Measuring Temporal Lags in Delay-Tolerant Networks

Arnaud Casteigts*, Paola Flocchini*, Bernard Mans† and Nicola Santoro‡

* University of Ottawa, Canada,
{casteig,flocchin}@site.uottawa.ca
† Macquarie University, Sydney, Australia,
bernard.mans@mq.edu.au
‡ Carleton University, Ottawa, Canada,
santoro@scs.carleton.ca

***Abstract*—Delay-tolerant networks (DTNs) are characterized by a possible absence of end-to-end communication routes at any instant. In most cases, however, a form of connectivity can be established over time and space. This particularity leads to consider the relevance of a given route not only in terms of hops (topological length), but also in terms of time (temporal length). The problem of measuring temporal distances between individuals in a social network was recently addressed, based on *a posteriori* analysis of interaction traces. This paper focuses on the *distributed* version of this problem, asking whether every node in a network can know precisely and *in real time* how out-of-date it is with respect to every other. Answering affirmatively is simple when contacts between the nodes are *punctual*, using the temporal adaptation of vector clocks provided in [20]. It becomes more difficult when contacts have a duration and can overlap in time with each other. We demonstrate that the problem remains solvable with arbitrarily long contacts and non-instantaneous (though invariant and known) propagation delays on edges. This is done constructively by extending the temporal adaptation of vector clocks to non-punctual causality. The second part of the paper discusses how the knowledge of temporal lags could be used as a building block to solve more concrete problems, such as the construction of *foremost* broadcast trees or network backbones in periodically-varying DTNs.**

I. Introduction

Highly-dynamic networks, and in particular delay-tolerant networks, are characterized by a possible absence of end-to-end communication routes – *direct journeys* – at any instant. In most cases, however, communication can still be achieved over time through disconnected routes – *indirect journeys* – using store-carry-forward-like mechanisms. This particularity led researchers to develop a number of routing techniques based for example on pro-active knowledge on the network schedule [3], [15], probabilistic [22] or encounter-based strategies [7], [13], [16], and new metrics derivated from cotact statistics [19]. A good taxonomy of approaches is presented in [27].

On the analytical side, the time-dimension has had a strong impact on the research, which has focused on extending most of the usual connectivity concepts – e.g, paths and reachability [1], [17], distance [3], diameter [6], or connected components [2] – to a temporal version. In particular, the fact that connectivity takes place over arbitrary long periods of time implies that the latency of a given route no more depends on the sole number of hops separating the nodes. A question that immediately follows is *how far apart in time the nodes can be? can this be measured precisely for every node?*

This type of question has been recently regarded in a number of works from the field of social network analysis [14], [21], [20], [25]. Indeed, social networks are in essence very similar to delay-tolerant networks, and contrarily to these latter, generate large datasets available for *a posteriori* analysis. In [20] in particular, Kossinets et al. ask the question of how out-of-date each node could be with respect to every other node. They provide a *centralized* algorithm – intended to process a known sequence of contact history – to measure these distances based on a temporal adaptation of vector clocks.

Besides looking at the question from a centralized point of view, these studies assume that the contacts between nodes are punctual in time – and generally given as triplets (u, v, t) where u and v are two entities and t a date of contact between them. This assumption is certainly due to the fact that most of the datasets available are traces of punctual interactions, such as email exchanges or message posts on community websites. The situation in DTNs is different, because the contact between nodes can have arbitrarily durations and possibly overlap in time with one another. This aspect renders the computation of exact temporal distances more complex because it implies the possible co-existence of indirect routes on the one hand, and *continuums* of direct routes on the other hand. Typical DTNs exhibit a mixture of both in various proportions.

In such a context, we look at the *distributed* version of the problem and ask: *is it possible for a node to know precisely, and in real time, how out-of-date it is with respect to every other node in the network?* We answer positively to this question for the case of contacts with arbitrary durations and non-instantaneous (though invariant and known) propagation delay on edges. The feasibility is demonstrated through an algorithm that further extends and generalizes the temporal adaptation of vector clocks done in [20].

The second part of our work is concerned with how the knowledge of temporal lags could serve as a building block for more concrete network problems, such as the distributed construction of *foremost* broadcast trees in periodically-varying networks (following a line of work initiated in [4]). The solution we provide builds a set of time-dependent broadcast trees that guarantee the earliest possible delivery for any

IPDPS'11, May 16–20, 2011, Anchorage, Alaska, USA.

potential emitter and emission date. Interestingly, the union of these trees for a given emission date is precisely what the authors of [20] refer to as *network backbones*. Our algorithm therefore computes all the backbones as a by-product.

In both parts of the paper, we use the *time-varying graph* formalism proposed in [5] to describe the environment and interaction between entities, as well as to analyze the protocol correctness. This formalism, which is semantically equivalent to that of *evolving graphs* [8], offers in comparison an *interaction-centric* perspective that proves more convenient to express and manipulate temporal aspects that this work requires, *e.g.* focusing on the evolution of an edge independently from that of the entire graph.

The paper is organized as follows: in Section II, we describe the model and terminology that are used throughout the paper. Section III discusses the problem of measuring temporal lags in the general case of overlapping contacts and non-instantaneous progapation delays, and Section IV shows how this information can be leveraged to build foremost broadcast trees (and network backbones) in the particular case of periodically-varying DTNs.

II. Model and Terminology

Consider a set V of *nodes*, susceptible of making contacts with each other over a (possibly infinite) time interval $\mathcal{T} \subseteq \mathbb{T}$, called *lifetime* of the system; the temporal domain $\mathbb{T}$ corresponds here to $\mathbb{R}$ (continuous-time system). Let the contacts between nodes define a set of intermittently available undirected edges $E \subseteq V \times V$ such that $\forall x, y \in V, (x, y) \in E \Leftrightarrow x$ and y have at least one contact in $\mathcal{T}$.

Using the *time-varying graph* formalism from [5], we represent the network as a structure $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$, where $\rho : E \times \mathcal{T} \rightarrow \{0, 1\}$, called *presence* function, indicates whether a given edge is available at a given time, and $\zeta : E \times \mathcal{T} \rightarrow \mathbb{T}$, called *latency* function, indicates the time it takes to propagate a message over a given edge at a given date. In this work, we assume the latency function to be constant for all edges and presence times, and thus denote it as a constant value ζ called *propagation delay*. If a message is sent less than ζ time before the disappearance of an edge, it is lost. The intervals of presence of an edge, on the other hand, can be arbitrarily long and can vary among several appearances of the edge. Given an edge e, we allow the notation $\rho_{[t_1,t_2)}(e) = 1$ to signify that $\forall t \in [t_1, t_2), \rho(e, t) = 1$.

A sequence of couples $\mathcal{J} = \{(e_1, t_1), (e_2, t_2) \ldots, (e_k, t_k)\}$, where $\{e_1, e_2, ..., e_k\}$ is a walk in G and $t_i + \zeta \leq t_{i+1}$ for $1 \leq i < k$, is a *journey* in $\mathcal{G}$ iff $\rho_{[t_i,t_i+\zeta)}(e_i) = 1$. We will denote by $departure(\mathcal{J})$, and $arrival(\mathcal{J})$, the starting date t_1 and the last date $t_k + \zeta$ of the journey $\mathcal{J}$, respectively. Journeys can be thought of as *paths over time* from a source to a destination and therefore have both a *topological* length and a *temporal* length[1]. The *topological length* of $\mathcal{J}$ is the number $|\mathcal{J}|_h = k$ of couples in $\mathcal{J}$ (i.e., the number of *hops*), and its *temporal length* is its duration $|\mathcal{J}|_t = arrival(\mathcal{J}) - departure(\mathcal{J}) = t_k - t_1 + \zeta$. For example the journey $\{(ac, 2), (cd, 5)\}$ in Figure 1 has a topological length of 2, and a temporal length of $3 + \zeta$ units of time.

[1]The concept of temporal distance in a delay-tolerant network seems to have been first formalized in [3]. Interestingly, this concept has been used under various terminologies, e.g. *reachability time* [14], *information latency* [20], or *temporal proximity* [21]. The concept of journey (term used in [3]), has been variously called *schedule-conforming path* [1], *time-respecting path* [14], [17], or *temporal path* [6].

Let us denote by $\mathcal{J}^*_{\mathcal{G}}$ the set of all journeys in time-varying graph $\mathcal{G}$, and by $\mathcal{J}^*_{(u,v)} \subseteq \mathcal{J}^*_{\mathcal{G}}$ those journeys starting at node u and ending at node v. Clearly, the concept of journey is not symmetrical: the existence of a journey from u to v in $\mathcal{G}$ does not imply the existence of a journey from v to u; this holds regardless of whether the edges are directed or not, because the time dimension creates its own level of direction. For example, in the time-varying graph of Figure 1, there are several journeys from a to d whereas $\mathcal{J}^*_{(d,a)} = \emptyset$.

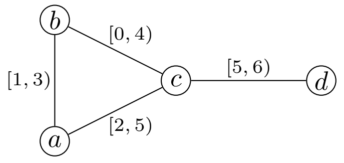


Figure 1. A time-varying graph $\mathcal{G}$; the labels on the edges indicate the time intervals in which those edges are present. The propagation delay is $\zeta \leq 1$.

We say that a journey is *direct* if the presence of every two successive edges overlap in time and their use follow on directly; it is said *indirect* otherwise. An example of direct journey in the graph of Figure 1 is $\mathcal{J}_1 = \{(ab, 2), (bc, 2+\zeta)\}$. Examples of indirect ones include $\mathcal{J}_2 = \{(ac, 2), (cd, 5)\}$, and $\mathcal{J}_3 = \{(ab, 2), (bc, 2 + \zeta), (cd, 5)\}$.

Entities can detect the appearance or disappearance of an incident edge, and associate dedicated operations in reaction to both events. Since the presence intervals are by convention right-open, we assume that the disappearance of an edge at time t is always handled *before* the appearance of an edge at date t, which is consistent with looking at journeys whose presence intervals strictly follow each other as indirect ones (*e.g.* $\mathcal{J}_{(a,d)} = \{(ac, 5-\zeta), (cd, 5)\}$ in the above example).

Processing times are considered negligible. We assume that the message propagation time ζ is known to the nodes. The entities do not necessarily share the same global time, but local clocks advance at the same rate. The schedule of the network ρ is *not* known to the nodes. Finally, because the objective is to collect information with respect to remote nodes, we assume unique identifiers; this assumption is not strictly necessary (*e.g.* could be replaced by an adaptation of *sense of direction* [11] to dynamic networks).

III. Measuring Temporal Lags

This section is concerned with the general problem of making the nodes aware, at any time, of how *out-of-date* they are with respect to every other node. An appropriate abstraction for this problem is certainly that of *temporal view*, introduced in [20] in the context of social network analysis[2].

[2]This concept was called simply "view" in [20]; since the term *view* has a very different meaning in distributed computing (e.g., [26]), the adjective "temporal" has been added to avoid confusion.

The temporal view that a node v has of another node u at time t, denoted $\phi_{v,t}(u)$, is the latest (i.e., largest) $t' \leq t$ at which a message received by time t at v could have been emitted at u; that is, in our formalism,

$$\phi_{v,t}(u) = \text{Max}\{departure(\mathcal{J}) : \mathcal{J} \in \mathcal{J}^*_{(u,v)} \wedge arrival(\mathcal{J}) \leq t\}.$$

By convention, $\phi_{v,t}(v) = t$ for any node v and time t, and $\phi_{v,t}(u) = -\infty$ if no journey from u to v exists before t. The question under investigation is: can the nodes know their temporal views in real time? That is, can a node v know the exact value of $\phi_{v,t}(u)$ at any time t for any node u?

A. Instantaneous contacts

This question has a simple answer if contacts between nodes and propagation delays are both *instantaneous*, the situation examined by Kossinets, Kleinberg, and Watts [20]. Their solution consists of using a "temporal" adaptation of the *vector clock* mechanism. Vector clocks were introduced independently by Fidge [9] and Mattern [24] to track causality relations between events that processes generate in a distributed system, when no assumption are made with respect to the processes clocks; they establish a type of logical time that ensures a complete causal ordering of the events in the system. As shown in [20], the same mechanism can be used to measure temporal distances between nodes.

The decentralized solution, easily adapted from the centralized algorithm of [20], is described in details in Algorithm 1. Assume for simplicity that all the local clocks share the same global time; this assumption can be easily removed, as explained at the end of the section. Informally, every node v stores and maintains a vector of all its temporal views $\phi_{v,t} = (\phi_{v,t}(u) : u \in V)$ refered to as its *vector clock*. Initially the vector contains only its own reflexive temporal view. Whenever a contact occurs between two nodes, they exchange their vectors; each will then locally operate a nodewise maximum between their temporal views (new views are inserted by copy); the resulting vector is considered as the new local vector on both sides.

Algorithm 1 Measuring temporal lags with instantaneous contacts.

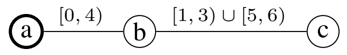
```
1:  $VectorClock\ vec \leftarrow \emptyset$

2:  onContact with a neighbor $ng$:
3:  $vec[myself] \leftarrow now()$
4:  $send(vec)\ to\ ng$

5:  onReception of a vector clock $vec_{ng}$:
6:  for all $n \in vec_{ng}.nodes()$ do
7:      if $n \notin vec.nodes()$ or $vec_{ng}[n] > vec[n]$ then
8:          $vec[n] \leftarrow vec_{ng}[n]$

9:  getView(Node $n$):
10: $return\ vec[n]$
```

In this code, $now()$ returns the current local time; $nodes()$ called on a given vector clock returns the list of nodes within (without the associated views); $myself$ stands for the underlying node.

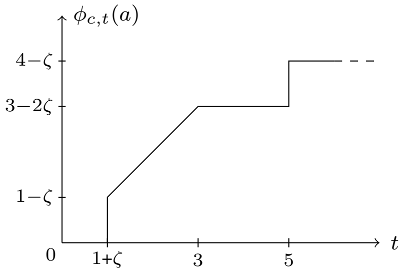
It can be easily proven that the value of the function $getView()$ called with parameter u at node v and time t returns $\phi_{v,t}(u)$.

B. Impact of lasting contacts on the temporal views

Algorithm 1 assumes that the duration of an edge appearance (and communication) is instantaneous. The general case of contacts with arbitrary durations and non-instantaneous propagation delays ($\zeta > 0$) is clearly more complex and, until now, no solution exists. In particular, a significant complication, due to the fact that contacts are not instantaneous, is that several edges can overlap in time at a same node. Consider for example the graph shown in Figure 2(a); the temporal view that node c has of node a is depicted as a function of time in Figure 2(b). Contrary to the case of instantaneous contacts – where temporal views can only increase by discrete steps – there is here a mixture of discrete increases (e.g. at time $1+\zeta$ or 5) and continuous increases (from time $1+\zeta$ to 3).



(a) A simple time-varying graph; $\zeta \leq 1$.



(b) View that node c should have of node a.

Figure 2. Nature of the temporal view in presence of overlapping contacts.

This continuous artefact is due to the existence of direct journeys – or more precisely, a *continuum* of direct journeys – taking place between a and c. Interestingly, the temporal view that c has of a during this period only depends on the journeys topological lengths, here 2, which is why node c could progressively receive between times $1+\zeta$ and 3 any message emitted by a between times $1-\zeta$ and $3-2\zeta$. It is important to note that direct journeys are in fact not systematically faster than indirect ones (imagine a very long direct journey, versus a short indirect one whose edges presences closely follow each other). This implies that the temporal view can possibly be determined by an indirect journey even when a direct one is available.

In the next sections we denote the temporal views that are due to direct journeys and indirect journeys by the terms *direct view* and *indirect view*, respectively.

C. The algorithm

Keeping track of direct and indirect views is done in different ways. Measuring indirect views can be done using the same mechanism as in Algorithm 1 for instantaneous contacts. The case with direct views is different, because the nodes cannot exchange an infinity of message to track the continuous increase of a view. As mentioned above, the duration of a direct journey only depends on its topological length. So if the nodes know exactly what are the lengths of the (shortest) direct journey currently arriving to them from the other nodes, then they can deduce their direct view in real time. Let us call this length the *level* of a node with respect to another. Formally

$$level_{v,t}(u) = \\ Min\{|\mathcal{J}|_h : \mathcal{J} \in \mathcal{J}^*_{(u,v)} \wedge isDirect(\mathcal{J}) \wedge arrival(\mathcal{J}) = t\},$$

where $isDirect(\mathcal{J})$ is true iff $\mathcal{J}$ is a direct journey. By convention, $level_{v,t}(u)$ is considered to be $-\infty$ if no direct journey from u is arriving to v at time t. Observe that the definition of level is relative to the *reception* only, and is not concerned for example with the fact that some edges of the journey may have disappeared by the reception time. The only thing that matters is to know *locally* that messages could still be currently arriving from a given emitter through a given number of hops. Adopting this particular definition of the level is the key point, because it allows us to consider that the direct view that v has of u at time t is always equal to $now() - level_{v,t}(u) \times \zeta$.

Now that the appropriate concepts of *views* and *level* are considered, the intuition behind the algorithm follows quite naturally. Reusing the idea of vector clocks, we adapt the mechanism of Algorithm 1 to maintain up-to-date information about two kinds of variables: the $level$ (of the local node with respect to every other node), and the largest $date$ that could have been carried until the local node from every other node (these dates account for the indirect views). The corresponding vector clock structure now associates to a given node both a $level$ and a $date$. Assume again for simplicity that all the local clocks share the same global time; this assumption can be easily removed, as explained at the end of the section. The detailed description of the algorithm is in Algorithm 2.

D. Correctness.

In the following, $level_{v,t}(u)$ and $date_{v,t}(u)$ denote the values of variables $vec[u].level$ and $vec[u].date$, respectively, read at node v at time t. The notation $getView_{v,t}(u)$ similarly stands for the result of the function $getView()$ called on v at time t with parameter u. Let us start with a few observations:

Property 1: Whenever an edge appears, the local vector is immediately sent on it (see $onAppearance()$).

Property 2: Whenever the local vector is modified, it is immediately sent to all the current neighbors.
This is true at the initialization; after that, the vector can only be modified in the function $updateVector()$, at the beginning of which a copy is made and at end of which the vector is sent, if different from the copy.

Lemma 3: For any two nodes u, v and time t, if the edge (v_1, v_2) is always present during a period $[t, t+\zeta)$, then $level_{v_2,t+\zeta}(u) \leq level_{v_1,t}(u) + 1$ for any u.

Proof: Two possibilities must be considered, depending on whether the edge (v_1, v_2) appeared before or after $level_{v_1}(u)$ took the value that it has at time t. If it appeared before, Property 1 guarantees that v_2 has received it by time $t+\zeta$; otherwise Property 2 guarantees the same. Every time a vector is received, the function $updateVector()$ is executed. This function goes through all entries of the received vector (among others) and guarantees the property by line 22. ■

Lemma 4: For any two nodes v_1, v_2 and time t, if the edge (v_1, v_2) is always present during a period $[t, t+\zeta)$, then $date_{v_2,t+\zeta}(u) \leq date_{v_1,t}(u) + 1$ for any u.

Proof: Same ideas as for Lemma 3, but considering line 24 instead of line 22. ■

The proof of the correctness of the algorithm will be carried out through a sequence of lemmas. Lemmas 5 and 6 are intermediate properties that are used for the proof of Lemma 7. Lemma 8 is similarly used for Lemma 9. The final theorem concludes based on Lemmas 7 and 9. Note that, in the following, the term "communication message" (or simply "message") does not refer to the messages generated by Algorithm 2; it denotes instead any message that could be exchanged by an application running in the network.

Lemma 5: No communication messages could be received by a node v at time t through a direct journey $\mathcal{J} \in \mathcal{J}^*_{(u,v)}$ unless $level_{v,t}(u) \leq |\mathcal{J}|_h$.

Proof: Let $\mathcal{J} = \{(e_1, t_1), (e_2, t_2) \ldots, (e_k, t_k)\} \in \mathcal{J}^*_{(u,v)}$ where $e_i = (v_i, v_{i+1})$, $1 \leq i \leq k+1$, with $v_1 = u$ and $v_{k+1} = v$. By Lemma 3, $level_{v_{i+1},t_i+\zeta}(u) \leq level_{v_i,t_i}(u) + 1$ $(1 \leq i \leq k)$; since the journey is direct, $t_i + \zeta = t_{i+1}$. Thus $level_{v,t_{k+1}}(u) = level_{v_{k+1},t_{k+1}}(u) \leq level_{v_1,t}(u) + k$. Since $v_1 = u$ and $level_{u,t_1}(u) = 0$ by definition, the Lemma holds. ■

Lemma 6: No communication message could be received by a node v at time t through an indirect journey $\mathcal{J} \in \mathcal{J}^*_{(u,v)}$ unless $date_{v,t}(u) \geq departure(\mathcal{J})$.

Proof: Let $\mathcal{J} = \{(e_1, t_1), (e_2, t_2) \ldots, (e_k, t_k)\} \in \mathcal{J}^*_{(u,v)}$ where $e_i = (v_i, v_{i+1})$, $1 \leq i \leq k+1$, with $v_1 = u$ and $v_{k+1} = v$. The proof follows a similar inspiration as that of Lemma 5, but requires an additional intermediate step because in general, $date_{u,departure(\mathcal{J})}(u) \neq departure(\mathcal{J})$. The intermediate step is as follows. Because $\mathcal{J}$ is indirect, then there exist at least one intermediate node v_j that has lost the edge from v_{j-1} before the appearance of the edge to v_{j+1} (that is, v_j is the last node such that $\mathcal{J}_{(u,v_j)} \subseteq \mathcal{J}$ is a direct journey). This has caused the function $updateVector()$ to execute on v_j, and thus v_j to convert its level w.r.t. u into a date (line 15), which by Lemma 5 is necessarily $\geq departure(\mathcal{J})$. Now, from Lemma 4 we have that $date_{v_{i+1},t_i+\zeta}(u) \geq date_{v_i,t_i}(u)$. By applying this inequality on the remaining edges (sequentially from v_j to v_{k+1}), we can conclude that $date_{v,t}(u) \geq departure(\mathcal{J})$. ■

Lemma 7: For any pair of nodes u, v and time t, $getView_{v,t}(u) \geq \phi_{v,t}(u)$.

Proof: By contradiction, let there exist a pair u, v and a time t such that $getView_{v,t}(u) < \phi_{v,t}(u)$. This means, by definition, that a message emitted at u at some time $t' \leq t$ could have arrived at v at some time t'' whereas $getView_{v,t''}(u) < t'$. From the way $getView()$ is computed, this implies that both $t'' - level_{v,t''}(u) \times \zeta < t'$ **and** that $date_{v,t''}(u) < t'$. Consider now the journey described by such a message. If the journey is direct, then the first inequality is contradicted by Lemma 5; if the journey is indirect, the second inequality is contradicted by Lemma 6. ■

Algorithm 2 Measuring temporal lags with lasting contacts.

```
1: $VectorClock\ vec \leftarrow \emptyset$
2: $Map<Node, VectorClock>\ neighborsVCs \leftarrow \emptyset$    ▷ the vector clocks of all neighbors are locally memorized.
3: initialization:
4: $vec[myself].level \leftarrow 0$
5: $send(vec)\ to\ N_{now()}$    ▷ sends the vector to all the current neighbors
6: onAppearance of a common edge with neighbor $ng$:
7: $send(vec)\ to\ ng$
8: onDisappearance of an edge to neighbor $ng$:
9: $neighborsVCs[ng] \leftarrow \emptyset$
10: $updateVector()$
11: onReception of a vector clock $vec_{ng}$ from a neighbor $ng$:
12: $neighborsVCs[ng] \leftarrow vec_{ng}$
13: $updateVector()$
14: updateVector():
15: $updateDatesBasedOnLevels()$
16: $VectorClock\ vec' \leftarrow vec$    ▷ copies the vector for subsequent change detection.
17: for all $n \in vec.nodes()$ do
18:     $vec[n].level \leftarrow +\infty$    ▷ resets all levels.
19: for all $vec_{ng} \in neighborsVCs$ do
20:     for all $n \in vec_{ng}.nodes()$ do    ▷ go across all entries of all neighbors vectors, and for each source node,
21:         if $vec_{ng}[n].level < vec[n].level - 1$ then
22:             $vec[n].level \leftarrow vec_{ng}[n].level + 1$    ▷ update the underlying level whenever a smaller level is detected.
23:         if $vec_{ng}[n].date > vec[n].date$ then
24:             $vec[n].date \leftarrow vec_{ng}[n].date$    ▷ update the underlying date whenever a larger date is detected.
25: if $vec \neq vec'$ then
26:     $send(vec)\ to\ N_{now()}$    ▷ if the vector has changed, send it to the current neighbors.
27: updateDatesBasedOnLevels():
28: for all $n \in vec.nodes()$ do
29:     if $now() - vec[n].level \times \zeta > vec[n].date$ then
30:         $vec[n].date \leftarrow now() - vec[n].level \times \zeta$
31: getView(Node $n$):
32: $return\ max(vec[n].date, now() - vec[n].level \times \zeta)$
```

Lemma 8: For any pair of nodes u, v and time t,

$$\phi_{v,t}(u) \geq now() - level_{v,t}(u) \times \zeta.$$

Proof: Let us examine separately the cases where $level_{v,t}(u)$ is $+\infty, 0$, or an integer $n > 0$.

1) If $level_{v,t}(u) = +\infty$, then $now() - level_{v,t}(u) \times \zeta = -\infty$, which is either equal to $\phi_{v,t}(u)$ when the latter is undefined (by convention), or less than it otherwise.
2) If $level_{v,t}(u) = 0$, then v is necessarily the same node as u (because levels w.r.t. other nodes can only be modified through *incrementing* the value of a neighbor). Consequently, $t - level_{v,t}(v) \times \zeta = t$, which is, by definition, the value of $\phi_{v,t}(v)$.
3) Let $level_{v,t}(u) = n$ for some integer $n > 0$. Let us first observe that this implies the existence of another node v' such that $level_{v',t-\zeta}(u) = n - 1$ and $\rho_{[t-\zeta,t]}(v, v') = 1$ (otherwise the local level would have been decreased by the function $updateVector()$ after the loss of the neighbor causing $level$ to be n). Knowing that $level_{u,t}(u) = 0$ implies, by simple induction, that there exists a direct journey $\mathcal{J} \in \mathcal{J}^*_{(u,v)}$ such that $arrival(\mathcal{J}) = t$ and $|\mathcal{J}|_h = level_{v,t}(u)$, which in turn implies that a message emitted at u at time $t - level_{v,t}(u) \times \zeta$ would be received at v at time t.

■

Lemma 9: For any pair of nodes u, v and time t, $getView_{v,t}(u) \leq \phi_{v,t}(u)$.

Proof: By contradiction, assume there exist a pair u, v and a time t such that $getView_{v,t}(u) > \phi_{v,t}(u)$. From the way function $getView()$ is defined, $getView_{v,t}(u) > \phi_{v,t}(u)$ implies that either $now() - level_{v,t}(u) \times \zeta > \phi_{v,t}(u)$ **or** $date_{v,t}(u) > \phi_{v,t}(u)$. The first inequality is contradicted by Lemma 8. As for the second, according to the algorithm, a $date$ variable can only be increased by means of two actions: copying the date variable from a neighbors' vector, or converting the current $level$ into a $date$. The first action cannot generate an unappropriate increase of the value because if two nodes are able to exchange their vectors, it also means that they could exchange the messages they have received so far (such a copy is necessarily consistent). The second action cannot generate a larger date than $\phi_{v,t}(u)$, otherwise this would again contradict Lemma 8. ■

The correctness of the algorithm now follows from Lemmas 7 and 9:

Theorem 10: For any pair of nodes u, v and time t,

$getView_{v,t}(u) = \phi_{v,t}(u)$.

Finally, let us remark that the simplifying assumption that all clocks share the same global value is not necessary for the correctness of the algorithm. Indeed, if the nodes keep track of *when* each $date$ was locally received, they can easily add information about how long the date was locally stored when they transmit it further. Combining this information with the propagation delay ζ allows to convert any received date into the local referential.

IV. Building foremost broadcast trees and network backbones

This section illustrates how temporal lags measurements can be leveraged to solve concrete problems in delay-tolerant networks, such as the construction of foremost broadcast trees and network backbones in networks whose schedule (presence function ρ) is periodic.

A. Definitions and Background

As discussed above, the length of a journey can be measured both in terms of hops or time. This gives rise to distinct definitions of *distance* in a graph $\mathcal{G}$:

- The *topological distance* from a node u to a node v at time t, noted $d_{u,t}(v)$, is defined as $Min\{|\mathcal{J}|_h : \mathcal{J} \in \mathcal{J}^*_{(u,v)} \wedge departure(\mathcal{J}) \geq t\}$. For a given date t, a journey whose departure is $t' \geq t$ and topological length is equal to $d_{u,t}(v)$ is qualified as *shortest* ;
- The *temporal distance* from u to v at time t, noted $\hat{d}_{u,t}(v)$ is defined as $Min\{arrival(\mathcal{J}) : \mathcal{J} \in \mathcal{J}^*_{(u,v)} \wedge departure(\mathcal{J}) \geq t\} - t$. Given a date t, a journey whose departure is after t and arrival is $t + \hat{d}_{u,t}(v)$ is qualified as *foremost*; one whose temporal length is $Min\{\hat{d}_{u,t'}(v) : t' \in \mathcal{T}_{[t,+\infty)}\}$ is qualified as *fastest*.

The problem of computing shortest, fastest, and foremost journeys in delay-tolerant networks was introduced in [3], and an algorithm for each of the three metrics was provided for the *centralized* version of the problem (assuming complete knowledge of $\mathcal{G}$). Computing optimal journeys is useful for many usages, the most obvious of which are routing and broadcasting. Consider the graph in Figure 3, with a propagation delay of $\zeta = 1$.

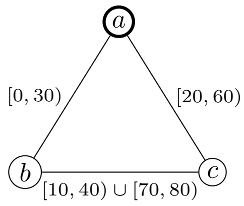


Figure 3. Another example of time-varying graph, with propagation delay $\zeta = 1$.

We may ask for example what are the foremost journeys that a message emitted at a should follow to reach every node as soon as possible. Clearly, this choice depends on the initial emission date (or *initiation* date), and even then, several possibilities could be available. A nice property of the "foremost" metric is that among all the possible journeys, there is always (at least) one whose prefixes are also foremost journeys.[3] This allows us to consider, for a given initiation date, a *tree* of foremost journeys that we refer to as a *foremost broadcast tree* (foremost BT, for short) for that date. The foremost BTs corresponding to the example of Figure 3 with emitter a are shown in Figure 4 as a function of the initiation date. Each tree indicates the route that a message should take to arrive at its destination the earliest.

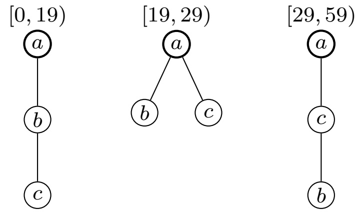


Figure 4. Foremost broadcast trees corresponding to the graph of Figure 3, for emitter a and initiation dates in $[0, 59)$.

The problems of designing *distributed* algorithms for building shortest, fastest, and foremost trees have recently been investigated in [4]. In the paper the authors study the feasibility of broadcast (with termination detection) with respect to the three different metrics depending on the assumptions on $\mathcal{G}$ or on the knowledge that the nodes have of $\mathcal{G}$. Three cases are considered: i) no assumptions are made on the schedule of edge appearances, ii) the edges are recurrent, that is, if they appear once, they appear infinitely often, and iii) this recurrence is bounded by a known duration. The feasibility and complexity with respect to the three metrics varies among these cases, but in none of them the computation of *reusable* foremost BTs can be achieved.

B. Periodically-varying graphs

We consider here the (distributed) problem of building foremost broadcast trees in periodically-varying graphs, that is, graphs whose schedule ρ is such that $\forall e \in E$ and $\forall t \in \mathcal{T}, \rho(e,t) = \rho(e, t + kp)$ for all integer k and a known duration p. The periodic assumption holds for example in networks whose entities have periodic movements (*e.g.,* satellites, subways, guards tour, or buses). Previous works in periodic DTNs include exploration by mobile agents [10], [12] and scalable routing [18], [23]. This assumption is also made in part of the work in [20].

The problem of building foremost BTs in periodic networks is made possible because the optimality of any given journey holds modulo p. Consider again the graph of Figure 3, and assume it to be periodic with period 100; in this case the first tree in Figure 4 is optimal for any initiation date in $[0, 19)$, as well as in $[100, 119)$, and $[900, 919)$, *etc.* (Note that the optimal tree for initiation dates in $[59, 100)$ would here be the same as the first one.) It is therefore possible to build foremost broadcast trees relatively to a complete period, and use them unchanged afterwards.

[3]This property may seem obvious, but it is not the case for the "fastest" metric (see related proofs in [3]).

Let us recall that the knowledge of a foremost BT can be used by the nodes to make optimal forwarding choices. For example, if node a wants to initiates a broadcast at time 25, it knows that it must forward the message to b and c, whereas if the initiation date is 80, it must only forward the message to b (as soon as the corresponding edge appears). It is important to keep in mind that the forwarding choices a node must make do *not* depend on the current date, but only on the date when the broadcast was initiated. Besides, a node does not have to know the entire set of trees to make the correct forwarding choices. It only needs to know what neighbors it should forward a message to, depending on the initial emitter and initiation date. The corresponding information in the case of our example relatively to emitter a is shown in table I.

on a	Initiation date	$[0, 19)$	$[19, 29)$	$[29, 59)$	$[59, 100)$
	Children	$\{b\}$	$\{b, c\}$	$\{c\}$	$\{b\}$
on b	Initiation date	$[0, 19)$	$[19, 59)$	$[59, 100)$	
	Children	$\{c\}$	$\emptyset$	$\{c\}$	
on c	Initiation date	$[0, 29)$	$[29, 59)$	$[59, 100)$	
	Children	$\emptyset$	$\{b\}$	$\emptyset$	

Table I
SET OF CHILDREN FOR THE EXAMPLE OF FIGURE 3 (RELATIVELY TO EMITTER a ONLY).

The algorithm we describe below generates such a table at every node, for any potential emitter.

C. The algorithm

1) High-level strategy: Tables of children like that of Table I are actually produced in two steps. Every node first determines its set of optimal *parents* in the trees, relatively to one complete period of initiation dates and all emitters (also called *sources*). This is done based on information on temporal lags provided by (an adaptation of) Algorithm 2. The resulting information is stored in a structure equivalent to Table II.

on b	Initiation date	$[0, 29)$	$[29, 59)$	$[59, 100)$
	Parent	a	c	a
on c	Initiation date	$[0, 19)$	$[19, 59)$	$[59, 100)$
	Parent	b	a	b

Table II
SET OF PARENTS FOR THE EXAMPLE OF FIGURE 3 (RELATIVELY TO EMITTER a ONLY).

Once a node has determined its set of parents with respect to all sources and initiation dates, it notifies them by sending them the corresponding intervals. Since the network is periodic, these notifications cannot, locally to a notifying node, last more than one period. On the parent side, the intervals are processed upon reception to fill in their children table. The way the intervals are sent by the children and processed by the parents is straightforward, and is therefore not described here. We focus instead on describing how the table of parents is built.

2) Using the concept of temporal view: There is a clear connexion between the problem of determining what parent is best to obtain a message in a foremost fashion, and the concept of temporal view discussed in Section III. In fact, the relation is direct: for a given source s and initiation date t, the parent that a node should select is precisely the first of its neighbors that can provide a temporal view $\phi(s) \geq t$. Interestingly, the algorithm introduced in Section III allows precisely to keep track of this type of information. The Foremost BT algorithm can then use Algorithm 2 as its main building block, by watching evolutions of the direct and indirect views, and memorizing the corresponding neighbors appropriately. This requires a few adaptations of Algorithm 2 discussed below.

3) Adaptation of the temporal lags measurement algorithm: The current version of Algorithm 2 does not *identify* the neighbors that are responsible of a current temporal view. Instead, only the *value* of the best "levels" and "dates" are determined in the loop of the routine $updateVector()$, reproduced below for convenience.

```
14: updateVector():
15: $updateDatesBasedOnLevels()$
16: $VectorClock\ vec' \leftarrow vec$
17: for all $n \in vec.nodes()$ do
18:     $vec[n].level \leftarrow +\infty$
19: for all $vec_{ng} \in neighborsVCs$ do
20:     for all $n \in vec_{ng}.nodes()$ do
21:         if $vec_{ng}[n].level < vec[n].level - 1$ then
22:             $vec[n].level \leftarrow vec_{ng}[n].level + 1$
23:         if $vec_{ng}[n].date > vec[n].date$ then
24:             $vec[n].date \leftarrow vec_{ng}[n].date$
25: if $vec \neq vec'$ then
26:     $send(vec)\ to\ N_{now()}$
```

Identifying these particular neighbors, or *proxies*, can however be done simply by adding and maintaining two additional variables in the vector clocks, that account respectively for the best "level proxy", and the best "date proxy". These variables can be updated by overwriting their value whenever lines 22 or 24 are executed, respectively.

The relation between both algorithms then follows an *observer-observable* pattern as shown on Figure 5. Initially, the Foremost BT algorithm makes itself known as an observer through calling a $register()$ function on Algo 2. This latter notifies in turn the Foremost BT algorithm whenever a change in level or an improvement in date are detected (to which events the FBT algorithm can associate operations to memorize the corresponding neighbors).

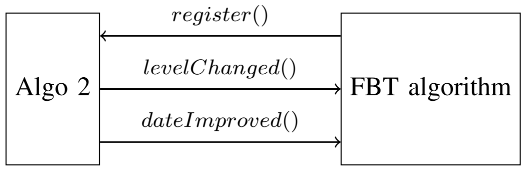


Figure 5. Interactions between Algorithm 2 and the Foremost BT algorithm.

These notifications are raised by Algo 2 after the loop in $updateVector()$, if the vector has changed (e.g., next to the $send$ operation of line 26), as follows:

- $levelChanged$(Node src, Integer $level$, Node $proxy$), called for each source whose level value or level proxy has changed during the loop.
- $dateImproved$(Node src, Integer $date$, Node $proxy$), called for each source whose date value has increased during the loop, if this date is larger than the direct view (that is, larger than $now() - level \times \zeta$ for the corresponding $level$).

In order to allow the Foremost BT algorithm to "plug" on top of an already running instance of Algorithm 2, we additionally assume that:

Property 11: The call to $register()$ causes Algo 2 to call immediately $levelChanged()$ on the Foremost BT algorithm for every source with respect to which the current level is not $+\infty$.

4) The Foremost BT algorithm: The detailed process of determining the parents is described in Algorithm 3. Its basic principle consists in monitoring the evolution of the temporal views based either on direct or indirect journeys, and record the corresponding neighbors as parents as follows. Whenever the temporal view is improved by means of an indirect journey ($dateImproved()$), the corresponding neighbor is associated with the provided date. More precisely, it is stored in a table that associates it with this date, which date corresponds in fact to the *end* of the interval this parent covers (the beginning of the interval being the end of the previous one, circularly). This strategy relies on the fact that if a node can provide a message initiated at t, then it can also provide any message initiated before t. As for the improvement of the temporal view by means of direct journeys, the algorithm maintains a dedicated variable to memorize the current levels (table $level$) and the corresponding neighbors (table $directProxy$). Whenever a notification occurs, whether related to a change in date or level, this variable can be used in $updateRecord()$ to determine what largest initiation date t could have already been covered by these direct journeys. If the date constitute an improvement, it is stored in the parent table together with the corresponding neighbor.

A few additional remarks:

- In some cases, the algorithm may lead to record consecutively the same parent; in such case, the intervals can simply be merged.
- Last but not least, the process can be started independently on each node, as long as Algorithm 2 is assumed to have already run on *all* the nodes for at least a duration of $max(|\mathcal{J}|_t : \mathcal{J} \in \mathcal{J}^*_{\mathcal{G}})$ – the *temporal diameter* of the network. This is to ensure that all the nodes know their respective temporal views.

5) Correctness: The correctness of Algorithm 3 essentially follows from periodicity and the correctness of Algorithm 2.

Theorem 12: The execution of Algorithm 3 in a periodically-varying graph $\mathcal{G}$ with known period p results in all nodes selecting the correct set of parents with respect to all Foremost BTs in $O(p)$ time.

Algorithm 3 Building Foremost BTs (determining the parents)

```
1: $Map <Node, <Date, Node>> parents \leftarrow \emptyset$
2: $Map <Node, Node> directProxy \leftarrow nil$
3: $Map <Node, Integer> level \leftarrow nil$
4: $Date\ startD \leftarrow nil$
5: init():
6: $startD \leftarrow now()$
7: $register\ to\ Algorithm\ 2$
8: dateImproved(Node $src$, Integer $date$, Node $proxy$):
9: $updateRecord(src)$
10: $parents[src].add(date, proxy)$
11: levelChanged(Node $src$, Integer $level$, Node $proxy$):
12: $updateRecord(src)$
13: if $level \neq +\infty$ then
14:     $directProxy[src] \leftarrow proxy$
15: $level[src] \leftarrow level$        ▷ keeps a local copy of the levels.
16: currentDirectView(Node $src$):
17: $return\ now() - level[src] \times \zeta$        ▷ based on the local copy.
18: updateRecord(Node $src$):
19: if $directProxy[src] \neq nil$ then
20:     if $currentDirectView() > parents[src].lastDate()$ then
21:         $parents[src].add(currentDirectView(),$
                $directProxy[src])$
22: when $now() == startD + p$:
23: $terminate.$
```

Sketch of the proof: The idea is to prove that the recorded parent for any initiation date t and source s is indeed (any of) the first neighbor to provide a temporal view of s that is greater or equal to t. For any t, this view is either direct or indirect. The information relative to the direct view – level and corresponding neighbor – is locally stored through lines 14 and 15 whenever it changes (as per Algorithm 2). This allows $currentDirectView()$ to indicate, at any time instant (thanks to Property 11), the corresponding direct view. As for the indirect view, the desired property follows from the way parents are recorded in the parents table: when an higher indirect view is provided ($dateImproved()$), first the neighbor responsible for the current direct view is stored for all initiation dates that it has already been able to cover ($updateRecord()$), then the one responsible for the new indirect view is stored in turn. The same update operation is executed when the level changes, but instead of being stored in turn, the new level proxy replaces the previous one. Due to the periodicity, the algorithm necessarily terminates in $O(p)$ time (in fact, in exactly one period p) because the first parent reappears and can deliver the same initiation dates as before, plus p. ■

D. Example traces

The tables below show some execution traces based on the example graph of Figure 3 (with respect to emitter a only). These traces consider that c starts at time 50 (modulo 100), and b at time 60 (modulo 100). These dates are chosen to reflect a variety of initial conditions and behaviors. The traces include the list of dated notifications and modifications of the parents table for both nodes (Tables III and IV), and the resulting tables of parents, using both their original and interval-based representations (Tables V and VI).

Date	Event	Parents table for source a
65	–	
71	$dateImproved(a, 59, c)$	$parents[a].add(59, c)$
1	$levelChanged(a, 1, a)$	
30	$levelChanged(a, 2, c)$	$parents[a].add(29, a)$
40	$levelChanged(a, +\infty, nil)$	$parents[a].add(38, c)$
165	$end\ of\ the\ period$	

Table III
RELEVANT TRACES WITH RESPECT TO NODE b AND EMITTER a.

Date	Event	Parents table for source a
50	$levelChanged(a, 1, a)$	
60	$levelChanged(a, +\infty, nil)$	$parents[a].add(59, a)$
11	$levelChanged(a, 2, b)$	
21	$levelChanged(a, 1, a)$	$parents[a].add(19, b)$
50	$end\ of\ the\ period$	

Table IV
RELEVANT TRACES WITH RESPECT TO NODE c AND EMITTER a.

Initiation date	$\rightarrow 59$	$\rightarrow 29$	$\rightarrow 38$	
Parent	c	a	c	
Initiation date	$[0, 29)$	$[29, 38)$	$[38, 59)$	$[59, 100)$
Parent	a	c	c	a

Table V
RESULTING TABLE OF PARENTS FOR NODE b.

Initiation date	$\rightarrow 59$	$\rightarrow 19$	
Parent	a	b	
Initiation date	$[0, 19)$	$[19, 59)$	$[59, 100)$
Parent	b	a	b

Table VI
RESULTING TABLE OF PARENTS FOR NODE c.

E. Network Backbone Construction

The authors of [20] define the concept of *essentiality* of a contact, with respect to a time t, as the fact that this contact contributes to reduce the information latency between (at least) one pair of nodes in the network with respect to a piece of information originated at time t. Based on this concept, they define the network *backbone* for time t, noted $\mathcal{H}_t$, as the set of all the essential edges w.r.t. time t. Interestingly, this backbone corresponds precisely to the exact union of all foremost broadcast trees for initiation date t. Consequently, our algorithm computes at once, as a by-product, the backbones for all possible initiation dates. Further, the backbones so-computed are generalizations of those of [20] in that the context does not assume instantaneous contacts nor instantaneous propagation delays on edges.

V. CONCLUDING REMARKS AND OPEN PROBLEMS

This paper addressed the general question of measuring temporal lags in highly-dynamic networks. A centralized version of this problem was recently formulated in the field of social networks, and solved for the particular case where contacts between entities are assumed to have instantaneous durations and propagation delays. We formulated this problem in a distributed setting, and solved it for the more general case of contacts that have arbitrary durations and can overlap with each other, as well as non-instantaneous propagation delays. The second part of the paper illustrated how temporal lags could be use to solve more elaborate network problems, such as the construction of foremost broadcast trees in periodically-varying graphs. Interestingly, the provided solution also solved, as a by-product, the construction of time-dependent backbones, a concept recently formulated in [20].

The feasibility of measuring temporal lags was demonstrated through an algorithm that adapts vector clocks to the task of measuring temporal lags in the general context of contacts with duration – when both direct and indirect journeys can co-exist in the network. The complexity of the algorithm – in its current version – is however high, and implementing it in a practical scenario definitely requires further optimization (same for the distributed version of [20] for instantaneous contacts). Improving the communication costs of this algorithm is an interesting research avenue. It is especially relevant if we consider that the measurement of time lags can serve as a building block for other tasks, as it was the case here. Indeed, any improvement in this regard would have instant repercussions – new upper bounds, typically – on *all* derivative problems and algorithms.

The type of problems and algorithms discussed here, as well as a number of currently investigated problem on highly-dynamic networks, raise interesting analytic questions on the analysis of solutions. Contrarily to most distributed algorithms for static networks, the complexity of an algorithm in this new context does not only depends on the number of nodes and edges, but is strongly dependent on the number of *topological events* during the execution (in fact, a vast majority of communications and computations in this paper algorithms are precisely triggered by topological events). It would therefore be interesting to study the impact of the network schedule (e.g., through mobility models) on the complexity.

Finally, we believe that in the same way as temporal lags were used here to build foremost broadcast trees, they could be used to solve a number of other problems, in particular for periodically-varying DTNs. Example of these open problems include building *fastest* broadcast trees – trees that minimize the time between the first and last exchange of message – or electing a leader based on its temporal eccentricity – the speed at which it can reach the other nodes.

ACKNOWLEDGMENTS

This work has been supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Australian Reerach Council (ARC), and Dr. Flocchini's University Research Chair.

References

[1] K. Berman, “Vulnerability of scheduled networks and a generalization of Menger’s Theorem,” *Networks*, vol. 28, no. 3, pp. 125–134, 1996.

[2] S. Bhadra and A. Ferreira, “Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks,” in *Proc. 2nd Intl. Conference on Ad Hoc, Mobile and Wireless Networks (AdHoc-Now)*, 2003, pp. 259–270.

[3] B. Bui-Xuan, A. Ferreira, and A. Jarry, “Computing shortest, fastest, and foremost journeys in dynamic networks,” *Intl. J. of Foundations of Comp. Science*, vol. 14, no. 2, pp. 267–285, April 2003.

[4] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro, “Deterministic computations in time-varying graphs: Broadcasting under unstructured mobility,” in *Proc. 5th IFIP Conference on Theoretical Computer Science (TCS)*, 2010, pp. 111–124.

[5] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, “Time-Varying Graphs and Dynamic Networks,” *Arxiv preprint arXiv:1012.0009*, 2010.

[6] A. Chaintreau, A. Mtibaa, L. Massoulie, and C. Diot, “The diameter of opportunistic mobile networks,” *Communications Surveys & Tutorials*, vol. 10, no. 3, pp. 74–88, 2008.

[7] H. Dubois-Ferriere, M. Grossglauser, and M. Vetterli, “Age matters: efficient route discovery in mobile ad hoc networks using encounter ages,” in *Proceedings ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, 2003, p. 266.

[8] A. Ferreira, “Building a reference combinatorial model for MANETs,” *IEEE Network*, vol. 18, no. 5, pp. 24–29, 2004.

[9] C. Fidge, “Timestamps in message-passing systems that preserve the partial ordering,” in *Proceedings of the 11th Australian Computer Science Conference*, vol. 10, no. 1, 1988, pp. 56–66.

[10] P. Flocchini, M. Kellett, P. Mason, and N. Santoro, “Mapping an unfriendly subway system,” in *Proc. 5th International Conference on Fun with Algorithms*, 2010, pp. 190–201.

[11] P. Flocchini, B. Mans, and N. Santoro, “Sense of direction: Definitions, properties, and classes,” *Networks*, vol. 32, no. 3, pp. 165–180, 1998.

[12] ——, “Exploration of periodically varying graphs,” in *Proc. 20th Intl. Symposium on Algorithms and Computation (ISAAC)*, 2009, pp. 534–543.

[13] M. Grossglauser and M. Vetterli, “Locating nodes with EASE: Last encounter routing in ad hoc networks through mobility diffusion,” in *Proc. 22nd Conference on Computer Communications (INFOCOM)*, vol. 3, 2003, pp. 1954–1964.

[14] P. Holme, “Network reachability of real-world contact sequences,” *Physical Review E*, vol. 71, no. 4, p. 46119, 2005.

[15] S. Jain, K. Fall, and R. Patra, “Routing in a delay tolerant network,” in *Proc. Conf. on Applications, Technologies, Architectures, and Protocols for Comp. Comm. (SIGCOMM)*, 2004, pp. 145–158.

[16] E. Jones, L. Li, J. Schmidtke, and P. Ward, “Practical routing in delay-tolerant networks,” *IEEE Transactions on Mobile Computing*, vol. 6, no. 8, pp. 943–959, 2007.

[17] D. Kempe, J. Kleinberg, and A. Kumar, “Connectivity and inference problems for temporal networks,” in *Proceedings 32nd ACM Symposium on Theory of Computing*, 2000, p. 513.

[18] A. Keränen and J. Ott, “DTN over aerial carriers,” in *Proceedings 4th ACM Workshop on Challenged Networks*, 2009, pp. 67–76.

[19] A. Khelil, P. Marron, and K. Rothermel, “Contact-based mobility metrics for delay-tolerant ad hoc networking,” in *Proceedings 13th IEEE Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2005, pp. 435–444.

[20] G. Kossinets, J. Kleinberg, and D. Watts, “The structure of information pathways in a social communication network,” in *Proc. 14th Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, 2008, pp. 435–443.

[21] V. Kostakos, “Temporal graphs,” *Physica A*, vol. 388, no. 6, pp. 1007–1023, 2009.

[22] A. Lindgren, A. Doria, and O. Schelén, “Probabilistic routing in intermittently connected networks,” *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, no. 3, pp. 19–20, 2003. [Online]. Available: papers/LDS03.pdf

[23] C. Liu and J. Wu, “Scalable routing in cyclic mobile networks,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 9, pp. 1325–1338, 2009.

[24] F. Mattern, “Virtual time and global states of distributed systems,” in *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, 1989, pp. 215–226.

[25] J. Tang, M. Musolesi, C. Mascolo, and V. Latora, “Characterising temporal distance and reachability in mobile and online social networks,” *ACM Computer Communication Review*, vol. 40, no. 1, pp. 118–124, 2010.

[26] M. Yamashita and T. Kameda, “Computing on anonymous networks: Part I and II,” *IEEE Trans. on Par. and Distributed Systems*, vol. 7, no. 1, pp. 69 – 96, 1996.

[27] Z. Zhang, “Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges,” *IEEE Communications Surveys & Tutorials*, vol. 8, no. 1, pp. 24–37, 2006.