Island Hopping: Efficient Mobility-Assisted Forwarding in Partitioned Networks

Natasa Sarafijanovic-Djukic, *Student, IEEE,* Michał Piórkowski, *Student, IEEE,* and Matthias Grossglauser, *Member, IEEE*

***Abstract*— Mobile wireless ad hoc and sensor networks can be permanently partitioned in many interesting scenarios. This implies that instantaneous end-to-end routes do not exist. Nevertheless, when nodes are mobile, it is possible to forward messages to their destinations through mobility.**

In these many interesting settings we observe that spatial node distributions are very heterogeneous and possess concentration points of high node density. The locations of these concentration points and the flow of nodes between them tend to be stable over time. This motivated us to propose a novel mobility model, where nodes move randomly between stable islands of connectivity, where they are likely to encounter other nodes, while connectivity is very limited outside these islands.

Our goal is to exploit such a stable topology of concentration points by developing algorithms that allow nodes to collaborate in order to discover this topology and to use it for efficient mobility forwarding. We achieve this without any external signals to nodes, such as geographic positions or fixed beacons; instead, we rely only on the evolution of the set of neighbors of each node.

We propose an algorithm for this *collaborative graph discovery* problem and show that the inferred topology can greatly improve the efficiency of mobility forwarding. Using the proposed mobility model we show through simulations that our approach achieves end-to-end delays comparable to those of epidemic approaches and requires a significantly lower transmission overhead.

***Index Terms*— Delay-tolerant networks; Store and forward networks; Mobile communication systems; Routing protocols; Simulation and modeling; Mobility modeling.**

I. Introduction

In many applications of wireless ad hoc and sensor networks, the network is frequently or permanently partitioned, i.e., end-to-end routes between some pairs of nodes do not exist. Such scenarios include large-scale emergency and military deployments without fallback infrastructures, environmental monitoring [1], transportation networks [2], self-organized "pocket-switched" networks [3]. These networks may be partitioned because of subcritical node density, channel fluctuations (shadowing and fading) and node mobility.

Although *instantaneous* end-to-end routes do not always exist a message can nevertheless be delivered over time, where the message has to be temporarily buffered at intermediate nodes to wait for the availability of the next link towards the destination. The problem of finding a route over time amounts to finding a sequence of mobile nodes that physically carry the message towards the destination. This approach has been referred to as *mobility-assisted forwarding*, or also as *store-carry-forward* [4], [5]. Finding such routes through space and time is obviously a complex problem in general and depends heavily on the joint statistics of link availability [6].

In this paper, we are interested in the case where network partitions arise because the distribution of nodes in space is heterogeneous. Specifically, we assume that the network possesses concentration points (CPs), i.e., regions where the node density is much higher than on average, and where nodes have therefore a much better chance of being connected to other nodes than on average. We believe that many real networks possess such concentration points, even though most network models assume homogeneous node distributions for convenience and tractability [7], [8].

Furthermore, we argue that the CPs, and the average flows of nodes between CPs, typically remain stable over relatively long time-scales. This is because they depend on features of the natural or constructed environment, which change over time-scales much longer than the delivery time of messages.

Our goal is to develop for mobility-assisted forwarding efficient schemes that take explicit advantage of the presence of stable concentration points. To achieve this goal, we make three distinct, but strongly related contributions: (i) we introduce a mobility model that explicitly embodies CPs, and we justify it through an analysis of two large mobility traces; (ii) we describe how a collection of mobile nodes can infer the CP topology without any explicit signals from the environment, such as GPS coordinates or beaconing signals; and (iii) we describe the Island Hopping (IH) algorithm that forwards messages through mobility. Finally, we summarize these three contributions.

A. Mobility Model with Stable Concentration Points (CPs)

Our first contribution is a new mobility model that embodies the presence of stable CPs. This model is pessimistic, in the sense that we assume that nodes are only able to communicate with other nodes at the same CP; outside these islands of connectivity, they are not able to communicate. We therefore view the network as a graph $G(V, E)$, where the vertex set V represents the CPs, and the edge set E represents flows of mobile nodes between the CPs.

Note that the only assumption we make about the connectivity of the set of nodes within a same CP is that they form a connected subgraph, i.e., that each node can reach other nodes. If all the nodes are within radio range of each other, then this is straightforward; if not, then a message sent between two nodes at a same CP may have to traverse multiple intermediate hops. Our routing and graph discovery algorithms rely on simple broadcast primitives within a CP. They can be implemented through physical-layer broadcast, or through flooding algorithms.

Results from Large Mobility Traces. We provide some evidence from two distinct large-scale data sets to justify the validity of our mobility model. We analyze a trace of the movements of ~ 800 taxis over a three-month period in the city of Warsaw, Poland and of ~ 600 taxis over a month period in the city of San Francisco, USA. We find that for both agglomerations there are some areas, where the expected number of cars within radio range of each other is much higher than elsewhere. Furthermore, we find that for both agglomerations these areas are stable over time, in the sense that an area that sees a high density of cars on a particular day is likely to see a high density on another day as well. This justifies our assumption of a stable topology of CPs in two realistic scenarios.

B. Collaborative Graph Discovery (COGRAD)

Our second contribution is a distributed algorithm that allows the nodes to collaboratively discover the CP graph, *in the absence of any signal from the environment*, such as GPS coordinates or fixed beacons. This is important, because it would be unrealistic to assume that the graph of CPs and the flows of mobile nodes between CPs is known a-priori. Instead, we assume that the only information that nodes have available is the set of other nodes that they can reach (either directly or over multiple hops), which can be discovered in a straightforward manner (hello messages, flooding, etc.)

In a nutshell, the COGRAD algorithm achieves this in two phases: vertex labeling and edge discovery.

Vertex Labeling. The goal of this phase is to generate a label, i.e., a unique identifier, for each vertex of V, that will remain stable over time, even though nodes move in and out of each vertex. Suppose that at a given time the nodes currently located at the same CP agree on a label for this vertex. Now another node i arrives at this vertex. Node i has not received, from the environment, any explicit clue that it has moved, and the other nodes have not received a clue that they have not moved. However, node i's set of neighbors has changed rather markedly, whereas the other nodes' neighbor set has only seen the addition of i. These nodes can therefore decide jointly that it is likely that node i has moved, and the other nodes have not; node i therefore accepts the label of this vertex.

This process associates labels with vertices. The labels remain stable even though the set of nodes at a vertex changes all the time. Although errors can occur in this process (e.g., if a vertex becomes completely empty for a while), this does not affect the performance of the routing algorithm if this occurs rarely.

Edge Discovery. Once we have associated a label with each vertex, a node discovers the edges of the CP graph as it moves from vertex to vertex. We also let nodes exchange edges they have discovered through a gossip protocol. This ensures that each node learns the entire graph, even though it may only visit part of the graph. Also, this ensures that outdated information (labels that have become invalid after errors) is flushed out.

C. Mobility-Assisted Forwarding in the CP Model

Our third contribution is a novel mobility-assisted forwarding algorithm called Island Hopping (IH). This algorithm explicitly exploits knowledge of the CP graph. It forwards a message through a sequence of CPs to its destination. Each CP represents an opportunity to pass the message to other nodes.

The key question is how to pass a message from one CP to the next CP through nodes whose future movements are random and unpredictable. If the future movements of nodes were known, we could pass the message to a single node that would move in the right direction, i.e., to a CP closer to the destination in $G(V, E)$. However, given that future movements are unpredictable, the IH algorithm makes a small number of copies of a message at each CP, in the hope that at least one copy will move to the intended next CP and the other copies will be discarded. The process repeats at the next CP, until the message reaches its destination. The

key challenges are to (i) not lose the message completely, and (ii) to avoid that an unnecessarily large number of copies are generated.

To summarize, we argue that stable concentration points are prevalent and that they can be exploited for efficient mobility-assisted forwarding. We present evidence to this effect, and a mobility model embodying CPs, in Section III. In Section IV, we describe *collaborative graph discovery (COGRAD)*, a distributed algorithm that infers the CP graph from each node's dynamic neighborhood set. This allows the routing algorithm to operate without any explicit clue to nodes about their location and movement. In Section V, we then describe the routing algorithm for mobility-assisted forwarding in the presence of stable CPs. For simplicity of presentation, we describe this algorithm assuming that the CP graph, and node positions on the graph, are known. In Section VI, we show extensive simulation results on synthetic graphs. We show that our IH algorithm in conjunction with COGRAD results in a scheme that achieves delays of the order of much more aggressive flooding-based schemes and requires a much smaller number of copies of each message. We conclude the paper in Section VII.

In our earlier paper [9], we describe our main ideas of this work, also present here. The novelties in this paper are: (i) we validate our mobility model in another large scale data trace, (ii) we present a new method for inferring CPs from large scale mobility traces by using only information about the nodes' connectivity and (iii) we improve both the COGRAD algorithm and the IH algorithm. The improved algorithms are now more robust to realistic mobility settings where the CPs might be unstable and where the connectivity is not as sharply divided into CPs and non-CPs as in our mobility model.

II. Related Work

Most mobility models give rise to homogeneous node distributions over a two-dimensional area [7]. These models lack an important feature of realistic mobility patterns, the fact that nodes often cluster around preferred areas [10]. Hsu et al. [11] propose a realistic Weighted Way Point (WWP) mobility model that incorporates the fact that destinations are not uniformly selected from the simulation area. The authors designed the WWP model as a time-varying Markov model, fitted to data from a survey on the USC campus. Tang and Baker in [12] analyze mobile traces with 24773 radios in a metropolitan-area network. They observe a significant clustering of radios in some areas, e.g., a financial district; however, they do not attempt to model user mobility. Tuduce and Gross in [13] studied the characteristics of mobile WLAN users and they proposed a framework for extracting mobility characteristics. Although they do not propose a mobility model that captures spatial distribution of CPs, they do however provide a powerful validation methodology of the realistic mobility models. In [14] Kim et al. provide a framework for extracting mobility characteristics from mobile traces of WLAN users. They propose a mobility model that focuses on movements between popular regions so-called *hot-spot regions* - popular destinations at which mobile users spend most of the time.

Our approach refers to CPs as to specific regions that are not necessarily preferred destinations. We envision CPs as places that provide mobile users with a *high opportunity* to encounter other mobiles. Moreover, we also propose a new method for inferring CPs that is based exclusively on the time evolution of the connecitivity of nodes without any additional information about their location.

Routing in partitioned networks has been investigated in two scenarios: (i) when the dynamics of connectivity between nodes is known in advance ([6]), and (ii) when it is unpredictable ([5], [15], [16], [17], [18], [19]). The latter can be further classified as controlled mobility and random mobility. Under controlled mobility [5], node trajectories can be controlled and adapted. Forwarding algorithms for random mobility are usually based on some form of flooding. For example, in epidemic routing (ER) [15], when two nodes meet, they exchange all messages that only one of them has a copy of. In this way, a message is essentially flooded to all nodes, which ensures that it will reach the destination. This flooding can be constrained by taking into account the mobility history of nodes, e.g., patterns of encounters of nodes, but also other parameters, such as the energy left at the node. PROPHET [16] is based on ER, where message exchanges between nodes take into account a probability that a node will meet the destination of a message in the future. This requires the exchange of vectors of estimated probabilities. In DTC [17] a node carrying a message checks periodically whether to transfer the message. The message is transferred if in the set of nodes (possible via multihop) connected to this node there is a node "closer" to the destination. "Closeness" depends on mobility history, system parameters, and future mobility (if available). A mathematical framework for calculating this "closeness" based on utility functions is given in [18]. In MDDV [19], forwarding decisions in a vehicular network are made based on knowledge of the road map, where nodes are equipped with GPS receivers. In the Spray and Wait algorithm by Spyropoulos et

al. [20], the number of transmissions of a message is constrained by letting only a fixed number L of copies of a message to be "sprayed" into the network. Then these copies "wait" until they meet the destination. The authors show that the parameter L controls the tradeoff between end-to-end delay and overhead.

Our approach considers random unpredictable mobility and distinguishes itself in that we explicitly *infer and use spatial information* of node mobility to limit flooding.

III. A Mobility Model for Partitioned Networks

In this section, we introduce a mobility model for partitioned networks. In particular, we focus on one feature that appears to be quite ubiquitous in real mobility processes: *concentration points* (CPs), i.e., regions where mobile nodes have a much higher chance of encountering other nodes than elsewhere.

Examples of CPs include:

- People in urban environments: workplace, restaurants, public transportation (train stations, airports), movie theaters, etc.
- Wildlife monitoring: watering holes, clearings, oases, etc.
- Office buildings: cafeterias, conference rooms, water coolers, hallways, etc.
- Road traffic: intersections, parking lots, gas stations, traffic lights, etc.
- Military: bases, camps, forts, ports, etc.

We argue that the presence and ubiquity of such concentration points has profound implications for the design of efficient means for dealing with and exploiting mobility.

In order to find an evidence for CPs we analyze two large-scale data sets of mobile traces of two taxi fleets in two large cities in Poland and in USA. We also propose and evaluate a method for extracting CPs from fine-grained mobile traces.

A. The Stability of Concentration Points in a Mobility Trace

Here we analyze the mobility traces of taxi cabs from two cities (Warsaw, Poland[1] and San Francisco, USA[2]). The Warsaw data set contains GPS coordinates of 825 taxis collected over 92 days in the Warsaw agglomeration area. The San Francisco data set contains GPS coordinates of 665 taxis collected over 39 days in the Bay area. In both cases each taxi is equipped with a GPS receiver and sends a *location update* (timestamp, identifier, geographical coordinates) to a central server. In the case of the Warsaw data set the updates are infrequent - the median time interval between two consecutive location updates generated by an arbitrary cab is approximately 500 seconds. In the case of the San Francisco data set the updates are more frequent - the median time interval between two consecutive location updates generated by an arbitrary cab is approximately 70 seconds.

[1] http://www.taximpt.com.pl
[2] http://www.yellowcabsf.com

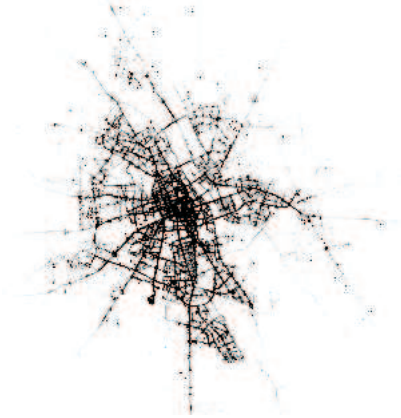


Fig. 1. The spatial distribution of location updates for the Warsaw data set (825 taxis over 92 days). Each cell, represented by a pixel, is coloured according to the $f(k,l)$ averaged over the entire period. The darker the cell (k,l) the higher the normalized population $f(k,l)$.

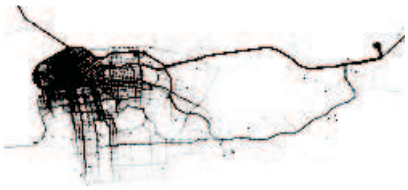


Fig. 2. The spatial distribution of location updates for the San Francisco data set (665 taxis over 39 days). Each cell, represented by a pixel, is coloured according to the $f(k,l)$ averaged over the entire period. The darker the cell (k,l) the higher the normalized population $f(k,l)$.

To confirm the existence of stable CPs we apply the following heuristic. First, we superimpose on the area of Warsaw and of San Francisco agglomerations a grid of cells of equal size. Then, for each day d and each cell (k,l) we find the *normalized population* - $f(k,l;d)$, interpreted as the empirical probability that a random update falls into the cell (k,l) on day d (cf. Figure 1 and 2). Our analysis shows the following:

1) The Spatial Distribution is Heavy Tailed: Figures 3(a) and 3(b) show the empirical complementary cumulative distribution function (CCDF) of $f(k,l;d)$ for the two data sets - from Warsaw and from San Francisco respectively. Both distributions have heavy tails, which implies that some cells in both cities have population density much above the average.

2) The Spatial Distribution is Stable Over Time: Figures 3(a) and 3(b) insets show a scatter plots of $f(k, l; d)$ for one randomly chosen pair of days (d_1, d_2). In both cases we observe significant clustering along the diagonal, which means that the spatial distribution on different days tends to be strongly correlated. Furthermore, we observe that the more densely populated cells (upper-right quadrant) tend to be particularly close to the diagonal, which is a good visual confirmation of our hypothesis. We observe the same behaviour for other pairs of days.

B. Model Based on a CP Graph

Given the above observations, we now define an idealized mobility model that embodies CPs. The network topology is given by a directed connected graph $G(V, E)$ whose vertex set V represents the CPs, and whose edges E describe the possible movements of nodes between CPs. We call this graph a CP graph. There are n nodes that move on this graph. At every time t, every node i is either located at one CP, or is en-route between two CPs u and v. We denote the current position of node i by $X_i(t)$. We assume that time is continuous.

We assume that nodes located at the same CP can communicate with each other (either directly or through multi-hop), whereas nodes at different CPs cannot. We call $B_i(t)$ the set of neighbors of node i at time t, where if node i is at a CP then $B_i(t)$ is the set of nodes located at the same CP as i (including i) and if node i is en-route between two CPs then $B_i(t) = \{i\}$ (cf. Figure 4).

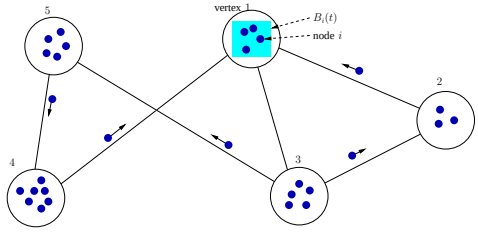


Fig. 4. Nodes move on a graph $G(V, E)$, which describes the network topology in terms of its CPs and the ways nodes can move between them.

C. Inferring the CPs from Mobility Traces

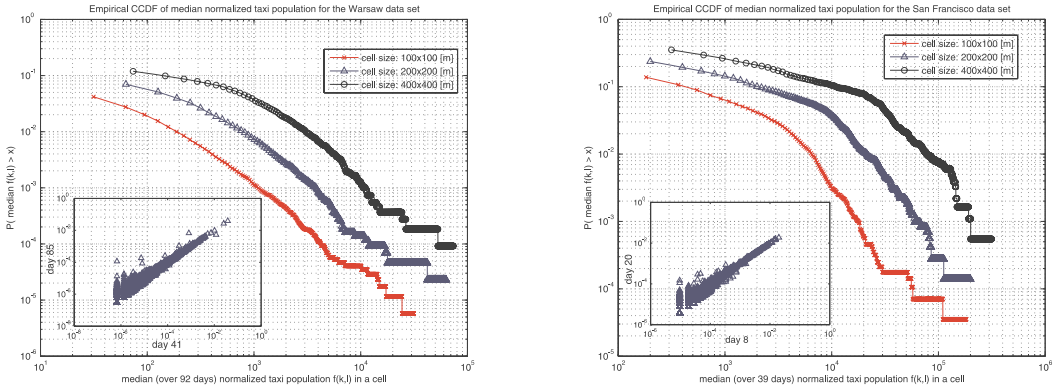
In section III-A we have revealed that in the real world the distribution of mobile nodes is heterogeneus and stable. Here our goal is to infer from mobile traces the stable high-density areas, where the islands of connectivity are likely to appear. The main idea behind our method is to focus on connected components, that represent these islands of connectivity, and to see if they are stable over time. The lesson we learn here is used further in the paper in Section IV-A to derive the distributed algorithm for vertex labeling of the collaborative graph discovery method.

As we explained in Section I, CPs represent regions where node density is higher than average and is stable over time. In order to find such regions one could apply one of the well-known data clustering algorithms, e.g. k-means clustering [21] - for every time instant one can find such regions and then identify which of them last for a long time. However, we take another approach - we identify connected components that last for a long time. Here we rely on an intuition that in highly populated regions nodes should form stable connectivity islands.

Let us first formally define the *connectivity graph* formed by nodes. We assume that a mobile node i is connected to node j if the distance r_{ij} between them is less than the connectivity range r_c. We assume that r_c is fixed and can be interpreted as the maximal radius allowed by power constraints. The mobile nodes and the corresponding wireless links define the *connectivity graph* $H(N, R)$, where $N(H)$ is the vertex set - the set of mobile nodes and $R(H)$ is the edge set - the set of radio links between mobile nodes, i.e., $R(H) = \{e = (i, j) | r_{ij} < r_c\}$. We define a cluster H_n as a connected component of H. We also define a set of all clusters that decompose the connectivity graph as follows: $C(H) = \{H_1, H_2 \cdots, H_m\}$, where $H = \bigcup_{n=0}^{n=m} H_n$.

As nodes are mobile, the connectivity graphs change dynamically over time. Thus, we define a CP as a connected component that lasts long enough in the time-sequence of consecutive connectivity graphs. If we take a snapshot of the connectivity graph at time t it might be very different form the one taken at time $t^- = t - \Delta t$. We can easily identify clusters at every time instant t, but how do we decide which clusters correspond to which clusters from the previous time instance $t^- = t - \Delta t$? To answer this question, we study the differences between sets of clusters defined for consecutive connectivity graphs, i.e., $C(H(t^-))$, $C(H(t))$. In our approach we want that a cluster at time t corresponds only to one cluster at time t^-, i.e. a cluster at time t shares the same identity as a cluster at time t^-. Obviously it is possible that a cluster at time t corresponds to none of the clusters at time t^-, e.g. nodes that were alone at time t^- form a new cluster at time t - then this cluster obtains new identity.

We first show two examples that explain two rules we use to decide which clusters from consecutive time instances correspond to each other, and then we give the algorithm formally. First, assume a scenario shown in Figure 5 and concentrate only on



(a) Warsaw data set (b) San Francisco data set

Fig. 3. Empirical CCDF of $f(k,l;d)$ for the entire period for three levels of discretization for two data sets (a) Warsaw and (b) San Francisco. Insets in (a) and in (b) shows the scatter plot of $f(k,l;d)$ on two random days - each point on the plot corresponds to a density in a cell (k,l) for different days.

cluster $H_1(t^-)$. At time t, nodes from this cluster are distributed over three clusters: $H_1(t)$, $H_2(t)$, and $H_3(t)$. So, which of these three clusters should correspond to cluster $H_1(t^-)$? We take it to be the one that shares the maximum number of nodes with $H_1(t^-)$; in our case it is $H_2(t)$. This is our first rule. More formally we can say:

Rule 1: A cluster $H_n(t^-)$ can pass its label to a cluster $H_m(t)$ if the number of nodes $H_n(t^-)$ shares with $H_m(t)$ is maximum compared to all other clusters at time t.

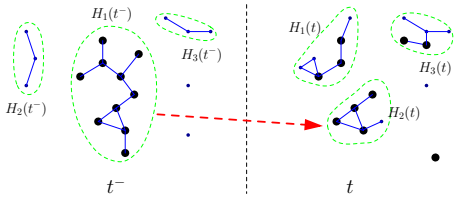


Fig. 5. The first rule used in the CCL algorithm that allows to decide which clusters from two consecutive connectivity graphs correspond to each other. The large black dots represent nodes that were part of cluster H_1 at time t^-.

Next, if we apply our first rule to a scenario shown in Figure 6, then all three clusters at time t^-: $H_1(t^-)$, $H_2(t^-)$, and $H_3(t^-)$ correspond to one cluster $H_1(t)$ at time t. Because we want to have a one-to-one correspondence between clusters, we assume that $H_1(t)$ corresponds to the one cluster at time t^- that shares the maximum number of nodes with $H_1(t)$; in our case it is $H_2(t^-)$. This is our second rule. More formally we can say:

Rule 2: A cluster $H_m(t)$ can inherit the label from a cluster $H_n(t^-)$ if the majority of the nodes from cluster $H_m(t)$ come from cluster $H_n(t^-)$.

Only if the above two rules are satisfied for a pair of clusters from two consecutive time instances then we say that these clusters correspond to each other. We want to track this correspondence of clusters in order to find out how long clusters live. To be

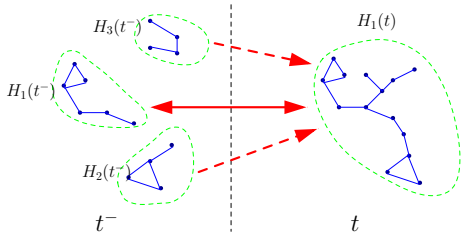


Fig. 6. The second rule used in the CCL algorithm that allows to decide which clusters from two consecutive connectivity graphs correspond to each other. The dashed arrows *connect* those clusters, from time t^- that satisfy the 1st rule, with the H_1 cluster at time t. The solid arrow *connects* cluster from time t^-, which satisfies both rules, with the cluster at time t, i.e., cluster $H_1(t)$ inherits the label from cluster $H_1(t^-)$.

able to do this formally, we introduce the concept of *cluster labeling*. This means assigning an unique identifier y (we call it a label) to every cluster $H_m(t)$, i.e., $y = Y(H_m(t))$, in such a way that the two corresponding clusters at the consecutive time moments have the same label. Hence, the cluster labeling means first giving an initial label to a newly formed cluster (we assign to such a cluster a random label drawn from the very large set of integers), and then finding the correspondence between clusters, which are defined by our two rules explained above. We call our labeling method as the Centralized Cluster Labelling (CCL) algorithm (cf. Algorithm 1). We use the $A(H_m(t))$ to denote a set of applicant clusters from time t^- that (according to our first rule) can pass their label to $H_m(t)$. We also use the $\eta(H_n(t^-), H_m(t)) = |N(H_n(t^-)) \cap N(H_m(t))|$ to denote a function that given a cluster from current time instant and a cluster from the previous instant finds the number of shared nodes.

Algorithm 1: The CCL Algorithm - Input: $H(t^-)$, $Y(H(t^-))$, $H(t)$**; Output:** $Y(H(t))$

1 For each $H_m(t) \in H(t)$
2 /*First rule*/

3 $A(H_m(t)) = \{H_n(t^-) : H_m(t) = \\ = \text{argmax}_{H_o(t) \in H(t)} \eta(H_n(t^-), H_o(t))\}$

4 If $A(H_m(t)) = \emptyset$

5 $Y(H_m(t)) =$ random

6 Else

7 /*Second rule*/

8 $H_n(t^-) = \\ = \text{argmax}_{H_o(t^-) \in A(H_m(t))} \eta(H_o(t^-), H_m(t))$

9 $Y(H_m(t)) = Y(H_n(t^-))$

10 End If

11 $Y(H(t)) = Y(H(t)) \cup Y(H_m(t))$

12 End For

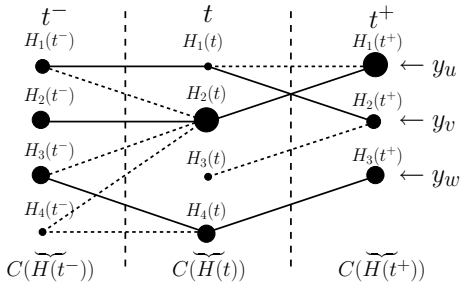


Fig. 7. An example of the execution of the CCL algorithm. The black balls represent clusters. The size of each ball corresponds to the number of nodes that form a cluster. The lines represent migration of nodes between clusters - the solid line means: *the label of $Y(H_m(t^-))$ is passed to cluster $H_n(t)$ since both rules apply*. The dashed line means: *nodes from a cluster $H_m(t^-)$ are part of a cluster $H_n(t)$*. In this example the CCL algorithm finds three distinct labels, i.e., $y_u = Y(H_1(t^-)) = Y(H_1(t)) = Y(H_2(t^+))$, $y_v = Y(H_2(t^-)) = Y(H_2(t)) = Y(H_1(t^+))$ and $y_w = Y(H_3(t^-)) = Y(H_4(t)) = Y(H_3(t^+))$.

In Figure 7, we show in a more abstract way an example of the execution of the CCL algorithm, where the clusters are denoted as balls of different size depending on the clusters' size. We can see here which clusters can pass their label to which clusters (our first rule) and which label is accepted (our second rule). In the example shown on Figure 7 we can identify three CPs, i.e. u, v and w that are labeled with three distinct labels: y_u, y_v and y_w respectively.

In the sequence of corresponding clusters: $\{H_m(t_{birth}), \ldots, H_n(t_{death})\}$, where t_{birth} corresponds to the time when a sequence of corresponding clusters was first observed and t_{death} - to the disappearance of this sequence, all clusters have the same unique label. Thus each sequence of corresponding clusters is uniquely identified by a label. We expect CPs to be stable over time and have a large size, thus we seek labels that last for a long time and are owned by a large (on average) number of nodes. Note that the CCL algorithm does not use nodes' positions - it uses only the connectivity graph, which is one advantage over the data clustering algorithms. Because of that there is no information whatsoever about location of the CPs. However, given the evidence presented in section III-A, we believe that stable clusters should appear at certain fixed locations. We examine this hypothesis in the following subsection.

D. CPs in San Francisco

We use the CCL algorithm to infer CPs from the fine-grained San Francisco trace. In order to construct a time sequence of connectivity graphs we need to know the location of each taxi at every time instant. Unfortunately, the GPS devices installed in cabs are not synchronized. Luckily, the data set is fine-grained enough to allow us to interpolate the position of every taxi given two consecutive location updates. We assume that the vehicle's speed between two consecutive location updates is constant. We assume that if a cab *disappears* (due to the GPS receiver being shut down) for some time and then *re-appears* again in proximity, it did not move between these two consecutive location updates.

Here we specify the metrics we use to analyze sequences of corresponding clusters:

- ρ: median size of a labeled cluster from a sequence of corresponding clusters
- $\tau_s = t_{death} - t_{birth}$: lifetime of a label

In order to extract CPs from collection of connectivity graphs, by applying the CCL algorithm, we use the part of the San Francisco traces - approximately 500 taxis over 24 hour period. The connectivity graphs are generated every $\Delta t = 10$ seconds for connectivity range $r_c = 300$ meters.

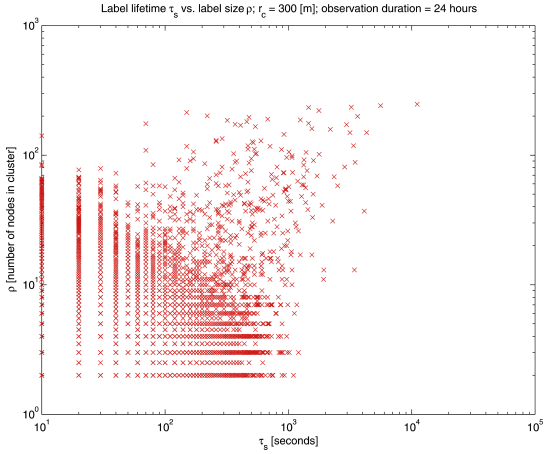


Fig. 8. Correlation between the lifetime of a label τ_s and the median size of a labeled cluster from a sequence of corresponding clusters h in loglog scale.

We use the scatterplot in loglog scale shown in Figure 8 to check how the lifetime of a label is related to its median size. The visual inspection allows us to draw a conclusion that there are at

least two types of labels - (i) those that are highly populated and live a long time and (ii) those that are owned by a small number of nodes and live a short time. The former ones are the best candidates for real CPs. We used two threshold values, one for the lifetime and second for the size of the label: 1800 seconds and 10 nodes respectively, to determine which labels may correspond to CPs. We found 19 such labels. However, these 19 labels do not specify 19 distinct CPs. This is because the same connectivity island may re-appear at different time of the day, which cannot be captured by the CCL algorithm. In order to better understand the characteristics of these stable labels and to find to how many CPs they correspond to, we perform three additional tests. First, we check if a labeled cluster appears in the same area during its lifetime. We superimpose on the map of San Francisco the locations of nodes that own the same label for three different snapshots. We visualize this on Figure 9 where the taxis that own the same label (marked with the same type of a marker) cover the same area at different time instants (different colors of the same marker). This confirms our intuition that islands of connectivity are stable in space as well. Second, we check the location of the nodes that own distinct labels. We identify four such locations - namely the aquatic park/shopping center, the downtown area, the taxi company premises and the airport (cf. Figure 9). Third, we check how many different labels correspond to the same CP. It turns out that the CP located in the downtown area is represented by ten distinct labels, the taxi company premises - by five distinct labels, the airport - by three distinct labels and the aquatic park by one label only. Although the number of CPs for this particular mobility trace is small, the results of our analysis justify the presence of CPs in the real world and our CP graph mobility model.

One should keep in mind that the mobility pattern of taxis is very specific and may not give sufficient evidence of other CPs located for example nearby sport centers, gas stations, movie theaters etc. Thus we believe that given a large set of mobile traces for the same area, which contains GPS location updates of different types of vehicles, it should provide us with more CPs. We also believe that CPs might be much easier observed at a larger scale, e.g. not at the San Francisco agglomeration scale only, but at the whole Bay area scale. Note also that this is the first attempt to identify CPs where a connected component is chosen to give the evidence for the CP existence. The presented results show that this choice might be too extreme for determining if a CP exists or not.

Fig. 9. Four CPs in San Francisco identified by the CCL algorithm. Markers represent taxis that belong to CPs. Different types of markers correspond to different CPs: *triangles* - the aquatic park/shopping center, *balls* - the downtown area, *crosses* - the taxi company premises, *squares* - the airport. The color of each marker represents members of a corresponding CP at different time moments.

IV. Collaborative Graph Discovery (COGRAD)

We now specify how *collaborative graph discovery (COGRAD)* infers the CP graph in a distributed way, from changes in neighborhood sets without any other signal from the environment. Note that a single node would obviously be unable to find out anything about the network topology; a COGRAD protocol is necessarily collaborative. More formally, the goal of COGRAD is:

- for every node to learn the CP graph $G(V, E)$;
- for every node i to know its current position $X_i(t)$ at all times t.

As we have mentioned in Section I, COGRAD achieves this by having the nodes run two algorithms: vertex labeling and edge discovery. The vertex labeling algorithm decides for each node if this node is currently at a vertex (CP) or en-route between two vertices; in the former case, it also identifies the CP through a label. The output of this algorithm is then used as input into the edge discovery algorithm. The edge discovery algorithm estimates the edge set E of the CP graph. We next describe these two algorithms.

A. Vertex Labeling

Every node i maintains a variable $Y_i(t)$, which is either $Y_i(t) = \phi$ if node i thinks that it is en-route between CPs at time t or $Y_i(t) = y$ if node i thinks that it is at the CP with label y. In the former case we say that node i does not have any label, whereas in the latter case we say that node i has label y. The goal of a vertex labeling algorithm is therefore that every node i decides on $Y_i(t)$ at every time t.

We assume that the only information available to node i at time t is:

- $B_i(t)$ - the set of neighbors of node i at time t, i.e., the set of nodes in the connected component (cluster) where node i is;
- $Y_j(t^-)$ for every $j \in B_i(t)$ - the values of labels of all nodes in the cluster where node i is at the time instant before time t (denoted as t^-).

It is impossible that node i knows this information at every continous time t. Instead, we assume that there is a generic neighbor discovery protocol that periodically every Δt gives this information to node i, where Δt is a short period of time. Thus, t^- is in fact $t - \Delta t$.

Now, after we defined which information is available to node i, let us see how node i determines its new label $Y_i(t)$ at time t. If $B_i(t) = B_i(t^-)$ (the neighborhood set of node i did not change) then node i keeps its label (i.e., $Y_i(t) = Y_i(t^-)$). If $B_i(t) \neq B_i(t^-)$ (the neighborhood set of node i changed) then node i recomputes its label. From the data of $Y_j(t^-)$, $j \in B_i(t)$ node i first computes the set of tuples $L_i(t) = \{(y, b^y(t), b^y(t^-))\}$, where $b^y(t)$ is the number of nodes in $B_i(t)$ that had label y at time t^- and $b^y(t^-)$ is the total number of nodes that had label y at time t^-. If none of nodes in $B_i(t)$ have a label then $L_i(t) = \emptyset$. Then, node i runs a Decentralized Cluster Labeling (DCL) algorithm (cf. Algorithm 2) with the input $(B(t), L(t)) = (B_i(t), L_i(t))$ to get the output $Y_j(t) = Y(t)$. Note that all nodes in a cluster must have the same label, because all nodes in this cluster obtain the same information from the neighbor discovery protocol.

Algorithm 2: The DCL Algorithm - Input: $(B(t), L(t))$**, Output:** $Y(t)$**.**

```
1   Y(t) = φ
2   While L(t) ≠ ∅
3       z = argmax_y b^y(t)
4       If b^z(t) > b^z(t^-)/2 then
5           Y(t) = z /* nodes lose label y */
6           Break /*stop while loop */
7       Else
8           L(t) = L(t) \ (z, b^z(t), b^z(t^-))
9       End if
10  End while
11  If Y(t) = φ and |B(t)| ≥ h then
12      Y(t) = random /* nodes assign new label
        */
13  End if
```

Note that the DCL algorithm is directly inspired by the CCL algorithm described in Section III-C. The DCL algorithm 2 is based on two ideas. The first is how a CP is formed, and the second is how the CP is maintained. A CP is formed when there are more than h nodes in a cluster with no assigned label. The new CP gets a unique new label y (e.g., a random number). Let us now see how the CP y is maintained. Assume that nodes move and that after time Δt the change in neighborhood is noticed in several clusters. The idea is that only the one cluster with more than half nodes from CP y can be labeled as y. In this way we ensure that only one cluster can inherit label y. If there are several possible labels for a cluster then the majority label will be accepted by all nodes in the cluster.

Let us now show in a concrete example how the labeling algorithm works. Figure 10 shows a network at time t^- that has three clusters of which only one is a CP and it has label 15. The number above the node is a label if the node has it. Figure 11 shows the network at time t before the labeling took place.

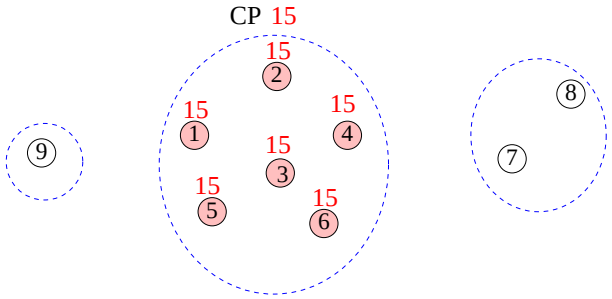


Fig. 10. At time t^-: three clusters, one is a CP with label 15, $b^{15}(t^-) = 6$.

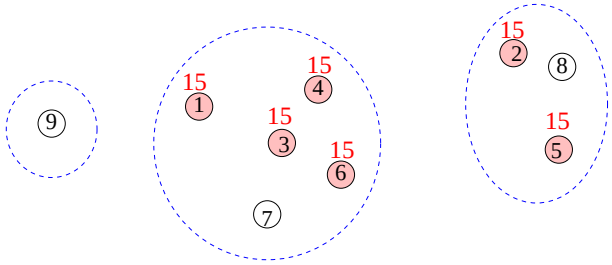


Fig. 11. At time t, before labeling.

Let us study how for example nodes 9, 4 and 8 determine their new labels at time t. The information that node 9 obtains from the neighbor discovery protocol is:

$$\begin{aligned} B_9(t) &= \{9\}, \\ L_9(t) &= \emptyset, \end{aligned} \tag{1}$$

The information that node 4 obtains is:

$$\begin{aligned} B_4(t) &= \{1, 3, 4, 6, 7\}, \\ L_4(t) &= (15, 4, 6), \end{aligned} \tag{2}$$

and the information that node 8 obtains is:

$$\begin{aligned} B_8(t) &= \{2, 5, 8\}, \\ L_8(t) &= (15, 2, 6). \end{aligned} \tag{3}$$

Note that $B_9(t^-) = \{9\}$, $B_4(t^-) = \{1, 2, 3, 4, 5, 6\}$ and $B_8(t^-) = \{7, 8\}$. Because

$B_9(t) = B_9(t^-)$, $B_4(t) \neq B_4(t^-)$ and $B_8(t) \neq B_8(t^-)$ only nodes 4 and 9 run Algorithm 2.

The DCL algorithm in node 4: $b^{15}(t) > b^{15}(t^-)/2$ (because $b^{15}(t) = 4$, $b^{15}(t^-) = 6$) and therefore $Y_4(t) = 15$. The DCL algorithm in node 8: $b^{15}(t) \leq b^{15}(t^-)/2$ (because $b^{15}(t) = 2$, $b^{15}(t^-) = 6$) and therefore $Y_8(t) = \phi$. Figure 12 shows new determined labels at time t.

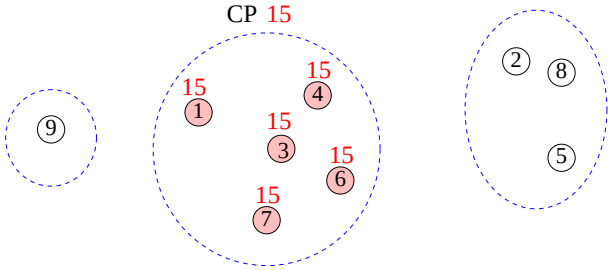


Fig. 12. At time t, after labeling.

Difference with the Vertex Labeling Algorithm given in [9]. The vertex labeling in [9] relies on two simplified assumptions. One is that the connectivity of nodes outside of CPs is very poor such that a node can not communicate with anyone else if it is traveling between CPs. The other is that every node knows its neighborhood set at any continuous time t and hence the neighborhood in two different times can not differ for more than one node. The improved vertex labeling in this paper relaxes these assumptions. We achieve this by changing both the way a CP is formed and the way the CP is maintained. In [9] two nodes are sufficient to form a CP, thus there we have $h = 2$, whereas here we set h to a higher value. Moreover, in [9] a node can lose a label only if it becomes isolated, whereas here the node can lose a label both if the majority of nodes in its cluster left and if the node becomes isolated.

1) Determining h: Next, we discuss how to determine the parameter h of the vertex labeling algorithm. A larger h results in more stable CPs, but also in a smaller number of formed CPs. As we have seen in Section I, for IH it is desirable that CPs be stable. However, if the number of CPs is smaller then IH may be less efficient because it loses opportunities to forward messages. For IH we want to have as many stable CPs as possible. Therefore, we want to find the minimum h that ensures the stability of a label.

To simplify our analysis, we look at the evolution of single cluster and we model it by a simple queuing model. We assume that nodes arrive and leave this cluster according to a random process and we assume that arriving nodes do not have any labels. We apply the labeling algorithm described above on this single cluster. Depending on how the number of nodes in the cluster fluctuates, the cluster is a CP with a label for some time, then it loses the label and is not a CP for some time, then it generates a new label, and so on. We show this in Figure 13, where we denote as T_{CP} the time the cluster is a CP and T_{non-CP} the time the cluster is not a CP. Our goal is to find h such that a new label for this cluster is generated rarely. More formally, we want to find the minimum h such that $E[T_{CP} + T_{non-CP}] \geq \tau_s$, where τ_s is a value that controls the desired stability of the CP.

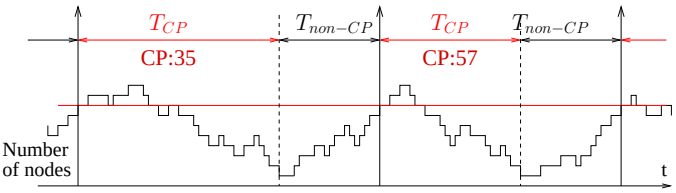


Fig. 13. Evolution of the labeling process of a cluster.

We model the cluster as an $M/M/\infty$-queue, where nodes arrive according to a Poisson process with arrival rate λ and every node stays for an exponentially distributed time with mean μ^{-1}. The mean number of nodes at the cluster is denoted by $\rho = \lambda/\mu$. Let the Markov process $N_t \in \{0, 1, 2, \ldots\}$ denote the number of nodes at the cluster at time t.

To further simplify the analysis, we assume that a period Δt is short with respect to μ^{-1}, so that only one node can leave the cluster during the period Δt, i.e., between the two consecutive executions of the labeling algorithm. This means that the cluster may lose its label only when $N_t = 1$.

Let

$$D_p(q) := \inf\{t > 0 : N_t = p | N_0 = q\}$$

denote the first passage time of state p from state q. Then, the value of interest for us is an expected value of $D := T_{CP} + T_{non-CP} = D_1(h) + D_h(1)$. Therefore, we search for the minimum h such that $E[D] \geq \tau_s$ for every value of λ and μ.

Using the following results [22]:

$$E[D_1(h)] = \sum_{k=1}^{h-1} \frac{1}{\lambda} \sum_{j=k+1}^{\infty} \frac{k!}{j!} \rho^{j-k}$$

$$E[D_h(1)] = \sum_{k=1}^{h-1} \frac{1}{\lambda} \sum_{j=0}^{k} \frac{k!}{j!} \rho^{j-k},$$

we find:

$$E[D] = \frac{e^{\rho}}{\lambda} \sum_{k=1}^{h-1} \frac{k!}{\rho^k}.$$

Let us express $\tau_s = n_s/\lambda$, where n_s is a constant that we can tune depending on the desired stability

of CPs. Thus, we need to find the minimum h such that

$$F_h(\rho) = e^{\rho} \sum_{k=1}^{h-1} \frac{k!}{\rho^k} \geq n_s \text{ for every } \rho.$$

Because it is impossible to find this analytically, we plot $F_h(\rho)$ for the fixed values of $h = 10, 12, \ldots, 20$ (c.f. Figure 14). We see that $F_h(\rho)$ has one minimum. Thus if we set h to a certain value, then the minimum of $F_h(\rho)$ over all ρ, denoted as F_h^{min}, is the constant n_s we can certainly achieve for all ρ. We find F_h^{min} numerically by finding ρ for which $dF_h(\rho)/d\rho = 0$ and plugging it into $F_h(\rho)$. We show the dependence of $F_h^{min} = n_s$ on the values of h in Figure 15. Depending on the desired stability of CPs, we specify the parameter n_s and then we find h using this figure.

In Figure 15 we see that an increase of h leads to an exponential increase of F_h^{min}. This means that the time during which a CP is stable grows exponentially with an increase of h, which is good news. Thus, for IH to work well, we expect that a fairly small value of h will be able both to make CPs stable enough and to form a large enough number of CPs in the network.

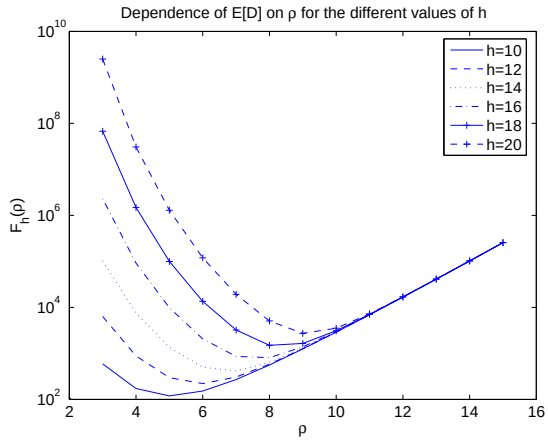


Fig. 14. Dependence of $E[D]$ on ρ for different fixed values of h.

B. Edge Discovery

As a node moves on the graph, it observes the label of every CP it visits, as described in the previous section. If a node moves directly from CP u with label y_u to CP v with label y_v, then this indicates the existence of a labeled edge (y_u, y_v), and we say that the node *directly observes* this edge.

To discover the edge set E, it would be possible for each node to rely only on its own observations of the edges it traverses. However, this approach has the following drawbacks. First, if node mobility is such that a node does not visit the entire graph,

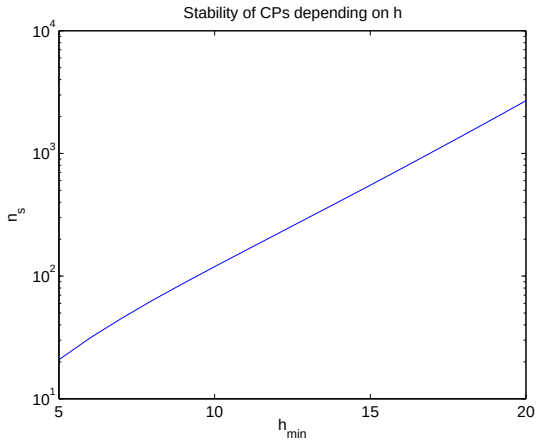


Fig. 15. Minimum value of $E[D]$ as a function of h.

then this node will never discover some parts of the graph, which can result in poor forwarding decisions. Second, even if a node moves over the entire graph, the discovery process would be rather slow, and the transient time (until every node knows most of the CP graph) would be excessively long.

We therefore would like to accelerate the dissemination of edge information to allow every node to learn the entire graph. One approach is for nodes to exchange labeled edges through a gossip protocol. This allows nodes to learn the entire CP graph more quickly.

Note however that the CP graph may change over time. This is either because nodes change their moving patterns and new CPs appear or the existing ones disappear, or because a CP changes its label due to the labeling process as explained in Section IV. For example, if the label of a CP u changes from y_u to y'_u, then all edges with label y_u become obsolete. We use an aging mechanism to eliminate such obsolete edges.

More precisely, in our gossiping scheme, node i's view of the CP graph, G_i, is represented by the set of pairs (e, t_{obs}), where t_{obs} is the time when edge e was directly observed. Node i has edge e in G_i either if it has directly observed e, or if it has received e through gossiping from other nodes.

Every node, upon arrival at a vertex, gossips a constant number n_e of randomly chosen entries (e, t_{obs}) from its view of the graph G_i to all nodes at this vertex. Node i updates G_i either when it directly observes an edge, or when it receives a gossip message from another node. When node i directly observes edge e at time t, it adds (e, t) to G_i and deletes the old entry for e from G_i if it exists. When node i receives an observation (e, t_{obs}) through gossiping for which it already has an entry, it retains the more recent of the two. If it does not have an entry for e, then it adds (e, t_{obs}) to G_i.

As we said, in addition to the process of learning edges, a node uses an aging mechanism to remove an edge from the graph if the edge grows too old. More precisely, a node removes entry (e, t_{obs}) from its graph at time $t_{obs} + T_{age}$, where T_{age} is a fixed constant for all nodes.

V. Island Hopping (IH)

Island Hopping (IH) is a mobility-assisted forwarding algorithm in which a node makes decisions about when to pass a copy of a message to other nodes and when to discard it, by using the knowledge of:

1) the CP graph, and its own position in that graph, and
2) the destination's position in the CP graph.

The three design goals are to minimize the number of copies made of a message, to minimize the end-to-end delay, and to maximize the delivery rate.

In this section we describe our IH scheme under the assumption that a node has knowledge of the CP graph G and its own position in G at all times. Inferring this knowledge in scenarios where nodes have external signals from the environment (such as GPS coordinates or signals from fixed beacons) is easier than in the case where no such external information exists. In Section IV, we showed how nodes could infer this knowledge without such external signals.

A. Message Progression Towards a Fixed Destination

In this subsection, we show the main ideas of IH under the further simplifying assumption that nodes know the position in G of a message's destination. In the next subsection, we then show how nodes can locate the destination.

Our IH scheme uses the following three ideas, which we illustrate in Figure 16.

1) Routing a Message through a Sequence of CPs: Assume that a node i, currently located at vertex $u \in V$, has a message m with destination node D located at vertex $w \in V$. The key is for node i to decide which vertex v should be the next hop in V for message m in order to make progress towards the destination. This desired next hop v is stored in the message in the field $m.next\_hop$. We choose this next hop v as a neighboring vertex on the shortest path between vertices u and w in the CP graph.

The next move of node i is in general not yet known. We allow node i to keep m for several hops until it reaches the desired next hop. For this purpose, there is a field $m.ttl$ set to the maximum allowed number of hops n_h. Thus, every time node i moves to a new vertex $v' \neq v$ it decrements $m.ttl$. Node i discards m when $m.ttl = 0$. If node i happens to move to the desired next hop v, then node i keeps m and sends m to other nodes in vertex v.

This process continues until the message reaches node D at vertex w.

In Figure 16(a), node S at vertex 1 originates a message m to node D at vertex 4. Node S makes several copies of m with $m.next\_hop := 2$ and $m.tll = 1$ (because of simplicity we set $n_h = 1$ in this example). Figure 16(b) shows what happens when these copies move to neighboring vertices. The node with the copy that moves to vertex 2 makes new copies of m with $m.next\_hop := 4$; whereas the node with the copy that moves to vertex 3 discards m because $m.next\_hop \neq 2$ and hence $m.ttl = 0$.

2) At Least One Copy Moves to the Next-hop CP: If none of the copies of message m move to $m.next\_hop$, then all these copies of m will be eventually discarded, and m will be lost. To boost the probability that at least one copy of m progresses towards the next-hop vertex, we introduce a "one-hop" acknowledgment (ACK) scheme. The goal of this scheme is to piggyback one-hop delivery information about message m through nodes moving in the reverse direction, and to generate additional copies if needed.

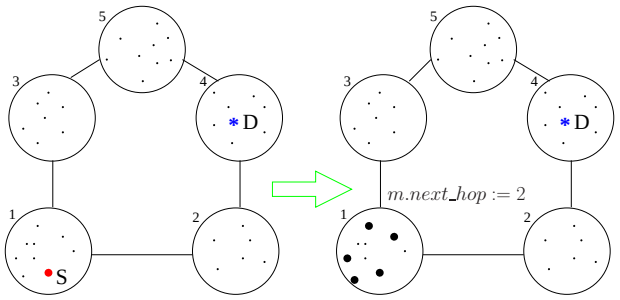
Assume that there exist copies of m at vertex u with $m.next\_hop = v$. Nodes at vertex u should be informed when a copy of m has reached v. When a node with m arrives at v, it broadcasts this fact to all nodes at v. If one of these nodes then moves to u, it broadcasts an ACK for m. All nodes at u can then discard m. But if a node at u holding m has not received an ACK by the time where only a small number c_1 of copies of m are left, it generates additional copies of m. This process repeats for at most c_2 times.

In Figure 16(b), a node with m moves from vertex 1 to vertex 2, where it generates new copies and broadcasts to all nodes the identity of m. In Figure 16(d), one of these nodes arrives at vertex 1 and broadcasts an ACK for m. Then all copies of m at vertex 2 are discarded.

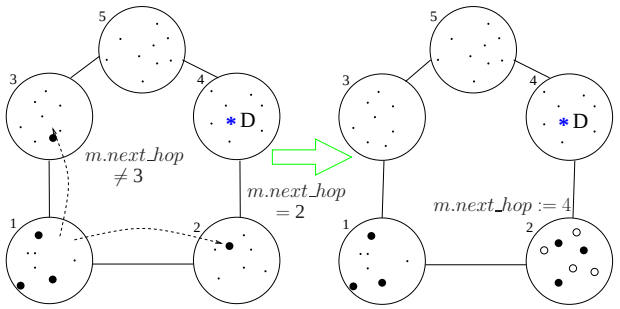
3) Only One Copy Survives to the Next-hop CP: If more than one copy of m with $m.next\_hop = v$ moves into vertex v, then new copies of m can be generated at v several times. This could lead to an exponential increase of the number of copies. We include a mechanism to suppress additional rounds of copying. If node i moves to v, it makes new copies only if none of the nodes currently at v have seen an earlier copy of m arrive at v.

Figure 16(c) shows what happens when a second

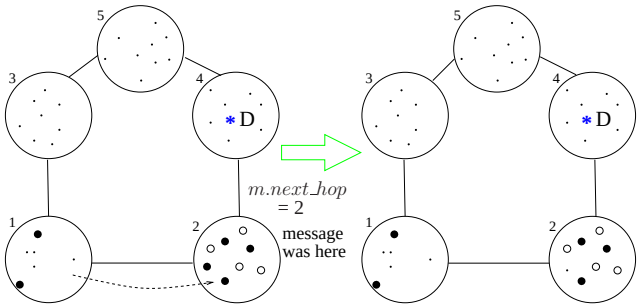
copy of m arrives at vertex 2 from vertex 1. Even if $m.next\_hop = 2$, the copy is discarded, because m has already been at vertex 2.



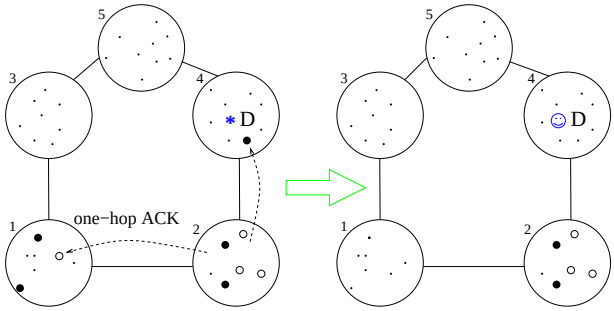
(a) Node S originates a message m to node D.



(b) Node with a copy of m moves to another vertex.



(c) Node with a copy of m moves to a vertex where m has already been observed.



(d) One-hop acknowledgment.

Fig. 16. Island hopping - example.
Small dots - nodes without a copy of m; large dots - nodes with a copy of m; empty dots at a vertex - nodes without a copy of m, but that know that m was at this vertex.

Difference with the Island Hopping Algorithm given in [9]. In the IH algorithm given here we allow copies of a message to live for a fixed number of hops ($n_h > 1$) until they move to the intended next CP, whereas in the algorithm [9] we allow copies to live only one hop ($n_h = 1$) until they move to the next intended CP. This makes IH more robust to the changes in the CP graph. To see this, look at a scenario shown in Figure 17 where an edge $56 - 95$ becomes obsolete. Assume that the node shown in vertex 56 still thinks that this edge exists. Then, this node might choose vertex 95 as the next hop for a message m. If the node discards m immediately when it moves to a vertex that is not the intended next hop, then m would be lost. But, if the node still keeps m for several hops ($n_h > 1$), then m would have a chance of reaching the desired vertex 95.

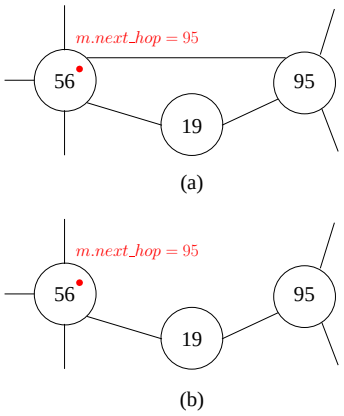


Fig. 17. CP graph: (a) the view of the red node (b) real.

B. Dynamically Locating Destination through Last Encounter Routing

So far, we have assumed that the location of the destination is fixed and known to the message, which is unrealistic. To discover the location of the destination of a message, we cannot resort to the classical methods such as flooding, because the network is partitioned.

To solve this problem, we borrow an approach from [23] called Last Encounter Routing (LER), where a node maintains a Last Encounter Table (LET), with an entry for every other node. An entry consists of the time and location of its last encounter with the node. In [23], the location of the node is its geographic location. We adapt this to our setting, where the location of the last encounter is a vertex: each node remembers for all other nodes the time and the label of the CP where they were last time collocated.

The LETs are used by a message to continually obtain more recent information about the location of the destination, as follows. Assume again that a node i at a vertex u has a message m destined for a node D. As we saw in Section V-A, node i needs to determine the next-hop vertex for m.

Before doing so, node i searches all nodes at u for the most recent LET entry for node D. This location is then used as an estimate of the position of node D to determine the next-hop vertex unless the node already has more recent information. The message remembers this estimate in a field $m.le$. As the message gets closer, it tends to find more recent information, "zeroing in" on the destination.

C. Operation of IH and COGRAD

So far, we have described the IH algorithm as operating on top of an oracle that reveals to every node the CP graph G and the node's position on G. We now describe how IH operates on top of COGRAD, where the CP graph G, and each node's position on G, are obtained through COGRAD. In this case, the obtained graph might contain some errors. This is because the CP graph G may change over time, as explained in Section IV-B; and the changes in G causes that during a period of time a node's view of G either misses new edges that has appeared or still keeps obsolete edges that has disappeared.

As we saw in Section V, a node i located at vertex u chooses the next hop for a message m with the destination at w as the next vertex on the shortest path $u \rightarrow w$ in its view of the CP graph. We have not specified which next-hop is chosen if several shortest paths, and thus several possible next-hops, exist. In this case, we use the age of edges obtained by COGRAD in the edge discovery part to choose between these possible next-hops. We choose the next-hop with the smallest age of its incoming edge from u, i.e., the vertex v whose entry $((u, v), t)$ is most recent.

We discuss two other conditions that can arise in IH due to errors either in the underlying graph discovery algorithm or the locating of the destination. The first situation arises when node i makes the next-hop decision for a message, but cannot find a shortest path to w. This can happen when in i's view of the graph G_i, (i) w is not known, or (ii) a path $u \rightarrow w$ does not exist in G_i. The second situation arises when the destination location (obtained by the last encounter tables as described in Section V) is the current node's location u, but the destination is elsewhere.

We resolve these situations as follows. The node keeps the message with setting $m.ttl = n_h$, and waits until it moves to another vertex v. There, it then tries again to find a next hop. If it does not succeed it decrements $m.ttl$, and if it succeeds it sets $m.ttl = n_h$.

There are still some rare situations in which the IH algorithm does not succeed in delivering a message to the destination. These are: (i) when none of the message copies reach the intended next hop within the n_h traversed hops, (ii) when a CP disappears, but a node still keeps obsolete edges of this CP in the node's view of G, (iii) when a CP changes its label due to the imperfection of the labeling algorithm and does not learn edges with the new label yet but keeps obsolete edges with the old label, and (iv) if a node can not determine a next hop for a message because of the reasons explained above within the n_h traversed hops. In order to keep IH fairly simple and since the performance of our algorithm is only slightly worsen due to these rare situations as shown in Section VI, we do not try to make IH robust to them.

VI. Performance Evaluation

In this section, we evaluate the performance of the IH algorithm in combination with the COGRAD algorithm. For this purpose, we developed a custom simulator implementing the CP graph-based mobility model, and the IH and COGRAD algorithms.

We compare our IH+COGRAD algorithm with ER [15] and PROPHET [16]. We also compare our algorithm with the scheme where a source transmits a message only to the destination (no routing algorithm is used). We denote this scheme as "no R". In addition, we evaluate how our algorithm depends on the various parameters.

A. Simulation Set-up

The nodes' mobility model is the CP graph-based model (described in Section III-B). More precisely, we let $N = c|V|$ nodes perform independent random walks on a graph, with $c > 0$ a parameter controlling the mean number of nodes per vertex. Note that the random walk is the most challenging mobility process for our purposes, for the following reason. When a node performs a random walk, its future movements are independent of the entire past. In other words, even if a node accumulates statistics about its past movements, this will not help predict the future. As such, all the nodes located at a vertex v at a given time t are statistically equivalent, and no information about the past (e.g., keeping track of when a node has last seen the destination, as in PROPHET) can be used to predict where a node will go in the future.

The location of a node is either a vertex $v \in V$ (which implies that the node can communicate with all other nodes currently located at the same vertex v), or an edge $e = (u, v) \in E$ (which implies that the node is en route from island u to island v, and is not able to communicate with any other node). Each node spends an exponentially distributed time

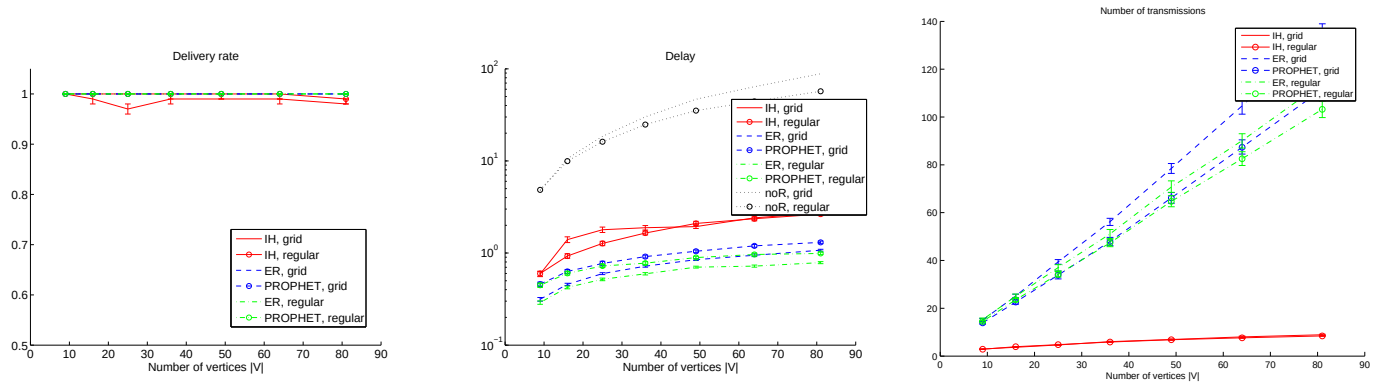


Fig. 18. The grid and the random regular graph; as a function of the number of vertices $|V|$; ($c = 15$, $h = 2$, $n_e = 5$, $T_{age} = 100(T_V + T_E)$, $n_h = 3$). (a) Delivery rate, (b) delay, (c) number of transmissions.

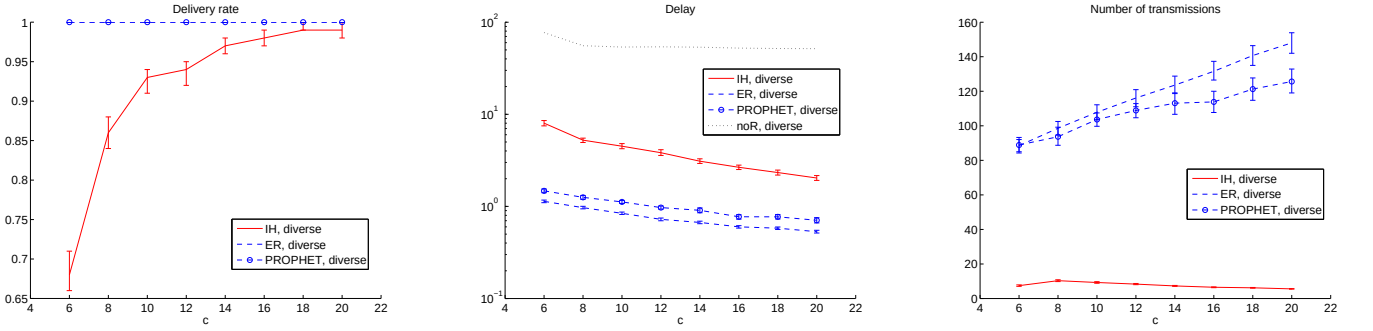


Fig. 19. The diverse degree graph; as a function of the mean number of nodes per vertex c; ($h = 15$, $n_e = 5$, $T_{age} = 100(T_V + T_E)$, $n_h = 3$). (a) Delivery rate, (b) delay, (c) number of transmissions.

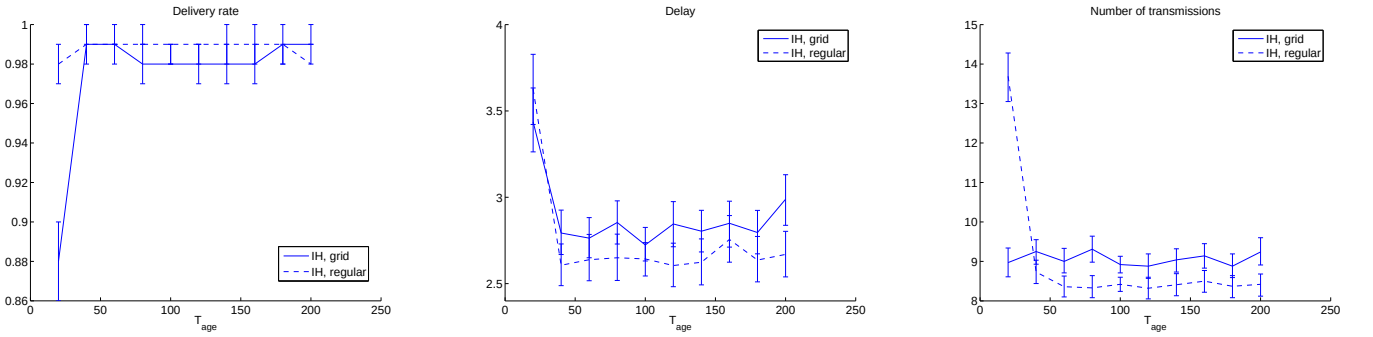


Fig. 20. The grid 9×9 and the random regular graph with $|V| = 81$ and $r = 4$; as a function of the COGRAD parameter T_{age}; ($c = 15$, $h = 2$, $n_e = 5$, $n_h = 3$). (a) Delivery rate, (b) delay, (c) number of transmissions.

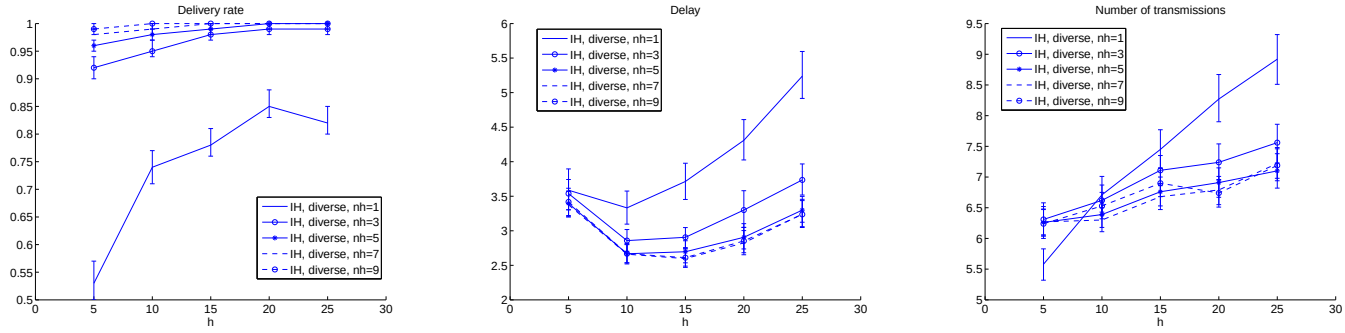


Fig. 21. The diverse degree graph; as a function of the COOGRAD parameter h and the IH parameter n_h; ($c = 15$, $h = 2$, $n_e = 5$, $n_h = 3$). (a) Delivery rate, (b) delay, (c) number of transmissions.

with mean T_V at a vertex, and an exponentially distributed time T_E at an edge, where we set $T_V = 10T_E$. All the delay results we report, as well as all time scales, are normalized by setting $T_V + T_E = 1$, i.e., we normalize to unity the average speed at which a node advances from vertex to vertex.

We present simulation results for various synthetic CP graphs:

- the 2-dimensional $k \times k$ grid, where we vary k from 3 to 9, i.e., $|V| = 9, \ldots, 81$,
- the regular random graph - a random graph where each vertex has the same degree r, with $r = 4$ and with different number of vertices $|V| = 9, \ldots, 81$,
- a diverse degree graph - a random graph with $|V| = 100$, where we divide vertices into ten groups with ten vertices of the same degree in each group and where we vary degrees of vertices from 1 to 10.

More the challenging CP graphs for our algorithm are those with the vertices having the diverse degrees. The reason is that in graphs where vertices have approximately equal degrees, the mean number of nodes per vertex is approximately equal to c for all vertices. Because of this, if c is large enough then all vertices will be well populated with nodes, and thus they will be stable CPs, which is easier for our IH algorithm to perform well. Hence, in our simulation results we include both types of graphs, where the diverse degree graph belongs to the more challenging type.

So far, the only assumption we make about the set of nodes within a same CP is that each node can reach other nodes (either because they are in direct radio range of each other, or because they can form a connected ad hoc network). However, for the purpose of simulations we assume for simplicity that nodes within a CP are directly connected. This assumption is favorable for both our algorithms and for the epidemic-based algorithms ([15],[16]) we compare with, because it makes it possible to transmit a message to all nodes at a CP in a single broadcast. If nodes at a CP were not directly connected, then the broadcast function would have to be replaced by a flooding primitive.

Each simulation we report is preceded by a warm-up phase that is needed to populate the last encounter tables (LETs). The warm-up phase terminates if 80% of the node pairs have encountered each other at least once; note that this is conservative in that the LE tables are asymptotically fully populated.

Other fixed parameters in our simulations are: $c_1 = 3$ and $c_2 = 2$ in IH (defined in Section V), and the maximum number of transmissions of a message allowed that one node performs in ER and PROPHET is set to 1000.

B. Performance Metrics

We use the following metrics:

- **delivery rate** - the number of messages delivered divided by the total number of messages sent by sources
- **delay** - the normalized delay for the delivered messages
- **number of transmissions per message** - how many times a message is transmitted until there are no more copies of this message in the network.

We compute these metrics by averaging over a number of randomly chosen source-destination pairs, where for each pair a source sends a single message to its destination. In simulation results we show the mean values of these metrics with 95% confidence intervals.

Note that the number of transmissions per message includes only the transmissions of actual messages, not the control messages generated by IH and COGRAD. We justify this as follows. First, in the IH algorithm, control messages result only when a traffic message is to be transmitted. Therefore, the IH control overhead should be considered relative to the overhead to transmit messages. Usually, data

messages tend to be orders of magnitude larger than control messages. This is in compliance with the proposed architecture for delay tolerant networks in [4]. Therefore, the IH overhead per data message can be neglected. Second, in the COGRAD vertex labeling algorithm, control overhead accrues when a node discovers its neighborhood set $B_i(t)$, and this type of control overhead is also present in ER and PROPHET. Third, in the COGRAD edge discovery algorithm, control overhead accrues when a node arrives at a vertex and broadcasts a small number of edges. In comparison, ER and PROPHET exchange a summary vector and predictability vectors respectively, whenever a node meets a new node, which in our setting means the broadcast of these vectors per each node's arrival at a vertex. Therefore, the control overhead in our scheme is comparable to that in ER or PROPHET, and small compared to data messages - except when the network is very lightly loaded. Hence, we consider only traffic messages in our simulation results.

C. Simulation Results - Comparison with ER and PROPHET

Here, we compare our IH+COGRAD algorithm with ER and PROPHET. We discuss the tuning parameters of our algorithm in Section VI-D.

We show in Figure 18 the performance metrics for 1) the grid topology and 2) the regular random graph topology, as a function of the number of vertices $|V|$, for fixed $c = 15$.

Figure 18(a) shows the delivery rate. We notice that the delivery rate of the flooding-based approaches (ER, PROPHET) is essentially 100%. The delivery rate of IH is slightly lower, but remains very close to 1. We discussed the reasons for this slight drop-off in Section V-C.

Figure 18(b) shows the end-to-end delivery delay. Although the delay of IH is higher than that of flooding-based approaches (around 1.5 times higher), the figure suggests that it is only a small constant factor higher than ER/PROPHET as a function of network size.

Figure 18(c) shows clearly the main advantage of our scheme: it requires a significantly lower overhead per message than ER/PROPHET.

We show in Figure 19 the performance metrics for the diverse-degree graph, as a function of the parameter c to explore the sensitivity of IH to small values of c. We see in Figure 19(a) that our algorithm achieves a high delivery rate (above 90%) even for a fairly small value of $c = 10$.

Figures 19(b) and 19(c) confirm our finding that IH+COGRAD achieves a very favorable tradeoff, with a delay close to that of flooding-based approaches that are essentially the lowest possible delay, but with much lower transmission overhead, which implies that a network operating under IH+COGRAD has a capacity gain of more than an order of magnitude over ER/PROPHET for the scenarios considered here.

This favorable tradeoff is possible because our scheme tightly controls the copies of a message, by not making new copies of a message that strays from the shortest path towards the destination. In flooding-based approaches, messages diffuse throughout the network; in particular, it is difficult in these approaches to ensure that all copies of a message get discarded after one copy of the message has been delivered to the destination. This problem does not arise in IH.

D. Simulation Results - Dependence on the Parameters of Our Algorithm

Let us recall the control parameters of our algorithm. In COGRAD in vertex labeling, there is the parameter h, the threshold of the number of nodes when a new CP is formed. In COGRAD in edge discovery, there is the parameter n_e, the number of edges that each node gossips upon its arrival at a vertex, and there is the parameter T_{age}, the age threshold fixed for all nodes upon which an edge is removed from the CP graph. In IH, there is the parameter n_h, the maximum number of hops (i.e. traversed vertices) that a message can live before it reaches the desired next hop.

In Figure 20 we show the dependence on T_{age}, and in Figure 21 we show the dependence on n_h and h and in both cases we set $n_e = 5$.

From all reported simulation results we see that a fairly small $n_e = 5$ gives satisfactory results (e.g., see Figure 18).

We see in Figure 21 that the increasing of n_h makes better all considered performance metrics of IH+COGRAD. However, the larger n_h makes the memory consumption grow; thus, we limit ourselves to $n_h = 3$.

There are still T_{age} and h left to be determined. We set first T_{age} independently of h, and then look at the dependence on h with already tuned T_{age}. We achieve this by looking at the dependence on T_{age} in the cases of the CP graph with vertices with non-diverse degrees for large c (the grid and the regular graph for $c = 15$, in Figure 20). Here, all vertices are well populated with nodes on average, which makes them stable CPs even with $h = 2$, hence in these cases h is not important.

We see in Figure 20 that T_{age} need to be larger than some value but after this value the performance metrics are not sensitive for the broad range of values of T_{age}. Of course, for very large values

of T_{age}, the performance will be degraded because there will be more obsolete edges that are gossiped, thus wasting transmission resources. Therefore, it is important to properly set-up the lower bound for T_{age}.

Finally, we look at the dependence on h for other already tuned parameters. We look at the case where the CP graph is the diverse degree graph for $c = 15$ in Figure 21. Here, some vertices (those with low degree) will be poorly populated with nodes, and hence if $h = 2$ they will be unstable CPs. We expect that here the parameter h plays a significant role by avoiding that unstable CPs be formed. Our simulation results shown in Figure 21 confirm this.

The remained question is how nodes can estimate the parameters T_{age} and h. We saw in Section IV-A.1 that if a node can estimate τ_s (the average required life time of a label) then it is easy to find h. We argue that one rough but good enough estimate for both T_{age} and τ_s is the mixing time of the random walk on the graph (we denote it as T_{mix}). This time is the time neeeded for the random walk to approach its stationary distribution. This means that after this time a node's position is independent of its starting position. Hence, if we set up $T_{age} = \tau_s = T_{mix}$ we expect that this time is long enough to ensure that the nodes remove only the obsolete edges from the CP graph and that the CPs are enough stable. According to [24], T_{mix} is of the order of $1/log(1/x)$, where x is the second largest eigenvalue in modulus. The calculations showed a good match with the simulation results. For example, for the grid $9x9$ the calculated $T_{mix} \approx 29.2(T_V + T_E)$, which gives $h \approx 15$ ($n_s = \tau_s\lambda = \tau_s c\ (T_V + T_E) \approx 400$, then the result for h follows from Figure 14, Section IV-A.1). According to the simulation results shown in Figure 20 these values gives good results.

Comparison with IH+COGRAD reported in [9]. The IH+COGRAD algorithm reported in [9] is a subclass of the IH+COGRAD algorithm reported here and is obtained by setting parameters $n_h = 1$ and $h = 2$. Figure 21 shows that setting $n_h > 1$ and $h > 2$ gives a considerable improvement in the case of the diverse degree CP graph where some CPs are not stable. Thus, our algorithm reported here is much more robust to the more realistic mobility models where some CPs might be unstable.

VII. Discussion and Conclusion

We have argued in this paper that node distributions that are heterogeneous in space and relatively stable over time provide an opportunity for mobility-assisted forwarding in partitioned networks, because the underlying topology of concentration points (CPs) and the flows of nodes between the resulting islands of connectivity can be learned and used to make progress towards the destination without flooding the network with too many copies of a message.

We have shown that in the presence of stable CPs, our approach significantly outperforms approaches where copies of messages are made without the benefit of an underlying topology. Our approach achieves end-to-end delays of the same order as aggressive flooding-based approaches, but with up to an order of magnitude fewer transmissions per message. This benefit comes from the ability to decide, at every vertex, whether a particular copy of a message has made progress towards the destination or not; we can realize this benefit *even if the mobility process is not predictable* (random walk), and *even if the CP graph is not known a-priori* (thanks to COGRAD).

To do this, we had to take a significant detour, and first develop a scheme to discover this topology. We have shown that under the assumptions of our mobility model, it is - somewhat surprisingly - possible to achieve this by processing the changing set of neighbors of each node, without relying on any explicit signal from the environment or from dedicated infrastructure.

Once we have inferred the CP graph, we can try to forward messages along the shortest path of islands towards the destination. Although we cannot, of course, control the movement of individual nodes, we can nevertheless make progress towards the destination by making a few copies, and letting only those copies that go in the right direction survive.

For this to work, a message has to be able to locate the destination in the CP graph; for this, we have adopted the idea of last encounter routing described in [23], but here locations are vertex labels rather than geographic coordinates. This allows a message to have an estimate of its destination's current location; the precision of this estimate tends to improve as the message approaches the destination.

Our simulation results show that our approach significantly outperforms schemes that do not explicitly exploit topological information. Of course, this advantage depends on the presence of a stable topology of concentration points; otherwise, it is probably hard to achieve significantly better performance than schemes such as ER and PROPHET. However, we believe that heterogeneous and stable node distributions tend to be the norm rather than the exception, and we hope to establish this concept through further study of a diverse and representative set of mobility traces. The key point we make in this paper is that stable heterogeneity is beneficial, as it provides structural clues that can be exploited

by routing and mobility-assisted forwarding algorithms.

ACKNOWLEDGMENT

The authors would like to thank Henri Dubois-Ferrière, Daniel R. Figueiredo, Maciej Kurant, Hung Nguyen, and Dominique Tschopp for valuable feedback and discussions on this paper. We also thank Holly Cogliati for help in improving the manuscript. We are indebted to the MPT Radio Taxi company in Warsaw, Poland, for making the GPS database available to us. We also like to thank to the Exploratorium - the museum of science, art and human perception for making the San Francisco taxi traces publicly available[3].

The work presented in this paper has been supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5006-67322.

REFERENCES

[1] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.S.Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet," in *Proc. ASPLOS-X 02*, San Jose, CA, October 2002.

[2] "UMassDieselNet: A Bus-based Disruption Tolerant Network," http://prisms.cs.umass.edu/diesel/.

[3] P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot, "Pocket Switched Networks and Human Mobility in Conference Environments," in *WDTN '05: Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*. New York, NY, USA: ACM Press, 2005, pp. 244–251.

[4] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proc. SIGCOMM '03*, August 2003.

[5] W. Zhao, M. Ammar, and E. Zegura, "A message ferrying approach for data delivery in sparse mobile ad hoc networks," in *Proc. ACM Mobihoc '04*, Tokyo Japan, May 2004.

[6] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant networking," in *Proc. SIGCOMM '04*, August/September 2004.

[7] T. Camp, J. Boleng, , and V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research," *Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, vol. 2, no. 5, pp. 483–502, 2002.

[8] D. Johnson and D. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," *Mobile Computing (Tomasz Imielinski and Hank Korth, eds.)*, pp. 153–181, 1996.

[9] N. Sarafijanovic-Djukic, M. Piórkowski, and M. Grossglauser, "Efficient Mobility-Assisted Forwarding in Partitioned Networks," in *IEEE SECON 2006, Proceedings of the 3rd IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2006.

[10] J. Kang, B. Steward, W. Welbourne, and G. Borriello, "Extracting a Places from Traces of Locations," in *WMASH'04: Proceeding of the 2nd ACM international workshop on Wireless Mobile Applications and Services on WLAN Hotspots*, 2004.

[11] W. Hsu, K. Merchant, H. Shu, C. Hsu, and A. Helmy, "Preference-based Mobility Model and the Case for Congestion Relief in WLANs using Ad Hoc Networks," in *VTC2004-Fall: Proceeding of the 60th IEEE VTC*, 2004.

[12] D. Tang and M. Baker, "Analysis of a Metropolitan-Area Wireless Network," *Wireless Networks*, vol. 9, pp. 107–120, 2002.

[13] C. Tuduce and T. Gross, "A Mobility Model Based on WLAN Traces and its Validation," in *Proceeding of the 24th IEEE INFOCOM 2005*, 2005.

[14] M. Kim, D. Kotz, and S. Kim, "Extracting a mobility model from real user traces," in *Proceeding of the 25th IEEE INFOCOM 2006*, 2006.

[15] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks technical report cs-200006," Duke University, Tech. Rep., April 2000.

[16] A. Lindgren, A. Doria, and O. Schelen, "Probabilistic routing in intermittently connected networks," *Mobile Computing and Communications Review*, July 2003.

[17] X. Chen and A. L. Murphy, "Enabling disconnected transitive communication in mobile adhoc networks," in *Proc. Workshop on Principles of Mobile Computing '01*, 2001.

[18] M. Musolesi, S. Hailes, and C. Mascolo, "Adaptive routing for intermittently connected mobile ad hoc networks," in *Proc. IEEE 6th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM'05)*, June 2005.

[19] H. Wu, R. Fujimoto, R. Guensler, and M. Hunter, "Mddv: A mobility-centric data dissemination algorithm for vehicular networks," in *Proc. ACM Workshop on Vehicular Ad Hoc Networks (VANET)*, October 2004.

[20] T. Spyropoulos, K. Psounis, and C. Raghavendra, "Spray and Wait: an efficient routing scheme for intermittently connected mobile networks," in *Proc. Workshop on delay tolerant networking and related networks (WDTN-05)*, August 2005.

[21] S. Theodoridis, *Pattern Recognition*. Elsevier, 2003.

[22] F. Roijers, M. Mandjes, and J. van den BergA., "Analysis of congestion periods of an m/m/inf-queue, report pna-e0606, issn 1386-3711," Centrum voor Wiskunde en Informatica (CWI), Netherlands, Tech. Rep., March 2006.

[23] M. Grossglauser and M. Vetterli, "Locating Mobile Nodes with EASE: Learning Efficient Routes from Encounter Histories Alone," *IEEE/ACM Trans. on Networking*, vol. 14, no. 3, June 2006.

[24] M. Chen, "Mixing time of random walks on graphs, Master Thesis," Mathematics Department, University of York, August 2004, supervisor: Keith Briggs.

[3] `http://www.cabspotting.org`