. In *Proceedings of the International Conference on Embedded Networked Sensor Systems*, 2009.

[20] J Walter, J Welch, and N. Vaidya. A Mutual Exclusion Algorithm for Ad Hoc Mobile Networks. *Wireless Networks*, 7(6):585–600, 2001.

[21] J.E. Walter, G. Cao, and M. Mohanty. A k-Mutual Exclusion Algorithm for Wireless Ad Hoc Networks. In *Proceedings of the International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 2001.

[22] Bo Yu, Jiayu Gong, and Cheng-Zhong Xu. Catch-Up: a Data Aggregation scheme for VANETs. In *Proceedings of the International Workshop on Vehicular Internetworking*, 2008.