Shortest, Fastest, and Foremost Broadcast in Dynamic Networks [*]

Arnaud Casteigts[1], Paola Flocchini[2], Bernard Mans[3], and Nicola Santoro[4]

[1]University of Bordeaux, France,
[2]University of Ottawa, Canada,
[3] Macquarie University, Sydney, Australia,
[4] Carleton University, Ottawa, Canada,

Highly dynamic networks rarely offer end-to-end connectivity at a given time. Connectivity in these networks can be established over time and space, based on temporal analogues of multi-hop paths (also called *journeys*). In a seminal work, Bui-Xuan, Ferreira, and Jarry (2003) proposed a collection of centralized algorithms to compute three types of optimal journeys in these networks – namely, shortest (min hop), fastest (min duration), and foremost (earliest arrival) journeys – given full prediction of the network evolution. We study the *distributed* counterparts of these problems, i.e. shortest, fastest, and foremost broadcast with termination detection (TDB), with minimal knowledge on the topology. We show that the feasibility of each variant requires distinct features on the evolution, through identifying three classes of dynamic graphs wherein the variants become gradually feasible : graphs in which the re-appearance of edges is *recurrent* (class $\mathcal{R}$), *bounded-recurrent* ($\mathcal{B}$), or *periodic* ($\mathcal{P}$), together with specific knowledge that are respectively n (the number of nodes), Δ (a bound on the recurrence time), and p (the period). To raise ambiguity, we *do not* require that all pairs of nodes get in contact – only a subset of them that makes the *footprint* of the graph connected.

Our results, together with the strict inclusion between $\mathcal{P}$, $\mathcal{B}$, and $\mathcal{R}$, implies a feasibility order among the three variants of the problem, i.e. TDB$[foremost]$ requires weaker assumptions on the topology dynamics than TDB$[shortest]$, which itself requires less than TDB$[fastest]$. Reversely, these differences in feasibility imply that the computational powers of $\mathcal{R}_n$, $\mathcal{B}_\Delta$, and $\mathcal{P}_p$ also form a strict hierarchy.

Keywords: dynamic networks, distributed algorithm, time-varying graphs, delay-tolerant broadcast, recurrent edges.

1. Introduction

Dynamic networks are widely addressed in distributed computing. Contexts of interest are as varied as fault-tolerance, interaction scheduling, dynamic membership, planned mobility, or unpredictable mobility. The recent emergence of scenarios where entities are truly mobile and can communicate without infrastructure (e.g. vehicles, satellites, robots, or pedestrian smartphones) brought to the fore the most versatile of these environments. In these *highly* dynamic networks, changes are not anomalies but rather integral part of the nature of the system.

The need to categorize and understand highly dynamic networks led the engineering community to design a variety of *mobility models*, each of which captures a particular context by means of rules that determine how the nodes move and communicate (see e.g. [23]).

[*]Preliminary results were presented at the 6th IFIP International Conference on Theoretical Computer Science. This work has been supported in part by NSERC (Canada) and Dr. Flocchini's University Research Chair.

A popular example includes the well-known *random waypoint* model [5]. The main interest of these models is to be able to reproduce experiments and compare different solutions on a relatively fair basis, thereby providing a common ground for the engineering community to solve practical challenges in highly dynamic networks, e.g. routing and broadcasting [8, 21, 25, 26, 28, 31, 32].

In the same way as mobility models enable to federate practical investigations in highly dynamic networks, *logical properties* on the graph dynamics, that is, *classes of dynamic graphs*, have the potential to guide a more formal exploration of their analytical aspects. A number of special classes were recently identified, for example graphs in which the nodes interact infinitely often (*e.g.,* uniform random scheduler for *population protocols* [1, 2, 12]); graphs whose dynamics is unrestricted but remains connected at any instant [29, 16]; graphs in which there exists a stable connected spanning subgraph in any T-time window (*a.k.a. T-interval connectivity*) [27, 22]; graphs whose edges appear or disappear with given probabilities [13, 4, 14, 30]; graphs that have a stable root component [6]; graphs whose schedule is periodic [10, 18, 19, 20, 24] or guarantees minimal reachability properties [9]. These classes (among others) were organized into a hierarchy in [11].

In this paper we are interested in studying specific relationship between some of these classes, namely three subclasses of those networks called *delay-tolerant networks* (DTNs), in which instant connectivity is never guaranteed, but still connectivity can be achieved over time and space (see e.g. [3]). These classes are:

- Class $\mathcal{R}$ of all graphs whose edges cannot disappear forever (*recurrent edges*). That is, if an edge appears once and disappears, then it will eventually re-appear at some unknown (but finite) date. It is not required that all pairs of nodes share an edge, but only that the *footprint* of all edges forms a connected graph (otherwise, even temporal connectivity is not guaranteed). This class corresponds to Class 6 in [11].
- Class $\mathcal{B}$ (for *bounded-recurrent* edges) consisting of those graphs with recurrent edges in which the recurrence time cannot exceed a given duration Δ. And again, the footprint is connected. This class corresponds to Class 7 in [11].
- Class $\mathcal{P}$ (for *periodic* edges) consisting of those graphs in which all topological events (appearance or disappearance) repeat identically modulo some period p. And again, the footprint is connected. This class corresponds to Class 8 in [11].

As far as *inclusion* is concerned, it clearly holds that $\mathcal{P} \subset \mathcal{B} \subset \mathcal{R}$, but what about the *computational relationship* between these classes? Considering different types of knowledge, namely the number n of nodes in the network, a bound Δ on the recurrence time, and (any multiple of) the period p, we look at the relationship between $\mathscr{P}(\mathcal{R}_n)$, $\mathscr{P}(\mathcal{B}_\Delta)$, and $\mathscr{P}(\mathcal{P}_p)$, where $\mathscr{P}(\mathcal{C}_k)$ is the set of problems one can solve in class $\mathcal{C}$ with knowledge k.

The investigation is carried out by studying a fundamental problem in distributed computing: *broadcast* with termination detection at the emitter (or TDB). This problem can have at least three distinct definitions in highly dynamic networks: TDB$[foremost]$, in which the date of delivery is minimized at every node; TDB$[shortest]$, where the number of hops used by the broadcast is minimized relative to every node; and $\text{TDB}[fastest]$, where the overall duration of the broadcast is minimized (however late the departure be). These three metrics were considered in [7] as part of an offline problem where, given a complete schedule of the network, one has to compute all shortest, fastest, and foremost journeys (temporal paths) from a given node.

Our contribution

In this paper we examine the feasibility and reusability of the solution (and to some extent, the complexity) of $\text{TDB}[foremost]$, $\text{TDB}[shortest]$, and $\text{TDB}[fastest]$ in $\mathcal{R}$ and $\mathcal{B}$ with knowledge $\emptyset$, n, or Δ. We additionally draw some observations from external results in $\mathcal{P}$ with knowledge p [10], that complete our general picture of feasibility and reusability of broadcast in the three classes. Here is a short summary of the main results.

Feasibility: We first show that none of these problems are solvable in any of the classes unless additional knowledge is considered. We prove constructively that knowing n makes it possible to solve $\text{TDB}[foremost]$ in $\mathcal{R}$, but this is not sufficient to solve $\text{TDB}[shortest]$ nor $\text{TDB}[fastest]$, even in $\mathcal{B}$. $\text{TDB}[shortest]$ becomes in turn feasible in $\mathcal{B}$ if Δ is known, but this context is not sufficient to solve $\text{TDB}[fastest]$; this later problem being solvable in $\mathcal{P}$ knowing p [10]. Put together, these results allow us to show that

$$\mathscr{P}(\mathcal{R}_n) \subsetneq \mathscr{P}(\mathcal{B}_\Delta) \subsetneq \mathscr{P}(\mathcal{P}_p) \tag{1}$$

that is, the computational relationships between these three contexts form a *strict* hierarchy.

Let us define a binary relation $\preceq_{feasibility}$ over problems, such that for two problems P_1 and P_2, $P_1 \preceq_{feasibility} P_2$ implies that for some class of TVG C and knowledge k we have $P_1 \in \mathscr{P}(C_k)$ while $P_2 \notin \mathscr{P}(C_k)$. This relation induces a partial order on the *feasibility* of problems with respect to topological changes. Our results imply that

$$\text{TDB}[foremost] \preceq_{feasibility} \text{TDB}[shortest] \preceq_{feasibility} \text{TDB}[fastest] \tag{2}$$

Reusability: Regarding the possibility to reuse a solution, that is, a same broadcast tree, over several broadcasts, we find the intriguing fact that reusability in $\text{TDB}[shortest]$ is easier than that of $\text{TDB}[foremost]$. Precisely, when $\text{TDB}[shortest]$ becomes feasible in $\mathcal{B}$, it enables at once reusability of the broadcast trees, whereas $\text{TDB}[foremost]$, although it was already feasible in $\mathcal{R}$, does not enable reusability until in $\mathcal{P}$ [10]. Using a similar definition as for feasibility, we consider the relation $\preceq_{reusability}$ that induces a partial order over problems with respect to the reusability of a solution. Our results imply that

$$\text{TDB}[shortest] \preceq_{reusability} \text{TDB}[foremost] \tag{3}$$

Whether reusability is more or less difficult in $\text{TDB}[fastest]$ than in $\text{TDB}[foremost]$ is an open question, both of them being impossible in $\mathcal{B}_\Delta$ and possible in $\mathcal{P}_p$. Our results on feasibility and reusability are summarized in Table 1.

Metric	Class	Knowledge	Feasibility	Reusability	Result from
Foremost	$\mathcal{R}$	$\emptyset$	no	–	this paper
	$\mathcal{R}$	n	yes	no	this paper
	$\mathcal{B}$	Δ	yes	no	this paper
	$\mathcal{P}$	p	yes	yes	[10]
Shortest	$\mathcal{R}$	$\emptyset$	no	–	this paper
	$\mathcal{R}$	n	no	–	this paper
	$\mathcal{B}$	Δ	yes	yes	this paper
	$\mathcal{P}$	p	yes	yes	(same result)
Fastest	$\mathcal{R}$	$\emptyset$	no	–	this paper
	$\mathcal{R}$	n	no	–	this paper
	$\mathcal{B}$	Δ	no	–	this paper
	$\mathcal{P}$	p	yes	yes	[10]

Table 1. Feasibility and reusability of TDB in different classes of dynamic networks (with associated knowledge).

Metric	Class	Knowl.	Time	Info. msgs (1^{st} run)	Control msgs (1^{st} run)	Info. msgs (next runs)	Control msgs (next runs)
Foremost	$\mathcal{R}$	n	unbounded	$O(m)$	$O(n^2)$	$O(m)$	$O(n)$
	$\mathcal{B}$	n	$O(n\Delta)$	$O(m)$	$O(n^2)$	$O(m)$	$O(n)$
		Δ	$O(n\Delta)$	$O(m)$	$O(n)$	$O(m)$	0
		$n\&\Delta$	$O(n\Delta)^*$	$O(m)$	0	$O(m)$	0
Shortest	$\mathcal{B}$	Δ	$O(n\Delta)$	$O(m)$	$O(n) : 2n-2$	$O(n)$	0
either of $\{$		$n\&\Delta$	$O(n\Delta)$	$O(m)$	$O(n) : n-1$	$O(n)$	0
		$n\&\Delta$	$O(n\Delta)^*$	$O(m)$	0	$O(m)$	0

Table 2. Complexity of TDB for different classes of dynamic networks and associated knowledge. *(The * indicates that the emitter terminates implicitely, even in the first run.)*

Complexity: Although complexity is not the main focus here, we characterize the time complexity and message complexity of our algorithms and observe some interesting facts. For instance, the message complexity of our algorithm to TDB$[foremost]$ is lower knowing Δ than knowing n, and even lower if both are known. These results are summarized in Table 2. Note that TDB involves two processes: the actual dissemination of *information messages*, and the exchange of typically smaller *control messages* (e.g. for termination detection), both of which are separately analyzed. Regarding time complexity, we observe that for all our algorithms (except those which terminate implicitly), the termination detection phase takes the same order of time as the dissemination phase. Thus, Table 2 does not distinguish both phases.

2. Model and Basic Properties

2.1. *Definitions and Terminology*

Consider a system composed of a finite set of n entities V (or nodes) that interact with each other over a (possibly infinite) time span $\mathcal{T} \subseteq \mathbb{T}$ called *lifetime* of the system, where $\mathbb{T}$ is the temporal domain (typically, $\mathbb{N}$ or $\mathbb{R}^+$ for discrete and continuous-time systems, respectively). In this paper we consider a continuous-time setting with $\mathbb{T} = \mathbb{R}^+$.

Following [11], we describe the network as a *time-varying graph* (*TVG*, for short) $\mathcal{G} = (V, E, \mathcal{T}, \rho, \zeta)$, where $E \subseteq V \times V$ is the set of m *intermittently available* undirected edges such that $(u, v) \in E \Leftrightarrow u$ and v have at least one contact over $\mathcal{T}$, $\rho : E \times \mathcal{T} \to \{0, 1\}$ (*presence function*) indicates whether a given edge is *present* at a given time, and $\zeta : E \times \mathcal{T} \to \mathbb{T}$ (*latency function*), indicates the time it takes to cross a given edge (*i.e.,* send a message) if starting at a given time. In this paper, we assume ζ to be constant over all edges and dates, and call it the *crossing delay*. Thus we use the shorthand notations $\mathcal{G} = (V, E, \mathcal{T}, \rho)$. Finally, the (static) graph formed by the first two elements of $\mathcal{G}$, that is V and E is the *footprint* of $\mathcal{G}$ (also called *underlying graph* or *interaction graph*).

This formalism essentially encompasses that of *evolving graphs* [17], where $\mathcal{G}$ is represented as a sequence of graphs $G_1, G_2, ..., G_i, ...$ each providing a *snapshot* of the system at different times (which correspond either to discrete steps or topological events). In comparison, TVGs offer an *interaction-centric* view of the network evolution, where the evolution of each edge can be considered irrespective of the global time sequence, which turns out to be convenient when dealing with *locally* specified properties.

A given edge $e \in E$ is said to be *recurrent* if it appears infinitely often; that is, for any date t, $\rho(e, t) = 0 \implies \exists t' > t : \rho(e, t') = 1$. When all the edges of E are recurrent, we say that $\mathcal{G}$ is *recurrent* (keep in mind that in general $E \neq V^2$). Let $\mathcal{R}$ denote the class of recurrent TVGs whose footprint $G = (V, E)$ is connected. The recurrence of an edge e is said to be *time-bounded* (or simply *bounded*), if there exists a constant Δ_e such that the time between any two successive appearances of e is at most Δ_e. When the recurrence of all the edges of a graph $\mathcal{G} \in \mathcal{R}$ is time-bounded, we say that $\mathcal{G}$ is *time-bounded recurrent*, call $\Delta = \max\{\Delta_e : e \in E\}$, and denote by $\mathcal{B} \subset \mathcal{R}$ the class of time-bounded recurrent TVGs. An edge $e \in E$ is said to be *periodic* of period p_e if $\rho(e, t) = \rho(e, t + kp_e)$ for all integer k. A graph is said periodic if all its edges are periodic, and its *period* p is the least common multiple of all the edges periods. We denote by $\mathcal{P} \subset \mathcal{B}$ the class of periodic TVGs.

Given a TVG $\mathcal{G} = (V, E, \mathcal{T}, \rho)$, we consider that $G = (V, E)$ is always simple (no self-loop nor multiple edges) and that nodes possess unique identifiers.

The set of edges being incident to a node u at time t is noted $I_t(u)$ (or simply I_t, when the node is implicit). Finally, we note $\mathcal{G}_{[t_a, t_b)}$ the *temporal* subgraph of a TVG $\mathcal{G}$ restricted to lifetime $[t_a, t_b)$.

When an edge $e = (x, y)$ appears, the entities x and y can communicate. The time ζ necessary to transmit a message (*crossing delay*) is known to the nodes. The minimal duration of edge presence is assumed to be $2 \times \zeta$ (*i.e.,* long enough for a back and forth exchange of message). Algorithmically, this allows the following observations:

1. If a message is sent just after an edge has appeared, the message and a potential answer are guaranteed to be successfully transmitted.
2. If the recurrence of an edge is bounded by some Δ, then this edge cannot disappear for more than $\Delta - 2 \times \zeta$.

The appearance and disappearance of edges are instantly detected by the two adjacent nodes (they are notified of such an event without delay). If a message is sent less than ζ before the disappearance of an edge, it is lost. However, since the disappearance of an edge is detected instantaneously, and the crossing delay ζ is known, the sending node can locally determine whether the message was successfully delivered. We thus authorize the special primitive $send\_retry$ as a facility to specify that if the message is lost, then it is automatically re-sent upon next appearance of the edge, and this sending is necessarily successful (Property 1). Note that nothing precludes this primitive to be called while the corresponding edge is even absent (this actually simplifies the expression of some algorithms).

A sequence of couples $\mathcal{J} = \{(e_1, t_1), (e_2, t_2), ..., (e_k, t_k)\}$, with $e_i \in E$ and $t_i \in \mathcal{T}$ for all i, is called a *journey* in $\mathcal{G}$ iff $\{e_1, e_2, ...\}$ is a walk in G and for all t_i, $\rho(e_i)_{[t_i, t_i+\zeta)} = 1$ and $t_{i+1} \geq t_i + \zeta$. We denote by $departure(\mathcal{J})$, and $arrival(\mathcal{J})$, the starting date t_1 and last date $t_k + \zeta$ of $\mathcal{J}$, respectively.

Journeys can be thought of as *paths over time* from a source node to a destination node (if the journey is finite). Let us denote by $\mathcal{J}^*_\mathcal{G}$ the set of all finite journeys in a graph $\mathcal{G}$. We will say that node u can reach node v in $\mathcal{G}$, and note $\exists \mathcal{J}_{(u,v)} \in \mathcal{J}^*_\mathcal{G}$ (or simply $u \rightsquigarrow v$, if $\mathcal{G}$ is clear from the context), if there exists at least one possible journey from u to v in $\mathcal{G}$. Note that the notion of journey is asymmetrical ($u \rightsquigarrow v \not\Leftrightarrow v \rightsquigarrow u$), regardless of whether edges are directed or undirected.

Because journeys take place *over time*, they have both a topological length and a temporal length. The *topological length* of $\mathcal{J}$ is the number $k = |\mathcal{J}|_h$ of couples in $\mathcal{J}$ (i.e., number of *hops*), and its *temporal length* is its duration $|\mathcal{J}|_t = arrival(\mathcal{J}) - departure(\mathcal{J}) = t_k - t_1 + \zeta$. This yields to two 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 called *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 $t' \geq t$ and arrival is $t + \hat{d}_{u,t}(v)$ is called *foremost*, and one whose temporal length is $Min\{\hat{d}_{u,t'}(v) : t' \in \mathcal{T}\}$ is called *fastest*.

Informally, a *foremost* journey is one that minimizes the date of arrival at destination; a *shortest* journey is one that uses the least number of hops; and a *fastest* journey is one that minimizes the time spent between departure and arrival (however late the departure be) [7].

2.2. *Problems*

The problem of computing shortest, fastest, and foremost journeys in delay-tolerant networks was solved in [7] as a centralized (*i.e.,* combinatorics) problem, given complete knowledge of $\mathcal{G}$. We consider a *distributed* variant of the problem, namely performing *broadcast with termination detection* at the emitter, or TDB, according to either one of the three metrics.

TDB in general requires all nodes to receive a message with some information initially held by a single node x, called *source* or *emitter*, and the source to enter a terminal state after all nodes have received the information, within finite time. A protocol solves TDB in a graph $\mathcal{G}$ if it solves it for any source $x \in V$ and time $t \in \mathcal{T}$. We say that it solves TDB in a class $\mathcal{C}$ if it solves TDB for any $\mathcal{G} \in \mathcal{C}$. We are interested in three variations of this problem, following the optimality metrics defined above:

- TDB$[foremost]$, where *each* node receives the information at the *earliest* possible date following its *creation* at the emitter;
- TDB$[shortest]$, where each node receives the information within a minimal number of hops from the emitter;
- TDB$[fastest]$, where the overall duration between first global emission and last global reception is minimized.

For each of these problems, we require that the emitter detects termination, however this detection is not subject to the same optimality constraint (it just has to be finite). TDB thus involves two processes: the actual dissemination of *information messages*, and the exchange of typically smaller *control messages* used for termination detection, both being considered separately in this paper.

Finally, we call *broadcast tree* the (delay-tolerant) tree along which the broadcast takes place, without consideration to the dates when the edges are used, that is, considering only the "flattened" hierarchy of nodes the tree consists of. An optimal (*i.e.,* foremost, fastest, or shortest) broadcast tree is said to be *reusable* if this hierarchy can be purposedly followed to perform a subsequent optimal broadcast.

3. Basic Results and Limitations

Let us first state a general property of the computational relationship between the main three contexts of interest, namely knowing n in $\mathcal{R}$ (noted $\mathcal{R}_n$), knowing Δ in $\mathcal{B}$ (noted $\mathcal{B}_\Delta$), and knowing p in $\mathcal{P}$ (noted $\mathcal{P}_p$). These inclusions will be shown strict later on.

Theorem 2. $\mathscr{P}(\mathcal{R}_n) \subseteq \mathscr{P}(\mathcal{B}_\Delta) \subseteq \mathscr{P}(\mathcal{P}_p)$

Proof. The right inclusion is straight from the fact that $\mathcal{B} \subseteq \mathcal{P}$ and p is a valid bound Δ on the recurrence time. The left inclusion follows from the facts that $\mathcal{R} \subseteq \mathcal{B}$ and n can be inferred in $\mathcal{B}$ if Δ is already known. This can be done by performing, from any node (say u), a depth-first token circulation that will explore the underlying graph G over time. Having a bounded recurrence time indeed allows every node to learn the list of its neighbors in G within Δ time (all incident edges must appear within this duration). As the token is circulated to unvisited nodes, these nodes are marked as visited by u's token and the token is incremented. Upon returning to u, the token value is n. $\square$

We now establish a negative result that justifies the need for additional knowledge in order to solve TDB in any of the considered contexts. In fact we have:

Theorem 3. TDB *cannot be solved in* $\mathcal{P}$ *without additional knowledge.*

Proof. By contradiction, let $\mathcal{A}$ be an algorithm that solves TDB in $\mathcal{P}$. Consider an arbitrary $\mathcal{G} = (V, E, \mathcal{T}, \rho) \in \mathcal{P}$ and $x \in V$. Execute $\mathcal{A}$ in $\mathcal{G}$ starting at time t_0 with x as the source. Let t_f be the time when the source terminates (and thus all nodes have received the information). Let $\mathcal{G}' = (V', E', \mathcal{T}', \rho') \in \mathcal{P}$ such that $V' = V \cup \{v\}$, $E' = E \cup \{(u, v)$ for some $u \in V\}$, for all $t_0 \leq t < t_f$, $\rho'(e, t) = \rho(e, t)$ for all $e \in E$ and $\rho'((u, v), t) = 0$. Now, consider $\rho'((u, v), t) = 1$ for some $t > t_f$, and the period of $\mathcal{G}'$ is some $p' > t - t_0$. Consider the execution of $\mathcal{A}$ in $\mathcal{G}'$ starting at time t_0 with x as the source. Since (u, v) does not appear from t_0 to t_f, the execution of $\mathcal{A}$ at every node in $\mathcal{G}'$ is exactly as at the corresponding node in $\mathcal{G}$. In particular, node x will have entered a terminal state at time t_f with node v not having received the information, contradicting the correctness of $\mathcal{A}$. $\square$

We thus have the following corollary, by inclusion of $\mathcal{P}$.

Corollary 4. TDB *cannot be solved in* $\mathcal{B}$ *nor* $\mathcal{R}$ *without any additional knowledge.*

Hence, additional knowledge of some kind is required to solve TDB in these classes. We consider three types of knowledge, namely, the number of nodes $n = |V|$, an upper bound Δ on the recurrence time (when in $\mathcal{B}$), or the period p (in $\mathcal{P}$). We start by establishing a general impossibility result for TDB$[fastest]$ in $\mathcal{B}$ (and *a fortiori* in $\mathcal{R}$), which cannot be solved even if both n and Δ are known.

Theorem 5. TDB$[fastest]$ *is not solvable in* $\mathcal{B}$ *(and a fortiori in* $\mathcal{R}$*), regardless of the knowledge considered.*

Proof. The argument relates to the very existence of fastest journeys in an unstructured infinite setting. Consider for example the graph $\mathcal{G} = (V, E, \mathcal{T}, \rho) \in \mathcal{B}$ such that $V = \{v_1, v_2, v_3\}$, $E = \{e_1 = (v_1, v_2), e_2 = (v_2, v_3)\}$ and ρ is such that:

- $\forall t \in \mathcal{T}, \forall i \in \mathbb{N}^+, \rho(e_1, t) = 1 \iff i\Delta \leq t < i\Delta + \zeta$
- $\forall t \in \mathcal{T}, \forall i \in \mathbb{N}^+, \rho(e_2, t) = 1 \iff i\Delta + \zeta + i^{-1} \leq t < i\Delta + 2\zeta + i^{-1}$

for any $\Delta \geq 2\zeta + 1$. In such a setting, every period $[i\Delta, (i + 1)\Delta)$ enables a journey $\mathcal{J}_i$ from v_1 to v_3 such that $|\mathcal{J}_i|_t = 2\zeta + i^{-1}$. Since i^{-1} is decreasing[a], there is an infinite sequence of journeys $\mathcal{J}_1, \mathcal{J}_2, ..$ such that $|\mathcal{J}_{i+1}|_t < |\mathcal{J}_i|_t$. $\square$

[a]**Which notation should we use among i^{-1} or $1/i$? If the latter, please update at the four places in the proof.**

Theorem 6. [10] $\text{TDB}[fastest]$ *is feasible in* $\mathcal{P}$ *with a known period* p*, and the solution can be reused for subsequent broadcasts.*

The algorithm from [10] builds fastest broadcast trees in $\mathcal{P}_p$. The solution relies on learning at what time(s) in the period the *temporal eccentricity* of the emitter is minimum, then building a foremost broadcast tree for such date. Note that the broadcast tree so-built remains necessarily optimal in the future, since in $\mathcal{P}$ the whole network schedule repeats forever. It can thus be *memorized* for subsequent broadcasts, *i.e.,* the solution is *reusable*.

The rest of the paper focuses on $\text{TDB}[foremost]$ and $\text{TDB}[shortest]$ in $\mathcal{R}$ and $\mathcal{B}$, knowing n and/or Δ. We then draw some conclusions on the relative difficulty of these three problems, as well as on the computational relationship between $\mathscr{P}(\mathcal{R}_n)$, $\mathscr{P}(\mathcal{B}_\Delta)$, and $\mathscr{P}(\mathcal{P}_p)$.

4. $\text{TDB}[foremost]$

$\text{TDB}[foremost]$ in $\mathcal{R}$ or $\mathcal{B}$ clearly requires some sort of flooding, because the very fact of probing a neighbor to determine if it already has the information compromises the possibility to send it in a foremost fashion (in addition to risking the disappearance of the edge in-between the probe and the real sending). As a consequence of Theorem 3, this problem cannot be solved without knowledge. In this section we first show that it becomes possible in $\mathcal{R}$ if the number of nodes $n = |V|$ is known. The proof is constructive by means of Algorithm 1, whose termination is however not bounded in time. Being in $\mathcal{B}$ with the same knowledge allows its termination to be bounded. Knowing Δ instead of n in $\mathcal{B}$ then allows us to propose another solution (described in Algorithm 2) that has a lower message complexity. This complexity can be further improved if both Δ *and* n are known, as in this case we have the possibility to terminate *implicitly*. Regarding *reusability* for the broadcast of the information, none of the broadcast trees built in $\mathcal{R}$ or even $\mathcal{B}$ turn out to be reusable as such, due to the inherent lack of structure of these classes.

Theorem 7. *Foremost broadcast trees are not reusable as such in* $\mathcal{B}_\Delta$ *(and a fortiori in* $\mathcal{R}_n$*).*

Proof (sketch). By contradiction, let a tree T be a reusable foremost tree with respect to emission date t. Since the *order* in which edges re-appear in $\mathcal{B}$ is arbitrary (as long as they all occur within a Δ time window), an adversary can act on the schedule in such a way that the edges appear in a different order as that of the hierarchy of T, contradicting the fact that the tree is foremost. $\square$

Note that the proof argument does *not* relate to the non-existence of trees whose optimality repeat in the future; in fact, there must be at least one tree whose optimality holds infinitely often since there are finitely many possible trees and infinitely many time spans of duration Δ. The argument actually relates to the *non-decidability* of using a given tree. Nonetheless, observe that the knowledge acquired can be helpful to lower the complexity of the termination detection.

4.1. TDB$[foremost]$ *in* $\mathcal{R}$

In this section we only discuss the knowledge of n since Δ is not defined for $\mathcal{R}$.

4.1.1. *Knowledge of* n

In this section, we show how the problem is solvable when n is known.

The algorithm proceeds as follows (see Algorithm 1 for details). Every time a *new* edge appears locally to an informed node, this node sends the information message onto this edge, and remembers that this edge now leads to an informed node. The first time a node receives the information, it records the sender as parent, transmits the information on its available edges, and sends back a notification message to the parent. Note that these notifications create a parent-relation and thus a converge-cast tree. Each notification is propagated along the converge-cast tree and eventually collected at the emitter. When the emitter has received $n-1$ notifications, it knows all nodes are informed. Observe that the notification messages are sent using the special primitive $send\_retry$ discussed in Section 2.1, to ensure that the parent eventually receives it even if the edge disappears during the first attempt. Information messages, on the other hand, are sent using the normal $send$ primitive. Indeed, if the propagation of such a message fails because the corresponding edge disappears, it simply means that this edge at that particular time did not have to be used (i.e., it did not belong to a valid journey).

Theorem 8. *When n is known,* TDB$[foremost]$ *can be solved in $\mathcal{R}$ exchanging $O(m)$ information messages and $O(n^2)$ control messages, in unbounded time.*

Proof. Since a node sends the information to each new appearing edge, it is easy to see, by connectivity of the *underlying* graph, that all nodes will eventually receive the information. The dissemination itself is necessarily foremost because the information is either directly relayed on edges that are present, or sent as soon as a new edge appears. As for termination detection: every node identifies a unique parent and a converge-cast spanning tree directed towards the source is implicitly constructed; since every node notifies the source (through the tree) and the source knows the total number of nodes, termination is guaranteed. Since information messages might traverse every edge in both directions, and an edge cannot be traversed twice in the same direction, we have that the number of *information* messages is in the worst case $2m$. Since every node but the emitter induces a notification that is forwarded up the converge-cast tree to the emitter, the number of *notification* messages is the sum of distances in the converge-cast tree between all nodes and the emitter, $\sum_{v \in V \setminus \{emitter\}} d_{h\_tree}(v, emitter)$. The worst case is when the graph is a line where we have $\frac{n^2-n}{2}$ control messages. Regarding time complexity, the termination of the algorithm is unbounded due to the fact that the recurrence of the edges is itself unbounded. □

Reusability for subsequent broadcasts Foremost trees are *time-dependent* in the sense that they might be optimal for some emission dates and not be so for other dates. Still, they

Algorithm 1 Foremost broadcast in $\mathcal{R}$, knowing n.

```
1:  $Edge\ parent \leftarrow nil$          // edge the information was received from (for non-emitter nodes).
2:  $Integer\ nbNotifications \leftarrow 0$          // number of notifications received (for the emitter).
3:  $Set{<}Edge{>}\ informedNeighbors \leftarrow \emptyset$          // neighbors known to have the information.
4:  $Status\ myStatus \leftarrow \neg$informed          // status of the node (informed or non-informed).
5:  initialization:
6:    if $isEmitter()$ then
7:      $myStatus \leftarrow$ informed
8:      $send(information)\ on\ I_{now()}$          // sends the information on all present edges.
9:  onAppearance of an edge $e$:
10:   if $myStatus ==$ informed and $e \notin informedNeighbors$ then
11:     $send(information)\ on\ e$
12:     $informedNeighbors \leftarrow informedNeighbors \cup \{e\}$          // (see Prop. 1).
13: onReception of a message $msg$ from an edge $e$:
14:   if $msg.type == Information$ then
15:     $informedNeighbors \leftarrow informedNeighbors \cup \{e\}$
16:     if $myStatus == \neg$informed then
17:       $myStatus \leftarrow$ informed
18:       $parent \leftarrow e$
19:       $send(information)\ on\ I_{now()} \setminus informedNeighbors$          // propagates.
20:       $send\_retry(notification)\ on\ e$          // notifies that this node has the info.
              (this message is to be resent upon the next appearance, in case of failure).
21:   else if $msg.type == Notification$ then
22:     if $isEmitter()$ then
23:       $nbNotifications \leftarrow nbNotifications + 1$
24:       if $nbNotifications == n - 1$ then
25:         $terminate$          // at this stage, the emitter knows that all nodes are informed.
26:     else
27:       $send\_retry(notification)\ to\ parent$
```

remain *valid* trees (though, possibly non-foremost ones) regardless of the considered date. As such, they can be memorized by the nodes in order to be used as *converge-cast* trees for termination detection in subsequent broadcasts. Indeed, while the broadcast is required to be foremost, the detection of termination does not have such constraint. Hence, instead of sending a notification each time a new node is informed (as done previously), nodes can notify their parents (in the converge-cast tree) if and only if they are themselves informed and have received a notification from each of their children (in the converge-cast tree). This reduces the number of control messages from $O(n^2)$ to $O(n)$, having only one notification per edge of the converge-cast tree.

4.2. TDB$[foremost]$ *in* $\mathcal{B}$

If the recurrence time is bounded, then either the knowledge of n or an upper bound Δ on the recurrence time can be used to solve the problem (with various message complexities).

4.2.1. *Knowledge of n.*

Since $\mathcal{B} \subseteq \mathcal{R}$, one can obviously solve $\text{TDB}[foremost]$ in $\mathcal{B}$ using Algorithm 1 (and the same observations apply regarding reusability of the converge-cast tree). Here, however, the termination time becomes bounded due to the fact that the recurrence of edges is itself bounded.

Theorem 9. *When n is known, $\text{TDB}[foremost]$ can be solved in $\mathcal{B}$ exchanging $O(m)$ information messages and $O(n^2)$ control messages, in $O(n\Delta)$ time.*

Proof. Since all edges in E are recurrent within any Δ time window, the delivery of the information at the last node must occur within $(n-1)\Delta$ global time. The same property holds for the latest notification, bounding the overall process to a duration of $\Delta(2n-2)$. The rest follows from Theorem 8. □

4.2.2. *Knowledge of Δ.*

The information dissemination is performed as in Algorithm 1, but the termination detection is different. Thanks to the time-bound Δ on edge recurrence, a node can discover all of its neighbors within Δ time. This fact can be used by a node to determine whether it is a leaf in the broadcast tree (*i.e.,* if it has not informed any other node within Δ time following their own reception time)[b]. This allows the leaves to terminate spontaneously and notify their parent, which recursively terminate after receiving the notifications from all their children and notifying their own parent. With this type of termination, the whole process is in the same spirit as the Dijkstra-Scholten algorithm [15] for static graphs. In that algorithm, a particular node initiates a task that is progressively distributed over the network. The process then terminates recursively based on the relations created during the distribution, each node terminating after its children have terminated.

The temporal adaption for bounded-recurrent TVGs is as follows (see Algorithm 2 for details). First, everytime a *new* edge appears locally to an informed node, this node sends the information on the edge, and records it. The first time a node receives the information, it chooses the sender as parent, memorizes the current time (in a variable $firstRD$), transmits the information on its available edges, and returns an *affiliation* message to its parent using the $send\_retry$ primitive (starting to build the converge-cast tree). This affiliation message is not relayed upward in the tree, but is only intended to inform the direct parent about the existence of a new child (so this parent knows it must wait for a future notification by this node). If an informed node has not received any affiliation message after a duration of $\Delta + \zeta$, it sends a *notification* message to its parent using the $send\_retry$ primitive. The wait is bounded by $\Delta + \zeta$ for the following reasons (see also Figure 1). First note that (due to Prop. 1) the *information* messages cannot be lost when they are sent on an appearing edge, neither can their potential *affiliation* answer. Thus, the loss of information messages can only occur when the information is directly relayed by a node which received it (say,

[b]**This is where we need 2ζ I think...**

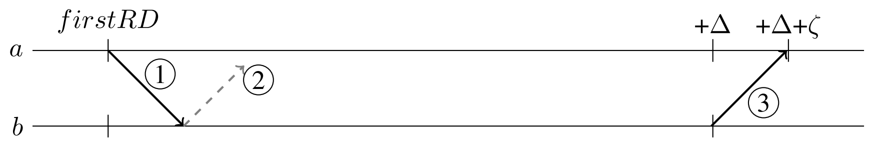

Figure 1. Case when a node waits $\Delta + \zeta$ for receiving potential *affiliation* messages.

as per Figure 1, node a, relaying at time $firstRD$ the information to node b). As before, if the information message is lost, then it simply means that this edge at that time did not have to be used. On the other hand, if the *affiliation* message is lost, it must be sent again ($send\_retry$). However, in the worst case, the common edge disappears just before the affiliation message is delivered, and reappears only $\Delta - 2 \times \zeta$ later (Prop. 1). Affiliation messages can thus be received until $firstRD + \Delta + \zeta$.

If a node has one or more children, it waits until it receives a notification message from each of them, then notifies its parent in the converge-cast tree (using $send\_retry$ again). Once the emitter has received a notification from each of its children, it knows that all nodes are informed.

Theorem 10. *When Δ is known,* $\text{TDB}[foremost]$ *can be solved in $\mathcal{B}$ exchanging $O(m)$ information messages and $O(n)$ control message, in $O(n\Delta)$ time.*

Proof. Correctness follows the same lines of the proof of Theorem 8. However the correct construction of a converge-cast spanning tree is guaranteed by the knowledge of Δ (i.e., the nodes of the tree that are leaves detect their status because no new edges appear within Δ time) and the notification starts from the leaves and is aggregated before reaching the source. The number of information messages is $O(m)$ as the exchange of information messages is the same as in Algorithm 1, but the number of notification and affiliation messages decreases to $2(n-1)$. Each node but the emitter sends a single affiliation message; as for the notification messages, instead of sending a notification as soon as it is informed, each node notifies its parent in the converge-cast tree if and only if it has received a notification from each of its children resulting in one notification message per edge of the tree. The time complexity of the dissemination itself is the same as for the foremost broadcast when n is known. The time required for the emitter to subsequently detect termination is an additional $\Delta + \zeta + \Delta(n-1)$ (the value $\Delta + \zeta$ corresponds to the time needed by the last informed node to detect that it is a leaf, and $\Delta(n-1)$ corresponds to the worst case of the notification process, chained from that node to the emitter). □

Reusability for subsequent broadcasts Clearly, the number of nodes n, which is not *a priori* known here, can be obtained through the notification process of the first broadcast (by having nodes reporting their number of descendants in the tree, while notifying hierarchically). All subsequent broadcasts can thus behave as if both n and Δ were known. Next we show this allows to solve the problem without any control messages.

Algorithm 2 Foremost broadcast in $\mathcal{B}$, knowing a bound Δ on the recurrence time.

```
1:  $Edge\ parent \leftarrow nil$          // edge the information was received from (for non-emitter nodes).
2:  $Integer\ nbChildren \leftarrow 0$          // number of children.
3:  $Integer\ nbNotifications \leftarrow 0$          // number of children that have terminated.
4:  $Set\textless Edge\textgreater\ informedNeighbors \leftarrow \emptyset$          // neighbors known to have the information.
5:  $Date\ firstRD \leftarrow nil$          // date of first reception.
6:  $Status\ myStatus \leftarrow \neg\texttt{informed}$          // status of the node (informed or non-informed).
7:  initialization:
8:    if $isEmitter()$ then
9:      $myStatus \leftarrow \texttt{informed}$
10:     $send(information)\ on\ I_{now()}$          // sends the information on all present edges.
11: onAppearance of an edge $e$:
12:   if $myStatus == \texttt{informed}$ and $e \notin informedNeighbors$ then
13:     $send(information)\ on\ e$
14:     $informedNeighbors \leftarrow informedNeighbors \cup \{e\}$          // (see Prop. 1).
15: onReception of a message $msg$ from an edge $e$:
16:   if $msg.type == Information$ then
17:     $informedNeighbors \leftarrow informedNeighbors \cup \{e\}$
18:     if $myStatus == \neg\texttt{informed}$ then
19:       $myStatus \leftarrow \texttt{informed}$
20:       $firstRD \leftarrow now()$          // memorizes the date of first reception.
21:       $parent \leftarrow e$
22:       $send(information)\ on\ I_{now()} \setminus informedNeighbors$          // propagates.
23:       $send\_retry(affiliation)\ on\ e$          // informs the parent that it has a new child.
24:   else if $msg.type == Affiliation$ then
25:     $nbChildren \leftarrow nbChildren + 1$
26:     $informedNeighbors \leftarrow informedNeighbors \cup \{e\}$
27:   else if $msg.type == Notification$ then
28:     $nbNotifications \leftarrow nbNotifications + 1$
29:     if $nbNotifications == nbChildren$ then
30:       if $\neg isEmitter()$ then
31:         $send\_retry(notification)\ to\ parent$          // notifies the parent in turn.
32:       $terminate$          // whether emitter or not, the node has terminated at this stage.
33: when $now() == firstRD + \Delta + \zeta$:          // tests whether the underlying node is a leaf.
34:   if $nbChildren == 0$ then
35:     $send\_retry(notification)\ on\ parent$
36:     $terminate$
```

4.2.3. *Knowledge of both n and Δ*

In this case, the emitter knows an upper bound on the broadcast termination date; in fact, the broadcast cannot last longer than $(n-1)\Delta$ (this worst case is when the foremost tree is a line). Termination detection can thus become implicit after this amount of time, which removes the need for any control message (whether of affiliation or of notification).

Theorem 11. *When Δ and n are known,* $\text{TDB}[foremost]$ *can be solved in $\mathcal{B}$ exchanging $O(m)$ information messages and no control messages, in $O(n\Delta)$ time.*

5. TDB$[shortest]$

Let us remind that the objective of $\text{TDB}[shortest]$ is to deliver the information to each node within a minimal *number of hops* from the emitter, and to have the emitter detect termination within finite time. We show below that contrary to the foremost case, knowing n is insufficient to perform a shortest broadcast in $\mathcal{R}$ or even in $\mathcal{B}$. This becomes however feasible in $\mathcal{B}$ when Δ is also known. Moreover any shortest tree built at some time t will remain optimal in $\mathcal{B}$ relative to any future emission date $t' > t$. This feature allows the solution to $\text{TDB}[shortest]$ to be possibly reused in subsequent broadcasts.

5.1. TDB$[shortest]$ *in $\mathcal{B}$*

We first show that the knowledge of n is not sufficient to solve $\text{TDB}[shortest]$ in $\mathcal{B}$ (and thus in $\mathcal{R}$), then describe how to solve the problem when Δ is known, and finally when both n and Δ are known.

5.1.1. *Knowledge of n*

The following theorem establishes that knowing n is not sufficient to solve $\text{TDB}[shortest]$ in $\mathcal{B}$ (and thus in $\mathcal{R}$).

Theorem 12. $\text{TDB}[shortest]$ *is not feasible in $\mathcal{B}$ (nor a fortiori in $\mathcal{R}$) knowing only n.*

Proof. By contradiction, let $\mathcal{A}$ be an algorithm that solves $\text{TDB}[shortest]$ in $\mathcal{B}$ with the knowledge of n only. Consider an arbitrary $\mathcal{G} = (V, E, \mathcal{T}, \rho) \in \mathcal{B}$ and $x \in V$. Execute $\mathcal{A}$ in $\mathcal{G}$ starting at time t_0 with x as the source. Let t_f be the time when the source terminates and T the shortest broadcast tree along which broadcast was performed. Let $\mathcal{G}' = (V', E', \mathcal{T}', \rho') \in \mathcal{B}$ such that $V' = V$, $E' = E \cup \{(x, v)$ for some $v \in V : (x, v) \notin E\}$, $\rho'(e, t) = \rho(e, t)$ for all $e \in E, 0 \leq t \leq t_f$, $\rho'((x, v), t) = 0$ for all $t_0 \leq t < t_f$, and $\rho'((x, v), t) = 1$ for some $t > t_f$ (we can take Δ as large as needed here). Consider the execution of $\mathcal{A}$ in $\mathcal{G}'$ starting at time t_0 with x as the source. Since (x, v) does not appear between t_0 and t_f, the execution of $\mathcal{A}$ at every node in $\mathcal{G}'$ will be exactly as at the corresponding node in $\mathcal{G}$ and terminate with v having received the information in more than one hop, contradicting the fact that T is a shortest tree, and thus the correctness of $\mathcal{A}$. □

5.1.2. *Knowledge of Δ*

The idea is to propagate the message along the edges of a *breadth*-first spanning tree of the underlying graph. We present the pseudo-code in Algorithm 3, and provide the following informal description.

Assuming that the message is created at some date t, the mechanism consists of authorizing nodes at level i in the tree to inform new nodes only between time $t + i\Delta$ and

$t + (i + 1)\Delta$ (doing it sooner would lead to a non-shortest tree, while doing it later is pointless because all the edges have necessarily appeared within one Δ). So the broadcast is confined into rounds of duration Δ as follows: whenever a node sends the information to another, it sends a time value that indicates the remaining duration of its round (that is, the starting date of its own round plus Δ minus the current time minus the crossing delay), so the receiving node knows when to start informing new nodes in turn (if it had not the information yet). For instance in Figure 2 when the node a attempts to become b's parent, node a transmits its own starting date plus Δ minus the current date minus ζ. This duration corresponds to the exact amount of time the child would have to wait, if the relation is established, before integrating other nodes in turn. If a node has not informed any other node during its round, it notifies its parent. When a node has been notified by all its children, it notifies its parent. Note that this requires parents to keep track of the number of children they have, and thus children need to send *affiliation* messages when they select a parent (with the same constraints as already discussed in Figure 1). Finally, when the emitter has been notified by all its children, it knows the broadcast is terminated.

Theorem 13. $\mathrm{TDB}[shortest]$ *can be solved in* $\mathcal{B}$ *knowing* Δ*, exchanging* $O(m)$ *info. messages and* $O(n)$ *control messages, in* $O(n\Delta)$ *time.*

Proof. The fact that the algorithm constructs a breadth-first (and thus shortest) delay-tolerant spanning tree follows from the connectivity over time of the underlying graph and from the knowledge of the duration Δ. The bound on recurrence is used to enable a rounded process whereby the correct distance of each node to the emitter is detected. The number of *information* messages is $2m$ as the dissemination process exchanges at most two messages per edge. The number of *affiliation* and *notification* messages are each of $n - 1$ (one per edge of the tree). The time complexity for the construction of the tree is at most $(n-1)\Delta$ to reach the last node, plus $\Delta+\zeta$ at this node, plus at most $(n-1)\Delta$ to aggregate this node's notification. (The additional ζ caused by waiting affiliation messages matters only for the last round, since the construction continues in parallel otherwise.) The total is thus at most $(2n - 1)\Delta + \zeta$. □

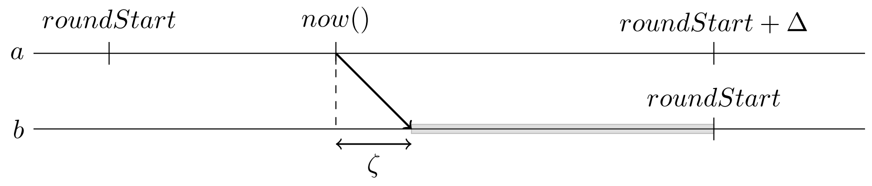

Figure 2. Propagation of the rounds of duration Δ.

Reusability for subsequent broadcasts Thanks to the fact that shortest trees remain shortest regardless of the emission date, all subsequent broadcasts can be performed within the same, already known tree, which reduces the number of information message from $O(m)$ to $O(n)$. Moreover, if the depth d of the tree is detected through the first notification process, then all subsequent broadcasts can enjoy an implicit termination detection that is itself optimal in time (after $d\Delta$ time). No control message is needed.

5.1.3. *Knowledge of n and Δ*

When both n and Δ are known, one can apply the same dissemination procedure as in Algorithm 3 combined with an implicit termination detection that avoids using control messages at all. Indeed, each node learns (and possibly informs) all of its neighbors within Δ time. Since the underlying graph is connected, the whole process must then complete within $n\Delta$ time. Hence, if the emitter knows both Δ and n, it can simply wait $n\Delta$ time, then terminate implicitly.

Theorem 14. *When n and Δ are known,* $\text{TDB}[shortest]$ *can be solved in $\mathcal{B}$ exchanging $O(m)$ info. messages and no control messages, in $O(n\Delta)$ time.*

However, such a strategy would prevent the emitter from learning the depth d of the shortest tree, and thus prevent lowering the termination bound to $d\Delta$ time. An alternative solution would be to achieve explicit termination for the first broadcast in order to build a reusable broadcast tree (and learn its depth d in the process). In this case, dissemination is achieved with $O(m)$ information messages, termination detection is achieved similarly to Algorithm 3 with $O(n)$ control messages (where however affiliation messages are not necessary, and the number of control messages would decrease to $n-1$). In this way we would have an increase in control messages, but the subsequent broadcasts could reuse the broadcast tree for dissemination with $O(n)$ information messages, and termination detection could be implicit with no exchange of control message at all after $d\Delta$ time. The choice of either solution may depend on the size of an information message and on the expected number of broadcasts planned.

6. Computational Relationship

On the basis of this paper results, we can conclude regarding the computational relationship between $\mathscr{P}(\mathcal{R}_n)$, $\mathscr{P}(\mathcal{B}_\Delta)$, and $\mathscr{P}(\mathcal{B}_\Delta)$, that is, prove the validity of Equation 1.

Theorem 15. $\mathscr{P}(\mathcal{R}_n) \subsetneq \mathscr{P}(\mathcal{B}_\Delta) \subsetneq \mathscr{P}(\mathcal{P}_p)$

Proof. The fact that $\mathscr{P}(\mathcal{R}_n) \subseteq \mathscr{P}(\mathcal{B}_\Delta) \subseteq \mathscr{P}(\mathcal{P}_p)$ was observed in Theorem 2. To make the left inclusion strict, one has to exhibit a problem Π such that $\Pi \in \mathscr{P}(\mathcal{B}_\Delta)$ and $\Pi \notin \mathscr{P}(\mathcal{R}_n)$. By Theorem 12 and Theorem 13, $\text{TDB}[shortest]$ is one such example. The right inclusion is similarly proven strict, based on the fact that $\text{TDB}[fastest]$ is in $\mathscr{P}(\mathcal{P}_p)$ (Theorem 6) but it is not in $\mathscr{P}(\mathcal{B}_\Delta)$ (Theorem 5). □

Now, considering the fact that $\text{TDB}[foremost] \in \mathscr{P}(\mathcal{R}_n)$ while $\text{TDB}[shortest] \notin \mathscr{P}(\mathcal{R}_n)$, and the fact that $\text{TDB}[shortest] \in \mathscr{P}(\mathcal{B}_\Delta)$ while $\text{TDB}[fastest] \notin \mathscr{P}(\mathcal{B}_\Delta)$, together with the inclusions of Theorem 15, we have

$$\mathrm{TDB}[foremost] \preceq_{feasibility} \mathrm{TDB}[shortest] \preceq_{feasibility} \mathrm{TDB}[fastest]$$

where $\preceq_{feasibility}$ is a partial order on these problems topological requirements (relative to *feasibility*). The order is "only" partial here because the variations of feasibility of these problems may be different in another set of assumptions. Following a similar reasoning (that is the fact that the solutions to $\mathrm{TDB}[shortest]$ are reusable in $\mathcal{B}_\Delta$ whereas those to $\mathrm{TDB}[foremost]$ are not) leads to

$$\mathrm{TDB}[shortest] \preceq_{reusability} \mathrm{TDB}[foremost]$$

where $\preceq_{reusability}$ is a partial order on these problems topological requirements (relative to *reusability*). This result suggests that a problem could be both easier or more difficult than another, depending on which aspect is looked at (feasibility *vs.* reusability). In other words, the difficulty of these problems seems to be multi-dimensional. Finally, whether reusability is easier for $\mathrm{TDB}[fastest]$ or $\mathrm{TDB}[foremost]$ is an open question, both of these problems being unsolvable in $\mathcal{B}_\Delta$ but solvable in $\mathcal{P}_p$ [10].

7. Concluding Remarks

In this paper we looked at three particular problems (shortest, fastest, and foremost broadcast) in three particular classes of dynamic graphs (recurrent, time-bounded recurrent, and periodic graphs). By comparing the feasibility of these problems within each class, we came to observe both requirement relationships between the problems and computational relationships between the classes.

The methodology exploited here goes certainly beyond the scope of these problems and graph classes. In general terms, the principle here is to connect problems through their feasibility in a set of hierarchized classes of dynamic graphs. One can take two problems P_1 and P_2 that are respectively feasible in classes C_1 and C_2, then show that both $C_1 \subseteq C_2$ and P_1 is not feasible in C_2, to finally conclude that P_1 is at least as difficult as P_2 in terms of topological requirements. As we have shown, such characterizations can also be used reversely to exhibit computational relationship between several classes/assumptions. We hope and believe this approach could be used as a generic mean to explore further distributed algorithms in a dynamic context.

Bibliography

[1] D. Angluin, J. Aspnes, Z. Diamadi, M. Fischer, and R. Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

[2] D. Angluin, J. Aspnes, D. Eisenstat, and E. Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[3] B. Awerbuch and S. Even. Efficient and reliable broadcast is achievable in an eventually connected network. In *Proceedings of the 3rd ACM symposium on Principles of distributed computing (PODC)*, pages 278–281, Vancouver, Canada, 1984. ACM.

[4] H. Baumann, P. Crescenzi, and P. Fraigniaud. Parsimonious flooding in dynamic graphs. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 260–269, Calgary, Canada, 2009. ACM.

[5] C. Bettstetter, G. Resta, and P. Santi. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(3):257–269, 2003.

[6] M. Biely, P. Robinson, and U. Schmid. Agreement in directed dynamic networks. In *Proceedings of the 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, 2012.

[7] B. Bui-Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science*, 14(2):267–285, April 2003.

[8] J. Burgess, B. Gallagher, D. Jensen, and B.N. Levine. Maxprop: Routing for vehicle-based disruption-tolerant networks. In *Proceedings of the 25th IEEE Conference on Computer Communications (INFOCOM)*, pages 1–11, Barcelona, Spain, 2006. IEEE.

[9] A. Casteigts, S. Chaumette, and A. Ferreira. Characterizing topological assumptions of distributed algorithms in dynamic networks. In *16th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 126–140, Piran, Slovenia, 2009. Springer. (Full version on *arXiv:1102.5529*).

[10] A. Casteigts, P. Flocchini, B. Mans, and N. Santoro. Measuring temporal lags in delay-tolerant networks. *IEEE Transactions on Computers*, 63(2):397–410, Feb 2014.

[11] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro. Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems*, 27(5):387–408, 2012.

[12] I. Chatzigiannakis, O. Michail, and P. Spirakis. Mediated population protocols. *36th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 363–374, 2009.

[13] A. Clementi, C. Macci, A. Monti, F. Pasquale, and R. Silvestri. Flooding time in edge-markovian dynamic graphs. In *Proceedings of the 27th ACM Symposium on Principles of distributed computing (PODC)*, pages 213–222, Toronto, Canada, 2008. ACM.

[14] A. Clementi, A. Monti, F. Pasquale, and R. Silvestri. Information spreading in stationary markovian evolving graphs. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12. IEEE, 2009.

[15] E.W. Dijkstra and C.S. Scholten. Termination detection for diffusing computations. *Information Processing Letters*, 11(1):1–4, 1980.

[16] C. Dutta, G. Pandurangan, R. Rajaraman, Z. Sun, and E. Viola. On the complexity of information spreading in dynamic networks. In *SODA*, pages 717–736, 2013.

[17] A. Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24–29, 2004.

[18] P. Flocchini, M. Kellett, P. Mason, and N. Santoro. Searching for black holes in subways. *Theory of Computing Systems*, 50(1):158–184, 2012.

[19] P. Flocchini, B. Mans, and N. Santoro. Exploration of periodically varying graphs. In *Proceedings of 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 534–543, 2009.

[20] P. Flocchini, B. Mans, and N. Santoro. On the exploration of time-varying networks. *Theoretical Computer Science*, 469:53–68, 2013.

[21] S. Guo and S. Keshav. Fair and efficient scheduling in data ferrying networks. In *Proceedings of ACM Conference on Emerging Network Experiment and Technology (CoNEXT)*, New York, USA, 2007. ACM.

[22] B. Haeupler and F. Kuhn. Lower bounds on information dissemination in dynamic networks. *arXiv preprint arXiv:1208.6051*, 2012.

[23] J. Harri, F. Filali, and C. Bonnet. Mobility models for vehicular ad hoc networks: a survey and taxonomy. *IEEE Communications Surveys & Tutorials*, 11(4):19–41, 2009.

[24] D. Ilcinkas and A. Wade. On the power of waiting when exploring public transportation sys-

tems. *Proceedings of the 15th International Conference on Principles of Distributed Systems (OPODIS)*, pages 451–464, 2011.

[25] P. Jacquet, B. Mans, and G. Rodolakis. Information propagation speed in mobile and delay tolerant networks. *IEEE Transactions on Information Theory*, 56(1):5001–5015, 2010.

[26] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *Proceedings of Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 145–158, 2004.

[27] F. Kuhn, N. Lynch, and R. Oshman. Distributed computation in dynamic networks. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC)*, pages 513–522, Cambridge, USA, 2010. ACM.

[28] C. Liu and J. Wu. Scalable routing in cyclic mobile networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(9):1325–1338, 2009.

[29] R. O'Dell and R. Wattenhofer. Information dissemination in highly dynamic graphs. In *Proceedings of the Joint Workshop on Foundations of Mobile Computing (DIALM-POMC)*, pages 104–110, Cologne, Germany, 2005. ACM.

[30] Y. Peres, A. Sinclair, P. Sousi, and A. Stauffer. Mobile geometric graphs: Detection, coverage and percolation. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 412–428. SIAM, 2011.

[31] Z. Zhang. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges. *IEEE Communications Surveys & Tutorials*, 8(1):24–37, 2006.

[32] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC)*, pages 187–198, 2004.

Algorithm 3 Shortest broadcast in $\mathcal{B}$, knowing a bound Δ on the recurrence.

```
1: $Edge\ parent \leftarrow nil$ // edge the information was received from (for non-emitter nodes).
2: $Date\ roundStart \leftarrow +\infty$    // date when the underlying node starts informing new
    nodes.
3: $Set\ <Edge>\ children \leftarrow \emptyset$    // set of children from which a notification is expected.
4: $Integer\ nbNotifications \leftarrow 0$ // number of children that have sent their notification.
5: $Set<Edge>\ informedNeighbors \leftarrow \emptyset$    // set of neighbors known to have the info.
6: $Status\ myStatus \leftarrow \neg$informed    // status of the node (informed or non-informed).
7: initialization:
8:    if $isEmitter()$ then
9:        $roundStart \leftarrow now()$    // causes the procedure "when
        $now() == roundStart$:" (below) to execute.
10: onAppearance of an edge $e$:
11:    if $myStatus ==$ informed then
12:        if $e \notin informedNeighbors$ then
13:            $send(roundStart + \Delta - now() - \zeta)$ on e // time until the end of the round.
14:            $informedNeighbors \leftarrow informedNeighbors \cup \{e\}$    // (see Prop. 1).
15: onReception of a message $msg$ from an edge $e$:
16:    if $msg.type == Duration$ then
17:        $informedNeighbors \leftarrow informedNeighbors \cup \{e\}$
18:        if $parent == nil$ then
19:            $parent \leftarrow e$
20:            $roundStart \leftarrow now() + msg$
21:            $send\_retry(affiliation)$ on $e$
22:    else if $msg.type == Affiliation$ then
23:        $children \leftarrow children \cup \{e\}$
24:    else if $msg.type == Notification$ then
25:        $nbNotifications \leftarrow nbNotifications + 1$
26:        if $nbNotifications == |children|$ then
27:            if $\neg isEmitter()$ then
28:                $send\_retry(notification)$ on $parent$
29:            $terminate$
30: when $now() == roundStart$:
31:    $myStatus \leftarrow$ informed
32:    $send(\Delta\text{-}\zeta)$ on $I_{now()} \setminus informedNeighbors$    // nodes that receive this message
    and
    have no parent yet will take this node as parent and wait $\Delta$-$\zeta$ before informing new
    nodes.
33: when $now() == roundStart + \Delta + \zeta$: // tests whether the underlying node is a leaf.
34:    if $|children| == 0$ then
35:        $send\_retry(notification)$ on $parent$
```