Prioritized Gossip in Vehicular Networks

Alejandro Cornejo[a], Calvin Newport[a], Subha Gollakota[a], Jayanthi Rao[b], T.J. Giuli[b]

[a] *Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Laboratory, 32 Vassar St, Cambridge MA 02319*
[b] *Research and Advanced Engineering, Ford Motor Company, Dearborn, MI, USA*

Abstract

We propose using real world mobility traces to identify tractable theoretical models for the study of distributed algorithms in mobile networks. Specifically, we derive a vehicular ad hoc network model from a large corpus of position data generated by San Francisco-area taxicabs. Unlike previous work, our model does not assume global connectivity or eventual stability. Instead, we assume only that some subset of processes might be 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. 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.

Keywords: Radio Networks, Gossip, Vehicular Networking

Email addresses: acornejo@csail.mit.edu (Alejandro Cornejo), cnewport@csail.mit.edu (Calvin Newport), subha@mit.edu (Subha Gollakota), jrao1@ford.com (Jayanthi Rao), tgiuli@ford.com (T.J. Giuli)

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. For example, Liu et al. [17] use traces of San Francisco-area taxicabs to study the performance of their *VMesh* strategy for local information storage, and Sarafijanovic-Djukic et al. [21] 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*. By contrast in this paper we propose using traces to derive models suitable for generating *theoretical results*.

Restricted Dynamic Graphs. Specifically, we begin with the *dynamic graph* model of [16], 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 use 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, 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 a particular 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 specific 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.

Data-Derived Dynamic Graph Properties. 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 in San Francisco-area taxicabs [1]. We study

networks of 100 and 200 vehicles. We next examine the properties of the connectivity graphs induced by these networks, first showing that there is never global connectivity. For example, in our 200 vehicle networks the largest connected component observed in any round contained no more than 65 vehicles. What we instead observe is a large amount of *transient connectivity* between vehicles—e.g., paths over time—with half of the vehicles in our 200 vehicle networks having transient connections to at least 150 other vehicles. We also observe moderately stable pairwise links, with 25% of links lasting at least 10 seconds and 10% lasting at least 30. We combine these observations of transient connectivity and pairwise link stability into what we call the *ℓ-stable transient path* property, which describes a transient path between two vehicles such that each hop in the path exists for at least ℓ 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 ℓ-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 [8, 27], 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 would have higher priority than an observation of a traffic jam. We refer to this problem as *prioritized gossip*, and we study it in the context of the dynamic graph model with ℓ-stable transient paths.

A challenge for gossip in our setting is the lack of strong connectivity assumptions. In contrast to previous work [16], 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 can only expect that *some* pairs may be connected by an ℓ-stable transient path, for varying ℓ values. This leaves the designer of prioritized gossip algorithms

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

the task of proving their algorithms leverage such paths, when and if they arise, to deliver as much high priority information as possible. Such results are weaker than those guaranteeing global termination, but because they make no 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 ℓ-stable transient path, ℓ different messages are delivered (given a sufficient number of messages existing in the system). 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 ℓ-stable transient path to v, and a message queue of size much larger than ℓ. The round robin approach might lead u to deliver ℓ low priority messages during the ℓ rounds it participates in the path. A good prioritized gossip algorithm, 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 ℓ-stable transient path. Our main performance theorems, summarized below, will leverage t-latency results proved with respect to our algorithm, CabChat, to lower bound the amount of high priority information the algorithm guarantees to be sent over a given ℓ-stable transient path.

Our Results. In particular we consider the prioritized gossip problem with exponentially distributed priorities (i.e., messages with priority 1 are twice as important as messages with priority 2, which are twice as important as messages with priority 3, and so on). We propose the distributed algorithm CabChat that leverages properties of a slight variation of the *binary carry sequence* [3], which also describes an optimal solution the classic *Towers of Hanoi* problem [23, 15].[2] To aid the proof of our main performance theorems,

[2]This versatile sequence has also been used to identify *Hamiltonian paths* in hypercube

we start by bounding the algorithm's t-latency. In the general case, we show that the CabChat algorithm guarantees a t-latency of $\frac{3}{2}2^t$. Moreover, for the important case where the t highest priority values span only $k < t$ distinct priorities, the CabChat algorithm 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 have higher priority), and there is an ℓ-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 ℓ 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 (1) 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 priority), then $\Omega(\min(\ell, t))$ messages of this priority eventually reach v by r'. Notice, because ℓ messages is the maximum number that can be communicated over an ℓ-stable transient path, this second result indicates that asymptotically 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.

In some sense, these results can be seen in terms of throughput guarantees. The first result guarantees that as the amount of available bandwidth between two processes increases, so does the throughput of their communication. The throughput growth, however, lags behind bandwidth growth, guaranteeing $\log \ell$ messages are delivered given the potential to have delivered ℓ. The advantage we gain from this slower growth is captured in the second result, which guarantees that high priority messages take advantage of a constant fraction of available bandwidth. We argue that these results represent a reasonable trade-off between these two competing claims on available bandwidth: the need to increase overall throughput and the need to to increase the speed at which high priority messages spread.

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 [22] for a good overview), in the last

graphs and generate *binary reflected codes*, also known as Gray codes.)

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., [10, 24, 25, 18]—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 [16] 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 [16], 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 San Francisco-area taxicabs, the global connectivity assumption of [16] never holds.)

The gossip problem, of course, has been studied in numerous models (see [9] for a survey of classical results and [12, 11, 4] for a sampling of more recent work). The results most relevant to ours come from the aforementioned study by Kuhn et al. [16], 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.

The notion of a transient path is not our own. In the theory context, Orda and Rom [19], for example, studied *time-dependent* paths in graphs with edge weights that change over time according to an arbitrary weight function. This model generalized many previous models that placed constraints on the weight function. More recently, Xuan et al. [26] studied related path problems in this dynamic setting, such as finding paths with small hop counts, near-future arrival dates, or small transit times.

Our notion of a transient path can be seen as a special case of these existing definitions; i.e., in which the edge weights for a given time are either 1 (i.e., a link exists at this time) or ∞ (i.e., no link exists). There are, however, two differences between our study of such paths and these past theoretical

studies. First, we introduce the property of path *stability*, which captures the potential bandwidth of a transient path. In more detail, we measure a transient path's stability by the minimum existence duration over all its links. To the best of our knowledge, this stability metric, which depends on the duration of each link's existence in the path, and not just their edge weights, has not yet been studied in the context of dynamic graph models.

The second difference between our work and previous work is that previous work assumes full knowledge of the graph. That is, these papers study centralized algorithms that are told how a graph will evolve in the future. A major novelty of our work is that our solutions are distributed and assume no future knowledge of the graphs evolution.

In a more practical context, identifying transient paths is a primary concern in the study of *delay tolerant networks*. To name one example among many, consider the UMASS DieselNet project [28], in which buses driving routes on the UMASS Amherst campus are equipped with local wireless networking equipment. In this setting, connectivity at any moment is limited (restricted to the small number of buses that happen to be near each other at the time), but transient connectivity is much higher. In [28], the authors embarked on a project similar to ours. By studying trace data from their network deployment they identified a generative model that captures the inter-contact time behavior of buses sharing the same route. Using this model they designed protocols that exploit transient paths more efficiently than simple epidemic routing. Chaintreau et al. [5], to name another example, performed a similar experiment in which they derived a power-law based model of inter-contact times from the study of four different real world mobile network traces. They also then used these models to generate protocols that yielded better performance than simple solutions.

Road Map. In Section 2 we analyze the connectivity of real vehicular networks. We use these observations to help define our formal model in Section 3. Then, in Section 4 we define the prioritized gossip problem, present CabChat, our distributed prioritized gossip algorithm, and prove a pair of performance theorems. In Section 5 we perform an experimental evaluation of the protocol to confirm idea that our theoretical results translate well to real deployments. Section 6 concludes 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 San Francisco. Our conclusions are summarized in three informal observations. In Section 3, we will use these observations to both justify and formalize our mobile network model.

Connectivity Traces. The source of our network mobility data is the Cabspotting project, which publishes GPS movement traces of cabs in San Francisco [1]. The specific Cabspotting data set we used for our experiments was retrieved from the CRAWDAD wireless network data archive, and included positions from approximately 500 cabs sampled over a 30 day period [20].

From this data set we extracted 100 mobility traces each containing 100 cabs, and 100 mobility traces each containing 200 cabs. The traces were all 15 minutes in length. We then converted these traces into the waypoint format used by the ns-2 [2] network simulator. This format models each cab moving at uniform speed in a straight line between each successive pair of locations.

To calculate the connectivity properties of these networks we implemented a simple beaconing protocol in which each cab broadcasts a message with its id once every 5 seconds. We simulated this protocol using ns-2 (v.2.3.4), with the `Ext` variants of the physical and MAC layer, which are improved versions of these modules optimized to more accurately describe wireless communication in a vehicular setting (see [7] for more details). We configured the simulator to describe 802.11a, broadcasting messages at its most reliable bitrate and encoding (6 Mbps and BPSK), with the parameters of the physical model configured to match an urban setting (i.e., as described in [13]).[3] Experiments indicate an effective communication range for these settings of approximately 150m (though the fate of a specific packet also depends on both interference and the random fading effects introduced by the physical layer model).

Using the received message logs we then determined the changing connectivity of the network in each simulation. In more detail, for each simulation we split time into 5 second *rounds*. For each round, we used the received message logs to determine which vehicles were connected during this inter-

[3]For the reader interested in replicating our results, we used the 802.11a configuration script included with this release of ns-2.

val, adding a directed edge between vehicles i and j if and only if j received a beacon from i during the round. The result is a dynamic communication graph with a structure that changes from round to round, where each round represents the connectivity of a 5 second interval of the corresponding simulation. We analyzed these communication graphs to produce the results described below.

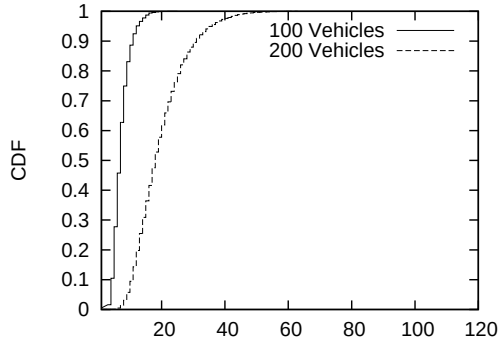

Figure 1: Cumulative distribution function plot of maximum component size (for both 100 vehicle and 200 vehicle communication graphs.).

Global Connectivity. A common assumption in the study of mobile networks is that the communication graph is connected; e.g., [16]. To test this hypothesis we calculated for every communication graph of each network size, and for each round communication round, the largest connected component 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 200 vehicle graphs the maximum component is always of size less than 65, and in the 100 vehicle graphs, the maximum component is always of size less than 25. On the other hand, in both the 100 and 200 vehicle networks, roughly half of the vehicles are in a component that involves at least 10% of the network. Put another way, the network is *never* connected, though small connected components are common, this is summarized below.

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

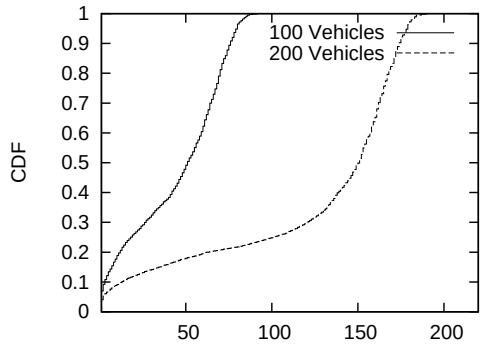

Figure 2: Cumulative distribution function plot of the number of vehicles reached (for both 100 vehicle and 200 vehicle communication graphs.).

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 that are transiently connected to a in the length of the experiment. By definition, it follows that the reach of any given vehicle can only increase when considering a larger time window. As with the rest of our experiments, we considered the time window to be 15 minutes. For each communication 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, for example in the 200 vehicle experiments the median reach was 150 vehicles, while in the 100 vehicle experiments, it was around 50 vehicles. We summarize these results as follows:

Observation #2: Most vehicles are transiently connected to a constant fraction of the network.

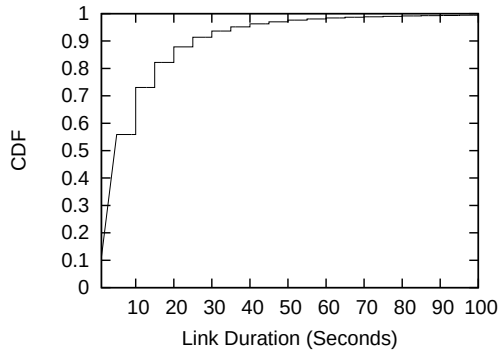

Figure 3: Cumulative distribution function plot of link duration over all links observed in the communication graphs.

Link Stability. We conclude our analysis by looking at the duration (or stability) of pairwise links. In contrast with the previous reach metric, as long as the time window considered is not too small, we do not expect the average duration of a link to change significantly when varying the length of the experiments. Figure 3 shows the cumulative distribution function of the link durations of all links observed in the 200 vehicle networks. (In this CDF, a point $f(x)$ describes the fraction of links with a duration less than or equal to x.) Notice, because we determine whether a given pair of cars is connected only once for every 5 second round, the link durations for which we have data points in our plot are all multiples of 5.

This plot indicates that short links are relatively common—around half of the links lasted for no more than a single 5 second round—but longer lasting links are hardly rare—around 25% of the links last at least 10 seconds, and around 10% lasted for at least 30 seconds. We should note that although contention is minimal in our simulations (due to the low density and broadcast rate), a single missed beacon due to a collision would terminate the corresponding link in our analysis. Therefore, this plot is likely a somewhat pessimistic estimation of link stability in practice. We summarize these results as follows:

Observation #3: Stable links are common.

3. System Model

We consider a synchronous radio broadcast network model where the connectivity between nodes can change, perhaps significantly, from round to round. Specifically, following [16], we assume the communication is described by 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 at round r. (In this paper we assume the natural numbers $\mathbb{N}$ start from 1).

An *algorithm* $\mathcal{A}$ is a collection of $|V|$ *processes*, one for each node in the graph. In a slight abuse of notation, we use *process* u to refer to the process associated with some node $u \in V$. An execution of an algorithm in a dynamic graph proceeds as follows. At every round r each process chooses a message to broadcast (if any). Then every process receives a set (potentially empty) of the messages sent during that round from each 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.

The model, as so far described, allows every node to be in isolation throughout the execution. Therefore before proving any useful results in this model it is necessary to impose some additional properties on the graph. In [16], for example, the authors assume the graph is connected in every round. In this paper, we instead use the observations made in Section 2 to identify a property geared specifically for our vehicular network setting. Specifically, from Observation 1 we know that the graph is not globally connected at any round (or even transiently through time). However, from Observation 2 we know that many nodes do have transient paths between them, and Observation 3 tells us that the individual hops of these transient paths are likely to be relatively stable (e.g., persist for more than a just a second or two). Motivated by these observations we define the concept of an *ℓ-stable transient path*, which is a path that exists over time (but not necessarily all at once) and where each hop is stable for ℓ consecutive rounds.

Notice, however, that in this definition we are making a jump from what we directly observed—transient connectivity and stable links—to a property that we did *not* directly observe—stable transient paths. It would strengthen our case if in the previous section we had shown that this style of paths is common in our real world traces. We omit such an analysis not because

we could not find these paths, but instead because we could not identify a computationally feasible way of searching for them in our data. Finding an efficient algorithm for this search is the focus ongoing work. However, we note that others have shown that temporal structures with a flavor similar to stable transient paths exist in various mobile and social networks datasets [6, 14]. In the meantime, the remainder of this paper rests on the stipulation that the observations of the previous section imply the existence of stable transient paths.

We continue now with our formal definitions:

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 ℓ-*stable*:

Definition 2 (ℓ-Stable Transient Path)**.** Given a transient path described by the round sequence $r_1, \ldots, r_m$ and the edge sequence $e_1, \ldots, e_m$, we say it is ℓ-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 ℓ-stable if each hop exists for ℓ consecutive rounds, and no two hops' round intervals overlap. Naturally, we expect that with larger values of ℓ, ℓ-stable transient paths become more uncommon. We direct the interested reader to the related work discussion of Section 1 for a comparison of our path definitions and those that have been previously studied.

4. Prioritized Gossip

The prioritized gossip problem requires processes to disseminate messages of various priorities while giving precedence to messages of higher priority. Without making any strong assumptions on the connectivity between vehicles, the traditional notion of *solving* the prioritized gossip problem by

completing all-to-all message exchange does not apply. Therefore, we instead turn our attention to how well an algorithm takes advantage of the connectivity that happens to exist in an execution to deliver the messages according to their priority.

In more detail, our aim is to prove a lower bound on the number of messages, and their priority, which are guaranteed to be delivered over the ℓ-stable transient paths that exist in a given 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 ℓ-stable path. The t-latency metric captures an algorithm's performance in terms of this goal. Ideally, an algorithm would that guarantees a t-latency of t for all t. This would ensure that given an ℓ-stable path between some u and v, starting at round r, v would be guaranteed to learn the ℓ-highest priority values known to u at r (or ℓ values of higher priority). [4] Unfortunately, it is not hard to see that it is impossible for an algorithm to achieve such a t-latency. 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 for all t a t-latency linear in t (i.e. of $c \cdot t$ for any positive constant c). In this work we consider exponentially distributed priorities, where messages with priority 1 are disseminated twice as much as messages with priority 2, which are disseminated twice as much as messages with priority 3, and so on.

The CabChat algorithm presented in this paper guarantees a t-latency of $\frac{3}{2}2^t$, for all t. For a constant t, the latency is also constant, which ensures that high priority messages are disseminated on ℓ-stable paths without requiring ℓ to be large. For large t, this value is large but bounded, ensuring that as bandwidth increases on a path (i.e., ℓ gets larger), so does the amount of unique messages that will be delivered. We also show that in the special and relevant case where the t highest priority values span a constant number of

[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 ℓ highest priority messages to w'.

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 ℓ-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 4.5). 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 4.6).

4.1. Problem Description

We assume that at the beginning of each round, each process produces a set of messages (perhaps empty) which it wishes 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:

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.

[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.

4.2. CabChat *Algorithm*

In this section we describe CabChat, a deterministic distributed prioritized gossip algorithm. At its core, the CabChat algorithm uses a priority scheduling function based on the *binary carry sequence* [3]. We will describe many useful properties of this sequence, one of them being that every positive integer i appears in every interval of length 2^i. It also guarantees that whenever integer i appears, the vicinity of size 2^{i-2} on either side contains all positive integers $j < i$. These two properties will turn out to 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. We define the priority scheduling function, *schedule*, used by CabChat to be 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 [15, 23]. 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. The following is a well known recursive definition of S_n.

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

A well known folklore result is that the first $2^n - 1$ numbers of the binary carry sequence plus one (i.e. the first 2^{n-1} numbers output by the *schedule* function) are equal to S_n. The following fact of the *schedule* function follows as a direct 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.

Algorithm Description. The detail pseudo-code for the CabChat algorithm appears in Algorithm 1, here we give a high-level overview of its operation. Each process maintains an initially empty linked list where each element in the list has a non-empty message queue. Let Q_i denote the i^{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 $i = \mathit{schedule}(r)$, 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^{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 $m \in \mathit{incoming}$, 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 $r = \mathit{schedule}(i)$ this algorithm does not necessarily broadcast a message with *priority* i. Instead it broadcasts a message with the i^{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.

4.3. Proof Outline

At the conclusion of this section we prove two main performance theorems. The first, Theorem 4.5, says that if a process u knows t messages, and an ℓ-stable transient path exists from u to v, then v will receive u's $\Omega(\mathit{min}(\log(\ell), t))$ highest priority messages (or a collection of messages of the same or better priority) in time proportional to the duration of the path. The second result, Theorem 4.6, 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 $\Omega(\mathit{min}(\ell, t))$.

To prove these results, we first establish some useful invariants that hold as a result of how the queues are manipulated by the algorithm. Aided by these invariants we then prove that regardless of the number of arriving

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
```

messages at every round, and the priority distribution of these messages, CabChat has a t-latency of $\frac{3}{2}2^t$ for every t. We also show that if the t messages with smallest priority value span k distinct priorities, then CabChat has a t-latency of $(t-k+2)2^{k+1}$. Finally, these t-latency results to derive the main performance theorems summarized above.

Proof Details. 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.

We say a value v is sampled from queue Q during an interval, if during that interval the SAMPLE(Q) operation returned v. The following lemma essentially shows that regardless of the interleaving of the SAMPLE and PUSH operation, during any interval a queue always maintains "unsampled" messages at the top of the queue.

Lemma 4.1. *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 the number of such operations $|S|$.

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 $T = T' \cup m$. □

As a straightforward corollary we can show that if during an interval we sample the value of a queue twice, then we have sampled all its values.

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 $[r, r']$.*

Define an execution as *stable* for process u at a round r, if at and after round r the incoming set at u is empty (i.e. starting at r the process at u does not receive or generate any new messages). It is not hard to show that if an execution is stable for process u at round r then CabChat has a t-latency of 2^t. This follows from the fact that fact 1 guarantees that the i^{th} queue is sampled every 2^i rounds, and Corollary 1 implies that no message is sampled unnecessarily. Unfortunately, this is not the case for general (i.e. non-stable) 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 also 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 $f(t)$ process u needs to send t messages of priority at most p by round $r + f(t)$. 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 at round $2^t - 1$ 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

instead of sending a new message, it will resend and old message. In fact, it will take an addition 2^{t-2} rounds (see Fact 2) to sample a new element, which already shows that $f(t) \geq 2^t + 2^{t-2} = \frac{5}{4}2^t$. This "bad" execution can be extended inductively, by forcing resampling of old queues by inserting new messages before each queue is sampled, see Figure 3.

(a) By the end of round 2^{t-1} the $t-1$ highest priority messages have been sent

(b) At round $2^t - 1$ a new queue displaces the t^{th} queue

Figure 4: Execution fragment which shows latency of is greater than 2^t.

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 $\frac{3}{2}2^t$. In other words, the impact of instability in the queues is a factor of 1.5 in the t-latency. Before proving this we need some intermediate results.

Above we argued how to increase the t-latency of an execution by having a queue with new messages Q be displaced by an queue with old messages Q' right before we sample Q. The next lemma shows that if a queue Q is scheduled to be sampled in the future, either Q is sampled or some queue with "better" messages is sampled "soon" instead.

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

Proof. 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 all queues which were inserted between round r and round r' (possibly none) where inserted at some position $j \geq i$, then the statement trivially holds. Hence suppose a new queue Q' was inserted at the j^{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 $r'' + 2^{j-1} \leq r'' + 2^{i-2} \leq r' + 2^{i-2} + 2^{i-2} \leq r' + 2^{i-1}$. □

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

Theorem 4.3. *For every $t \in \mathbb{N}$,* CabChat *has a t-latency of $\frac{3}{2}2^t$.*

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: If $t = 0$ then at round r process u has no messages and the statement is vacuously true.

Inductive step: Assume by inductive hypothesis that by round $r+\frac{3}{2}2^{t-1}$ process u has sent $t-1$ messages with priority at most p. After these $t-1$ messages have been sent, let Q_i be the first queue 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. Corollary 1 implies that 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 holds.
- 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 $i < t$ then by round $r+\frac{3}{2}2^{t-1}+2^i \leq r+\frac{3}{2}2^{t-1}+2^{t-1} = r+\frac{5}{4}2^t$ the queue at position i will get sampled (by fact 1). Moreover, Lemma 4.2 guarantees that either queue Q_i gets sampled at round $r + \frac{5}{4}2^t < r + \frac{3}{2}2^t$ and the theorem holds, or a queue with better priority (and which had not been sampled before) will be sampled by time $r + \frac{5}{4}2^t + 2^{i-1} \leq r + \frac{5}{4}2^t + 2^{t-2} = r + \frac{3}{2}2^t$, and the theorem holds.
- If $i = t$ we argue that without loss of generality we can assume the following two claims to hold (we prove them afterwards).

  Claim 1. The first $t-1$ queues contain exactly one message per queue, and every queue after Q_i of priority at most p has only unsampled messages.

  Claim 2. The t^{th} queue is not sampled before the $t-1$ messages are sent.

  From these two claims, it follows that after the first $t-1$ messages are sent, and at some round $r' \leq r+2^t$ (by fact 1), the queue at the t^{th} position will be scheduled. Finally, Lemma 4.2 implies that either queue Q_i gets

sampled at round r' or a queue with better priority (and which had not been sampled before) will get sampled by $r'+2^{t-1} \leq r+2^t+2^{t-1} = r+\frac{3}{2}2^t$, and the theorem follows.

□

Proof of Claim 1. First observe that by assumption the message at the front of each of the first $t-1$ queues is already sampled, and Corollary 1 implies that all the messages in these queues are sampled.

Therefore if a single of the first $t-1$ queues contained more than one message, it would imply that t messages of priority at at most p have already been sent. Similarly, if a queue after Q_i contained a sample message of priority at most p, then t messages of priority at most p would have already been sent. □

Proof of Claim 2. If the queue at position t is sampled before the first $t-1$ messages are sent, there are two possibilities.

1) The t element of the list either has no queue or it has a queue of priority greater than p. However, this implies that when the queue at position t was sampled the first $t-1$ queues had at least t messages of priority at most p. Therefore there is at least one queue of priority at most p with at two or more messages. But this would contradict Claim 1.

2) The t element of the list has a queue of priority at most p, let that be queue Q. If $Q = Q_i$ then it contradicts that Q_i has no sampled elements. If $Q \neq Q_i$ then it has moved to a position after t, but that would contradict Claim 1 (i.e. there is a queue after Q_i with priority at most p and with a sampled message). □

Notice that the t-latency result shown by Theorem 4.3 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 4.4. *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 $t' \leq t$ 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 $[r, r + (t' - k + 3)2^k]$ the t' smallest messages known by process u span at most $k - 1$ distinct priorities.

If $t' = t$ 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 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 $r' + (t-k+1)2^k = r + (t-k+3)2^k + (t-k+1)2^k = r + (t-k+2)2^{k+1}$ the missing messages have been sent and the theorem holds. □

When the number of priorities spanned by the t best messages is less than $t-1$ Theorem 4.4 gives tighter results than Theorem 4.3. For example, for the important case where a process has t messages of the highest possible priority (say, accident alerts), Theorem 4.4 guarantees CabChat will broadcast all of them 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 ℓ-stable transient paths. Informally, if there exists an ℓ-stable transient path from u to v, and ℓ is large enough (say $\ell \geq \frac{3}{2}2^t$), then, assuming that u has t messages of priority at most p, Theorem 4.3 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 ℓ is not large enough, the largest number of messages that can be successfully delivered to v is logarithmic in ℓ. In more detail: $\log \ell - 1$, is the largest value of t for which $\ell \geq \frac{3}{2}2^t$, as required by Theorem 4.3. This is captured by the following theorem statement:

Theorem 4.5. *Suppose at round r there exists an ℓ-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 4.4 instead of Theorem 4.3. The intuition is similar to the previous case: consider an ℓ-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 ℓ is large enough (say $\ell \geq (t - k + 2)2^{k+1}$) and u has t messages of priority at most p, then Theorem 4.4 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 ℓ 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 ℓ.

Theorem 4.6. *Suppose at round r there exists an ℓ-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 4.6 (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. Experimental Evaluation

In this section we perform a preliminary experimental evaluation of the CabChat protocol. Our goal is not to exhaustively explore the protocol's performance, but to instead provide support for the idea that our theoretical results will yield desirable results in real deployment.

Simulation Setup. The protocols were evaluated in the Ford Motor Company's research and development lab, using their internal simulator (which

is based on ns-2). The physical and MAC layer parameters were set to simulate 802.11a with link bandwidth of 11Mbps. The propagation model used is two-ray ground configured for a transmission range of roughly 200m. These settings are slightly different than the ones which were used at MIT to perform the connectivity analysis of Section 2.

Experiment Setup. The mobility traces for our simulation experiments were derived from the same San Francisco cab position logs used in Section 2. In more detail, we extracted 20 mobility traces, each 1100 seconds long and containing 250 cabs. Each experiment was repeated on the 20 mobility traces. To evaluate prioritized gossip protocols, we used messages of three priorities in our experiments namely priority 1 (highest priority), priority 2 and priority 3 (lowest priority). In each experiment, all 250 cars start with one message of each priority. We examined different durations for the protocol round length (i.e., time between broadcasts). We settled on a round duration of 10 seconds, which is long enough to minimize congestion, yet short enough to still allow a significant amount of propagation.

We implemented three prioritized gossip protocols to compare in our simulations. The first was the CabChat protocol, which we implemented with the following optimization: whenever the binary carry sequence returned a value larger than the current number of queues, the protocol sampled the highest priority queue. As a point of comparison, we also implemented the *round robin*-variant of the protocol, which behaves like CabChat except that it samples its queues in round robin order (e.g., instead of using the binary carry sequence). We also implemented a *single queue*-variant that disregards priorities, and keeps all messages in a single LIFO queue.

To evaluate the performance of these protocols, for each simulation, we calculated the *reach* of each message m to be the number of cars that have m in their message queues at the end of the simulation. We then examined the distribution of reach values for each of the three different priorities in the system, over all 20 simulations, for all three protocols.

Our goal was to show that CabChat spreads the highest priority messages farther than either the round robin or single queue protocols. This behavior is the expected result of Theorem 4.6 of Section 2, which establishes that CabChat takes advantage of a constant fraction of the available bandwidth when propagating high priority messages. At the same time, however, we also want to show that the lower priorities are not neglected, with their reach diminishing gracefully as the priority lowers. This behavior

is the expected result of Theorem 4.5, which establishes that for messages of lower priorities, throughput still increases along with available available bandwidth, albeit at a rate that decreases along with priority.

5.1. Results

In this section, we present the results obtained from the simulation experiments on real-life mobility traces as described earlier. The round length used for these experiments was 10s. Figure 5 shows the distribution of the reach value of messages when LIFO protocol is used for gossiping. In this case, a single LIFO queue is used to store messages. This leads to cars broadcasting the most recently received message during each round. The following observations can be made from the results. First, 20% of the messages have a reach value of zero. Second, the maximum number of cars that is reached by a message is about 90 (about 36% of cars). A majority of messages reach at most 40 cars (about 16%).

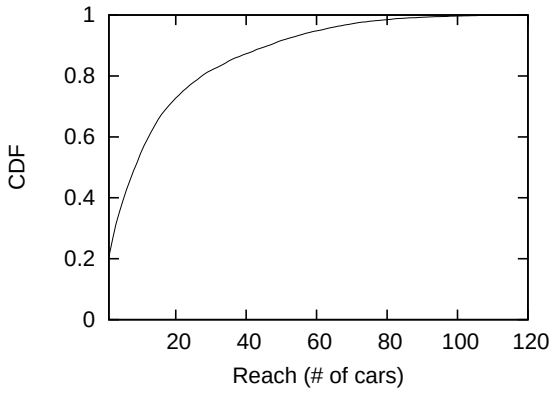

Figure 5: Reach metric for messages in LIFO protocol

The above experiment was repeated for *round robin* version of the protocol and the reach distribution obtained is shown in Figure 6. The reach curves of different priorities overlap with each other implying that *round robin* effectively treats all priorities alike. It can also be observed that the maximum reach of any message is same as LIFO (about 90 cars or 36%). However, the median reach value of messages is slightly higher compared with LIFO. This difference can be attributed to the maintenance of a separate queue per priority.

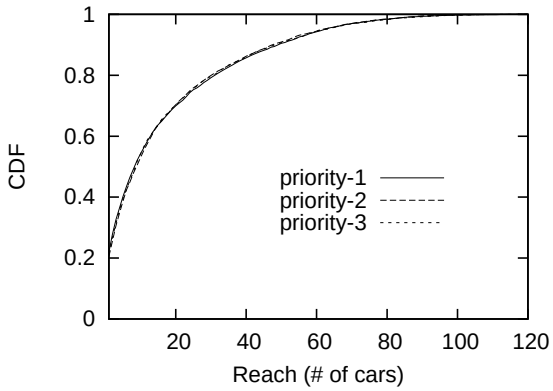

Figure 6: Reach metric for messages in Round-Robin protocol

Figure 7 shows the reach of messages when CabChat is used for gossiping. Clearly, with CabChat higher priority messages have greater reach compared to lower priority messages. The results also show that the fraction of messages that have a reach value of zero decreases with increasing message priority. It is also easy to observe that the reach of priority 1 messages is higher with CabChat than *round robin* or LIFO. Thus, the results validate that CabChat is able to allocate bandwidth to the priorities in such a way that the higher priorities enjoy better reach without starving the lower priority queues.

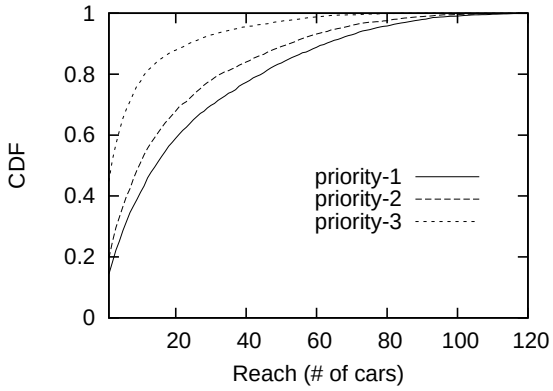

Figure 7: Reach metric for messages in CabChat protocol

To gain insight into the underlying reasons for the observed behavior of

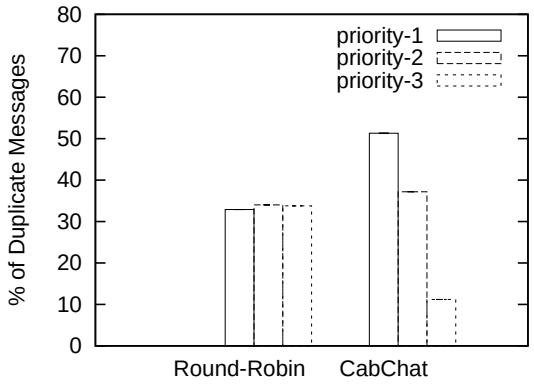

Figure 8: Reach metric for messages in CabChat protocol

the protocols under investigation, we measured the percentage of duplicate messages received by cars and studied their distribution across priorities. Recall that when cars receive messages from their neighbors, they store messages that do not exist in their queues and discard the duplicates. We found that for each of the protocols, around 73% of messages on average were duplicates. Since LIFO does not distinguish between priorities, we plot the priority-wise distribution of duplicates for *round robin* and CabChat in Figure 8 across the 20 runs with a confidence interval of 95%. The measured values do not vary much across simulations, and in fact the error bars were omitted since they were to small to be discernible. *Round robin* incurs approximately the same level of duplication for each of the priorities. However, CabChat has significantly higher level (50%) of duplication for the highest priority in comparison. Duplication gradually decreases with decreasing priority. This indicates that higher duplication (or frequent broadcast) of the higher priority messages results in better reach values as seen in Figure 7.

6. 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 ℓ-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. Finally, we validated our theoretical results through

an experimental evaluation. Much interesting work remains in this vehicular model. For example, to study upper and lower bounds for the t-latency metric for different restrictions on the priorities (in this work we considered only exponential priorities). 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.

7. Acknowledgments

We would like to thank Nancy Lynch for her insightful comments on multiple drafts of this paper, as well as the anonymous reviewers of earlier versions of this work for their constructive criticism. This work was supported in part by AFOSR (Award Number FA9550-08-1-0159), NSF (Award Numbers CNS-0715397, CCF-0726514 and CCF-0939370) and Mobile Mesh Networks (Ford-MIT Alliance Agreement January 2008)

References

[1] Cabspotting Project. http://cabspotting.org.

[2] The Network Simulator ns-2. http://www.isi.edu/nsnam/ns/.

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

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

[5] 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.

[6] A. Chaintreau, A. Mtibaa, L. Massoulie, and C. Diot. The diameter of opportunistic mobile networks. In *Proc. 2007 ACM CoNEXT conference*, page 12, 2007.

[7] Qi Chen, Felix Schmidt-Eisenlohr, Daniel Jiang, Marc Torrent-Moreno, Luca Delgrossi, and Hannes Hartenstein. Overhaul of IEEE 802.11 Modeling and Simulation in ns-2. In *Proceedings of the ACM Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, 2007.

[8] 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.

[9] 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.

[10] 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.

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

[12] 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.

[13] Arijit Khan, Shatrugna Sadhu, and Muralikrishna Yeleswarapu. Technical report, University of California Santa Barbara, http://www.cs.ucsb.edu/ arijitkhan/cs276.pdf.

[14] G. Kossinets, J. Kleinberg, and D. Watts. The structure of information pathways in a social communication network. In *Proc. 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 435–443, 2008.

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

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

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

[18] 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.

[19] A. Orda and R. Rom. Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length. *Journal of the ACM*, 37(3):607–625, 1990.

[20] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. CRAWDAD data set epfl/mobility (v. 2009-02-24). Downloaded from http://crawdad.cs.dartmouth.edu/epfl/mobility.

[21] 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.

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

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

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

[25] 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.

[26] BB Xuan, A. Ferreira, and A. Jarry. Computing Shortest, Fastest, and Foremost Journeys in Dynamic Networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, 2003.

[27] 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.

[28] X. Zhang, J. Kurose, B.N. Levine, D. Towsley, and H. Zhang. Study of a Bus-Based Disruption-Tolerant Network: Mobility Modeling and Impact on Routing. In *Proceedings of the International Conference on Mobile Computing and Networking*, 2007.