Time to Change: On Distributed Computing in Dynamic Networks

Nicola Santoro
School of Computer Science, Carleton University, Ottawa, Canada
santoro@scs.carleton.ca

Abstract
In highly dynamic networks, topological changes are not anomalies but rather integral part of their nature. Such networks are becoming quite ubiquitous. They include systems where the entities are mobile and communicate without infrastructure (e.g. vehicles, satellites, robots, or pedestrian smartphones): the topology changes as the entities move. They also include systems, such as peer-to-peer networks, where the changes are caused by entities entering and leaving the system. They even include systems where there is no physical mobility at all, such as social networks. A vast literature on these dynamic networks has been produced in many different fields, including distributed computing. The several efforts to survey the status of the research and attempts to clarify and classify models and assumptions, have so far brought more valuable bibliographic data than order and clarity. Goal of this note is to ask questions that might bring author and readers to start to clarify some important research aspects and put some order in a sometimes confusing field. The focus here is entirely on distributed computing, specifically on its deterministic aspects.

1998 ACM Subject Classification C.2.4 Distributed Systems, F.1.1 Models of Computation, F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases distributed computing, dynamic networks, time-varying graphs, mobile agents

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2015.3

Category Keynote

1 Introduction
Computing in networked environments has been one of the core research areas and concerns of distributed computing. The structure of such environments is modeled as a simple graph, where the nodes correspond to the computational entities and the edges to existing communication links between pairs of entities.

While assuming the network structure to be static, the research soon focused on the study of topological changes in order to model faults and failures occurring in real distributed systems. Examined in the context of fault-tolerance, the changes were however considered a small scale phenomenon, limited and localized. The possibility of extensive changes recurring in the lifetime of the system has been object of study within the field of self-stabilization: incorrect computations are allowed to take place in a period of instability; it is however assumed that the instability stops, at least long enough, so that the computation can eventually produce correct results. None of these studies can deal with systems where the topology is subject to
extensive changes that can occur everywhere and possibly never stop, systems where changes are not anomalies but rather integral part of their nature.

Such systems do indeed exist and are becoming quite ubiquitous. Such are for example systems where the entities are truly mobile and can communicate without infrastructure (e.g. vehicles, satellites, robots, or pedestrian smartphones): an edge exists between two entities if they are within communication range; the topology changes (possibly dramatically) as the entities move. In these systems end-to-end connectivity does not necessarily hold, the network might be always disconnected, still communication may be available over time and space making broadcast, routing, computations feasible. These networks have been extensively studied from engineers uder the names of delay-tolerant or challenged networks. In addition to these ad-hoc wireless mobile networks, the same level of dynamic changes occur in systems where there is no explicit communication, such as swarms of autonomous mobile robots: each robot sees the positions of the robots within its visibility range, and based on these positions, it computes a destination and moves there; the topology of the visibility graph changes during the execution of protocol. It can also occur in systems where the changes are caused by entities entering and leaving the system, such as peer-to-peer networks. It can even occur in systems where there is no physical mobility at all, such as social networks.

Due to the abundance of different contexts where these highly dynamic networks arise and their importance, a vast literature in many different fields has been produced, including by distributed computing researchers, focusing on one or another aspect of these systems. Significant efforts have been made to model and formally describe the aspects under examination; not surprising, the ’lexicon’ is rather confusing, with the same object being given different names (e.g., “temporal distance” introduced in [23], has been subsequently called “reachability time” in [55], “information latency” in [64], and “temporal proximity” in [65]) and sometimes the same name being used to define different classes of objects (e.g., “temporal graphs” defined in [65] vs the later use in [74]).

There have been several efforts to survey the status of the research, attempting to clarify and classify models and assumptions and research results (e.g. the recent [19, 57, 67, 74]), in particular the monumental effort by Holme [56]. From the distributed computing viewpoint, these efforts have so far brought more valuable bibliographic data and interesting information than order and clarity. Goal of this note is to ask questions that might bring author and readers to start to clarify some important research aspects and put some order in a sometimes confusing field.

2 What and How to Represent?

There are many popular ways to represent dynamic networks: temporal networks, evolving graphs, multiplex networks; as discussed below, they are actually all equivalent and equally limited by the restrictive assumptions they make.

A less constrained general mathematical formalism that describes many different types of dynamic networks is the one offered by TVG (for time-varying graph) introduced in [28] and described next.

2.1 TVG

TVG is a simple model that includes the other commonly used representations as instances, plus it allows to express other (possibly more complex) computational dynamic systems.
In this model, the dynamic system is described as a time-varying graph \( \mathcal{G} = (V, E, T, \psi, \rho, \zeta) \), where \( V \) is the set vertices or nodes, representing the system entities (e.g., vehicles, robots) and \( E \subseteq V \times V(\times L) \) is the set of (directed or undirected) edges representing connections (e.g., communication, contact, relation, link,...) between pairs of entities; edge can be labeled, the labels in \( L \) are domain-specific (e.g., intensity of relation, type of carrier, ...) and possibly multi-valued (e.g. satellite link; bandwidth of 4 MHz; encryption available;...).

The system exists for a contiguous interval of time \( T \subseteq \mathbb{R} \) called lifetime. The system is said to be limited if \( T \) is closed, unlimited otherwise; in both cases, it is generally assumed that the system has a beginning, which occurs at time \( t = 0 \).

The dynamics of the system is specified by the node presence function, \( \psi : V \times T \to \{0, 1\} \), and the edge presence function, \( \rho : E \times T \to \{0, 1\} \) \iff \( x \in V \) (resp., edge \( e \in E \) is present at time \( t \in T \).

An important concept is that of a snapshot of the system at time \( t \in T \), denoted by \( G(t) = (N(t), E(t)) \), where \( N(t) = \{ x \in N : \psi(x, t) = 1 \} \) and \( E(t) = \{ e \in E : \rho(x, t) = 1 \} \) are the nodes and edges present at time \( t \). The footprint of the system is just the aggregate graph of all footprints: \( G = (V, E) \). It is assumed that the footprint is connected; otherwise, the system is considered composed of separate non-interacting dynamic systems, each with a connected footprint. Observe that connectivity of the footprint \( G \) has no implication on the connectivity of any of the snapshots \( G(t) \).

The TVG model can be naturally extended by adding any number of (possibly temporal) functions on the nodes (e.g., \( f_i : V \times T \to F_i \)) and/or on the edges (e.g., \( h_j : E \times T \to H_j \)) to appropriate domains, to describe specific system features (e.g., cost, weight, energy, ...).

### 2.2 Synchronous Systems

The model can be also restricted by imposing assumptions. The most common restriction is by assuming a discrete synchronous system: the changes in the system occur at discrete time steps (i.e., \( T \subseteq \mathbb{N} \)), and its dynamics is fully described as a sequence of synchronous rounds.

For discrete synchronous systems, a TVG (or aspects of it) can be equivalently and conveniently expressed in other ways.

For example, a compact representation is by listing for each edge \( e \) the set \( I(e) \) of all the contiguous intervals of time when \( e \) was present. Indeed the couple \( I = (N, I) \), where \( I = \bigcup_{e \in E} I(e) \), is a common definition of the class of discrete synchronous systems, and it is known in the literature as temporal networks [57]. Note that temporal networks do not consider node presence and more importantly they have no latency; in other words they describe discrete synchronous systems with instantaneous contacts.

Another popular representation for discrete synchronous systems is by considering the sequence \( S(\mathcal{G}) = < G(0), G(1), G(2), \ldots, G(t), \ldots > \) of all\(^1\) the snapshots of \( \mathcal{G} \). Indeed any

\(^1\) A more succinct representation is to consider only the snapshots where a topological change occurs.
sequence of static graphs $S = \langle G_0, G_1, G_2, \ldots, G_t, \ldots \rangle$, called evolving graph [44], can be seen as the sequence of snapshots of a unique TVG $\mathcal{G}$ (once the latency function has been defined). The idea of representing a dynamic graph as a sequence of static graphs was first suggested in [52]; the proposal of using a sequence of graphs to model discrete synchronous systems was made by [89] in the context of social networks, and by [44] in the context of ad-hoc mobile networks. The evolving graph representation is perhaps the most commonly used for discrete synchronous systems (often without references and under new names). As mentioned, the latency has to be specified and added to this representation to make it equivalent to that of a (discrete synchronous) TVG.

Given the sequence of snapshots $S(G) = \langle G(0), G(1), G(2), \ldots, G(t), \ldots \rangle$, consider the multy-layer graph $\mathcal{M}(S)$ obtained connecting $G(t)$ to $G(t+1)$ by adding an edge from each node in $G(t)$ to the same node in $G(t+1)$ (if present in both snapshots). Clearly this multi-layer graph captures the same information as the evolving graph $S(G)$; thus, if enanced with the specification of the latency, it becomes computationally equivalent to the TVG $\mathcal{G}$. Such a multy-layer graph $\mathcal{M}$, called multiplex network, is a commonly used representation of a discrete synchronous systems (for recent survey see [67]).

### 2.3 Journeys and Distances

A crucial concept in dynamic networks is that of journey, the dynamic equivalent of “walk” in static graphs. More precisely a journey is a sequence $\mathcal{J} = \langle (e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k) \rangle$, where $\langle e_1, e_2, \ldots, e_k \rangle$ is a walk in $G$, $\rho(e_i, t_i) = 1$ and $t_i + \zeta(e_i, t_i) \leq t_{i+1}$. That is, the walk edges are present in the graph at the appropriate times with the latency long enough so each edge can be traversed in time. Time $t_1$ is the start time of the journey $\mathcal{J}$, and $t_k + \zeta(e_k, t_k)$ is the time it ends, denoted by $\text{start}(\mathcal{J})$ and $\text{end}(\mathcal{J})$ respectively. Journeys could actually be infinite, in which case they have no end.

Depending on whether or not there are time gaps in the walk, we can distinguish between two types of journeys: a journey $\mathcal{J} = \langle (e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k) \rangle$ is direct if, for all $1 \leq i < k$, $t_i + \zeta(e_i, t_i) = t_{i+1}$, i.e., there is no waiting before traversing any edge; otherwise is indirect. This distinction is sometimes relevant because there are dynamic networks where buffering is not supported (and thus only direct journeys are allowed). Some interesting
differences between allowing and not allowing waiting have been recently established [25, 58].
In the following, we assume that waiting is allowed, and thus make no distinction between
direct and undirect journeys.

A finite journey is a walk over time from a source to a destination and therefore has not
only a topological length $|J|$, defined as the number of edges in the walk, but also a temporal
length $||J|| = end(J) − start(J)$ defined as the time elapsed to perform the walk.

This gives rise to distinct definitions of distance in a time-varying graph $G$:

- **shortest distance**: $d(u, v, t) = \min \{|J| \colon J \in J(u, v) \land start(J) \geq t\}$ (i.e., min hop);
- **fastest distance**: $\delta(u, v, t) = \min \{|J| \colon J \in J(u, v) \land start(J) \geq t\}$ (i.e., min duration);
- **foremost distance**: $\partial(u, v, t) = \min \{end(J) \colon J \in J(u, v) \land start(J) \geq t\}$ (i.e., ends first);

where $J(u, v)$ denotes the set of all journeys from $u$ to $v$.

Indeed, for all all the classical measures of static graphs and networks (diameter, degree,
eccentricity, centrality, etc.) there exist (one or more) temporal counterparts (temporal
diameter, temporal degree, temporal eccentricity, temporal centrality, etc.).

## 3 What to Investigate?

### 3.1 Dead or Live, Centralized or Distributed?

Most of the existing algorithmic investigations and results assume global a-priori knowledge
of the system; that is, the entire graph $G$ is given as an input to the computation. In other
words, they consider dynamic networks that are (not only discrete synchronous but also)
limited; the investigation is post-mortem, i.e. after the system has ended its limited life-time
(and no more data is being produced); and the computation is off-line, totally centralized.
That is, they are centralized investigations of dead systems. This is for example the case of
[16, 23, 44, 48, 53, 59, 64, 78, 73], and in particular of all the investigations on data previously
collected from real dynamic networks (e.g., [62, 63, 66, 82, 84, 85, 92]).

The interest of this note is however on distributed computations in dynamic networks.
This means that the computation is distributively performed inside the time-varying graph.
In other words, the system is live; its lifetime $T$ is unlimited (as far as the computation
is concerned); and the computation is decentralized and localized.

The lifetime $T$ of $G$ is divided in three parts: the instantaneous present, $\hat{t}$; the limited
past, $past(\hat{t}) = \{t \in T \colon 0 < t < \hat{t}\}$; and the unlimited future, $future(\hat{t}) = \{t \in T \colon t > \hat{t}\}$.

**Figure 2** Three types of minimal journeys; in this example latency is 0.
Each computational entity (node, web site, vehicle, ...) operates in the present, and is aware only of the local events, i.e., topological changes in which it is involved; it might remember its past (if it has enough memory); however, it does not know the future.

### 3.2 Who is in Control?

To understand how to deal with the future, it is important to understand the relationship between the changes occurring in the system and the computation performed by the entities. To ask what is causing the system changes is important but it does not necessarily clarify the nature of the relationship. For example, in ad-hoc wireless mobile networks, it is the movement of the entities that causes the topological changes; similarly, in robotic swarms or in mobile sensor networks, the changes are generated by the movement of the entities. Apparently, in all these systems, the cause of changes is the same: the entities’ movements; there is however a fundamental difference between the former and the latter systems.

In the latter, the movement of an entity is influenced by the computation: where an entity goes next (and thus what new edges are being formed) is determined by the protocol; in other words, the computation generates the graph. This means that we (algorithm designers) could program the entities so to construct graphs with specific properties. This indeed is what happens when we design protocols that allow autonomous mobile robots with limited visibility to arrange themselves in space so to form a specific geometric pattern, or mobile sensors to spread over a territory so to homogenously cover it. We shall call these types of situations as of controlled generation of the graph.

Totally different is the first type of systems: the movements are independent of the computation (e.g., broadcast, routing, etc) performed by the entities. More precisely, the computation has no control over the topological changes of the system. We call this type of situation as of an uncontrolled generation of the graph; in this situation, we actually envision the changes as caused by an adversary operating against the computation.

It is on this type of systems that we focus in the rest of this note.

### 3.3 What Problems?

In the live systems we consider, the computational entities operate in a decentralized and localized matter in an ever changing scenario, without control over the changes. The research investigations on these systems have been both intensive and extensive, carried out mainly within the engineering community.

The distributed computing focus has been on a variety of classical problems, such as information propagation: routing, multicast, broadcast, gossip, etc. (e.g., [4, 5, 13, 14, 26, 27, 31, 32, 33, 35, 42, 54, 83, 93]); coordination: aggregation, naming, counting, etc. [3, 20, 34, 40, 76, 79]; computability (e.g., [7, 8, 21, 24, 28, 36, 39, 69, 75]); control: election, consensus, synchronization, etc. (e.g., [6, 10, 11, 15, 17, 18, 30, 36, 49, 50, 61, 68, 70, 88].

A separate area of research has been on computations by entities opportunistically moving from node to node by traversing edges when they appear. This is the classical environment of mobile agents (or robots) moving in a network, extended to dynamic networks. Also this case is one of uncontrolled generation, as the mobile agents have no control over the changes in the network; the changes are often seen as generated by the movement of carriers. The main research focus has been on search and exploration (e.g., [1, 2, 12, 22, 37, 41, 43, 45, 46, 58]).
Without Control What to Assume?

The fact that the future is unknown and under the control of an adversary means that, in order to be able to perform some meaningful computation, some assumptions have to be made. Usually called a priori knowledge and sometimes oracles, these assumptions restrict the universe under observation, limiting the power of the adversary.

As mentioned before, the most common assumption is that the system is discrete synchronous. Another common (usually hidden) assumption is that the footprint $G = (V, N)$ is finite, i.e., both $N$ and $E$ are finite. Let us make these assumptions. Still, they are not enough, and additional assumptions are necessary.

For example, in the non-deterministic realm, additional assumptions are made on the probability of the appearance of every edge (e.g., it obeys a Poisson process) or on the relationship between successive snapshots $G(t)$ and $G(t+1)$ (e.g., edge-Markovian process); e.g., see [14, 29, 31, 33]. The focus of this note is however on the deterministic side.

4.1 Frequency Assumptions

A type of deterministic additional assumptions are about the frequency of the changes. Let an edge $e \in E$ be called transient if it appears in a finite number of snapshots $G(t)$, recurrent otherwise. Notice that if there are both transient and recurrent edges, there exists a time $\bar{t}$ after which all appearing edges are recurrent.

We say that the system $G$ is recurrent if all edges are recurrent: $\forall e \in E, t \in T, \exists t' > t : e \in E(t')$; this restriction is sometimes called local fairness. Example of recurrent systems are population protocols with a fair scheduler. Investigations include e.g., [1, 2, 6, 7, 8, 26, 27, 75].

A more restricted class is that of $B$-bounded recurrent systems, $B \in \mathbb{N}$, defined by the assumption $e \in E(t) \Rightarrow e \in E(t')$ where $t < t' \leq t + B$ (e.g., [1, 2, 26, 27]).

Even more restricted is the class of P-periodic systems, $P \in \mathbb{N}$, defined by the assumption $e \in E(t) \Rightarrow e \in E(t + P)$. Examples of periodic systems are public transports with fixed timetable, low-earth orbiting satellite (LEO) systems. Study of computing in periodic systems include [1, 2, 26, 27, 45, 46, 58, 60, 71, 72]. Notice that to determine whether or not a system is periodic is undecidable. Similarly undecidable is to verify if a periodic system has period $P$. In other words, periodicity without knowledge of the period is not a useful assumption.

Note that all these frequency assumptions are restrictions on the functions $\psi$ and $\rho$. Another type of requirement sometimes imposed on those functions is that they should somehow reflect the behaviour of “real life” systems. With this motivation, the engineering community has developed and has been using several mobility patterns, i.e. restrictions on the functions $\psi$ and $\rho$ to mimic experimentally observed changes due to mobility of vehicles, humans, etc. (e.g., [38, 81, 87]).

4.2 Connectivity Assumptions

Of all the common assumptions we considered so far, none has any impact on the connectivity of the network. Indeed simultaneous end-to-end connectivity might not be guaranteed, and it is also possible that all snapshots $G(t)$ might be disconnected in spite of those assumptions. Not surprising, especially for discrete synchronous systems, a popular class of additional assumptions are those relating to connectivity.

The weakest such assumption is temporal connectivity, that is $\forall x, y \in V, t \in T$ there exists a journey $J \in J_{(u,v)}$ such that start$(J) \geq t$. This assumption is typical in the engineering investigations; it is also the one used in the pioneering work of Awerbuch and Even [13].
Temporal
Recurrent
Periodic
13interval
T3interval

Figure 3 Connectivity assumptions.

Stronger assumptions require simultaneous end-to-end connectivity to hold at some point in time. In increasing order of requirement’s strength, we have recurrent connectivity: for all \( t \in \mathcal{T} \), there exists \( t' \geq t \) such that \( G(t') \) is connected (e.g., \([10, 61]\)); B-bounded connectivity: for all \( t \in \mathcal{T} \), there exists \( t' \leq t + B \) such that \( G(t') \) is connected; and P-periodic connectivity: for all \( t \in \mathcal{T} \), there exists \( t' \geq t \) such that \( \forall j \in G(t' + jP) \) is connected.

Finally, permanent connectivity: for all \( t \in \mathcal{T} \), \( G(t) \) is connected; this assumption is also known as 1-interval connectivity. An even more stringent assumption is permanent connectivity with persistent backbone which requires that the same connected spanning subgraph persists for \( T > 1 \) consecutive snapshots: for all \( t \in \mathcal{T} \), \( G_T(t) = \bigcap_{0 \leq j < T} G(t + j) \) is connected; this assumption is also known as \( T \)-interval connectivity. Both permanent and \( T \)-interval connectivity are often assumed in distributed computations (e.g., \([3, 41, 43, 54, 59, 68, 69, 70, 83, 88]\)).

4.3 Power of Assumptions

Notice that to each set of assumptions corresponds the class of systems satisfying those assumptions. Important questions are about the computational power of these classes of systems. For example, is one class more powerful than another? What is the weakest class where a given problem is solvable? (i.e., what are the weakest assumptions which allow to solve a given problem?)

For example, with respect to the frequency of changes, we have identified the classes \( \mathcal{G}[\text{Recurrent}] \), \( \mathcal{G}[\text{Bounded}] \), and \( \mathcal{G}[\text{Periodic}] \). Let \( \mathcal{P}[\text{Recurrent}] \), \( \mathcal{P}[\text{Bounded}] \), and \( \mathcal{P}[\text{Periodic}] \) be the set of problems solvable in those classes. Clearly, \( \mathcal{P}[\text{Recurrent}] \subseteq \mathcal{P}[\text{Bounded}] \subseteq \mathcal{P}[\text{Periodic}] \). Consider the problem of minimal broadcast with termination detection: optimally diffusing some information and the source knowing (within finite time) of the completion of the process. As we discussed previously, with respect to “minimality”, there are three types of journeys and thus of broadcasts: foremost broadcast \( \text{FoB} \), in which the date of delivery is minimized at every node; shortest broadcast \( \text{ShB} \), where the number of hops used by the broadcast is minimized relative to every node; and fastest broadcast \( \text{FaB} \), where the overall duration of the broadcast is minimized (however late the departure be). Interestingly: \( \text{FoB} \in \mathcal{P}[\text{Recurrent}] \) but \( \text{ShB} \notin \mathcal{P}[\text{Recurrent}] \); furthermore \( \text{ShB} \in \mathcal{P}[\text{Bounded}] \) but \( \text{FaB} \notin \mathcal{P}[\text{Bounded}] \); on the other hand, \( \text{FaB} \in \mathcal{P}[\text{Periodic}] \). This implies first of all a strict order on the difficulty of the three problems:

\[
\text{FoB} < \text{ShB} < \text{FaB}.
\]

It also shows that the inclusion among the set of problems is strict:

\[
\mathcal{P}[\text{Recurrent}] \subsetneq \mathcal{P}[\text{Bounded}] \subsetneq \mathcal{P}[\text{Periodic}]
\]
and thus the strict hierarchy of computational power of those graph classes:

\[ \mathcal{G}[\text{Recurrent}] \prec \mathcal{G}[\text{Bounded}] \prec \mathcal{G}[\text{Periodic}] . \]

Note that these results, established in [26, 27], hold even without the discrete synchronous assumption.

It is clear that any result established under a set of assumptions gives some information about the computational impact of those assumptions. This information gives raise to a hierarchy of classes of systems, as pointed out in [28].

5 Conclusions

At the end of this note, it should be clear that researching “distributed computing in dynamic networks” means to consider a system that is (still) alive and evolving, and to understand that the computation is by necessity decentralized and localized. More precisely, the point of view is that of the computational entity, operating in an ever changing system; always in the present, the entity is only aware of the changes in which it is involved, possibly remembering the past, and in general without knowledge of the future.

The first distinction is on whether or not the computational entity has control over the topological changes in which it is involved; that is, on whether the generation is controlled or uncontrolled by the computation. In the case of controlled generation, an entity has a degree of control over the future. In the case of uncontrolled generation (a condition that occurs also in the case of mobile agents), assumptions have to be made to render computation possible. The next distinction is on what assumptions are made.

Among the open research goals the obvious ones are to establish new results and to explore new problems. An important goal is to remove assumptions. Indeed, when facing a problem the guiding question should be: what is the weakest assumption that makes the problem solvable?

The strongest assumption commonly made is that the system is discrete synchronous. Actually almost all the algorithmic investigations making this assumption further assume that the scheduling of the entities is fully synchronous: at each time steps, all the entities are active and participate in the distributed computation. A weaker assumption is that of a semi-synchronous setting: at each time step a (non-empty) subset of the entities are active, the activation choice made by a fair but adversarial scheduler. This model, quite common in
the context of robotic swarms [47], has just started to be examined in the other dynamic networks contexts [41].

A larger open research direction is to remove the discrete synchronous assumption completely, and look at the more general case possible, asynchronous and continuous. As mentioned, some investigations have been carried out and results established in the general case (e.g., [26, 27, 29]). Interestingly, in some investigations that assume discrete synchronous systems, the analysis is however carried out in the continuous setting (e.g., [65]).

References


