Random Walks in Distributed Computing: A Survey

Marc Bui[2,⋆], Thibault Bernard[1], Devan Sohier[1,2], and Alain Bui[1]

[1] Département de Mathématiques et Informatique
Université de Reims Champagne Ardennes,
BP1039 F-51687 Reims cedex, France
{alain.bui,thibault.bernard,devan.sohier}@univ-reims.fr
[2] LRIA – EPHE rue G. Lussac,
F-75005 Paris, France
marc.bui@univ-paris8.fr

Abstract. In this survey, we give an overview of the use of random walks as a traversal scheme to derive distributed control algorithms over a network of computers. It is shown that this paradigm for information exchange can be an attractive technique by using electric network theory as a mathematical tool for performance evaluation.

Keywords: Distributed System, Random Walks, Electrical Network, Distributed Algorithms

1 Introduction

We propose to use random walks in distributed computing to provide uniform and efficient solutions to distributed control of dynamic networks. Random walks have already been successfully exploited to design basic network control algorithms: self-stabilizing mutual exclusion by single token circulation [IJ90], gathering and dissemination of information over a network [AKL$^+$79], random structure over a network [BIZ89].

Indeed, we show that using accidental meetings of *circulating tokens*[1] that merges partial information of the network at each node independently gives an efficient traversal scheme for distributed control. Accidental meetings of several tokens have been ingeniously used in [IJ90] (by merging all tokens to one), in order to insure a single token circulation. It has been shown in [TW91] that all tokens should merge to one in polynomial time.

Random walks are interesting by providing a scalable mechanism to insert information into the distributed computation, for example when node insertion occurs in the distributed system or to update topology modification (edge or node deletion). Ad-hoc networks or pervasive distributed systems, because of their very limited communication bandwidth for network control, can also benefit of this approach.

Because of their inherent complexity, deterministic solutions to control large distributed systems are often unsatisfactory. One solution is to design randomized algorithms which can be simpler, especially for their correctness proof. Precise statement about their performance are, on the other hand, an interesting challenge: by introducing electrical networks, we present an original and elegant manner to compare solutions.

⋆ Corresponding author

[1] A data structure moving following a random walk scheme in the network

2 Preliminaries

In this section we present the motivation for using random walk in distributed computing, and a formal description of a random walk.

2.1 Distributed Systems

We define a distributed system to be a set of autonomous computing resources, exchanging messages via communication links. The system is modeled as a graph, the nodes of which represent the computers and the edges the communication channels.

We adopt the classical asynchronous message passing model, *i.e.* computers communicate by sending messages to their neighbors, and there is no bound on the time it takes to a message to reach its goal.

2.2 Random Walks Characteristics

A *random walk* is a sequence of vertices visited by a token that starts at i and visits other vertices according to the following transition rules: if the token is at i at time t then at time $t + 1$ it will be at one of the neighbors of i, this neighbor having been chosen according to some time-constant law. Various papers deal with random walks e.g. [Lov93,AKL$^+$79]. More formally, a random walk is a finite homogeneous Markov Chain with state set V and with transition probability matrix $P = (p_{ij})_{(i,j)\in V^2}$ given by

$$p_{ij} = \begin{cases} \frac{1}{\deg(i)} & \text{if } (i,j) \in E \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

where $\deg(i)$ is the degree of node i (*i.e.* the number of its neighbors).

Let P^t the t^{th} power of P, whose entries are $p_t(i,j),\ (i,j) \in V^2$.

Since G is connected, if it is not bipartite, the Markov Chain has only one acyclic ergodic class of states, then $\lim_{t\to\infty} P^t$ exists and is a matrix Q with identical rows $\pi = (\pi_i,\ i \in V)$, *i.e.* $\forall (i,j) \in V \times V, \lim_{t\to\infty} p_t(i,j) = \pi_i$. π is the stationary distribution and can be computed such that $\pi = \pi.P$. Note that, in the particular case of random walks, the stationary distribution satisfies

$$\pi_i = \frac{\deg(i)}{2|E|} \tag{1}$$

Some characteristic values are useful in the context of distributed computing. The mean time to reach vertex j (state j), starting from the vertex i (state i) which may be regarded as the conditional expectation of the random number of transitions before entering j for the first time when starting at i, is called *hitting time* and denoted h_{ij}. In particular, we have $h_{ii} = 1/\pi_i$. We often use the quantity $\max\{h_{ij}/j \in V\}$, which is an upper bound for a random walk starting at i to hit a fixed, but unknown vertex, for example, when the average time to look for an information owned by a unknown vertex is required. $h_{ij} + h_{ji}$ called the commute time, is the expected number of steps for a random walks starting at vertex i to reach vertex j for the first time and reach i again. It can be viewed as the average time to fetch back to i an information owned by the vertex j. The expected time for a random walk starting at i to visit all the vertices of the graph is called the *cover* *time* C_i. Let $C = \max\{C_i / i \in V\}$. C_i will be the average time needed by i to build a spanning tree thanks to the algorithm described above. C will be an upper bound of the average time for an unknown vertex to build a spanning tree thanks to the algorithm described above. Results on bounds on cover time can be found in [Lov93].

3 Random Walks and Network Structuration

In [Bro89], the author computes a random spanning tree by using a random walk in a graph. [BIZ89], applies this method to compute a spanning tree over a network. As they mention, a random spanning tree is more resilient to link failures: *"the probability that a bad channel will disconnect some nodes from the random tree is relatively small"*.

In [IJ90], random walks have been used as the design of a self-stabilizing algorithm for the mutual exclusion problem. In their system, a token typifies the privilege to execute critical section code. This token moves with a random walk policy. If several tokens are present in the network, they eventually meet on a site and the protocol will make them merge. [TW91] shows that the protocol developed in [IJ90] stabilizes in polynomial time (this is an average complexity).

In our works, we use random walks to structure a network. Many applications can use this structuration: data transfer in peer-to-peer network, job dispatching in GRID computing... We give here the main idea of the algorithms except in Section 3.3 where we give full specifications of the algorithm.

3.1 Routing with Mobile Agents

In [BDDN01], mobile agents are used to update the shortest path routing table in each site. Two protocols are presented. In the first protocol, there is no interaction between agent. Each site dispatches a mobile agent for a random walk. Each agent carries (and possibly updates) link state of its creator and updates routing tables of visited sites through the network. In the second protocol (the cooperative one), agent caries not only the links states of its creator but the routing table (updated each time the agent returns at home). This protocol increases the convergence of routing table, but increases the size of all agents. The aim of using random walk is to easily manage topological changes. If a channel becomes unavailable, the agent when returning home, is updated. If a new site get connected, it launches an agent with its links state (routing table) and thus updates other routing tables.

3.2 Random Spanning Tree Construction for Shortest Path Routing Computation

In [Fla01], the author presents how to collect topological informations by the use of a random walk, and more precisely a random circulating word, and a method to deduce a spanning tree of the network from the collected topological informations. That deduced spanning tree is not applied to the entire network, but locally stored in each node of the network. Then he gives original algorithms to reduce the size of the collected information without loss of the needed information for the spanning tree construction. At last, he gives a possible application of that "local" tree construction : the creation of shortest paths routing tables.

3.3 Network Design with Distributed Random Walks

In [BFG+03], we address the problem of constructing such a structure with a protocol that tolerates faults and adapts itself to dynamic topology changes that often occur in mobile ad hoc networks. Tolerating faults is crucial in such networks where mobile devices get frequently out of range or powered off by users. Therefore, we introduce distributed random walk (DRW) as a collection of random walks (RWs) that cooperate in order to establish a computation. The technique uses a collection of RWs that are coalescing into a final one that maintains the control structure. We apply this technique to compute a spanning tree (ST), which is selected uniformly at random among all possible ones for a network. To gather informations, we use a wave scheme. We can informally describe the whole procedure as follows: several mobile devices initiate a RW, with an explorer agent. Every mobile device, upon receiving an explorer token, marks itself visited with the identity of the token, except if it has already been visited by another token. It then forwards at random to one of its neighbors the received explorer token. The network is thus, explored in parallel and decomposed into sub-regions, one per token. Each token constructs a sub-tree of the network. When a token meets another one, or an already visited mobile device, a wave is initiated. This wave is a backward propagation wave that merges the two sub-trees. This process is conducted in parallel and, eventually, the waves will cover the network, resulting in the ST definition. The protocol is ready for termination when a single explorer token remains and all mobile devices of the network have been visited.

Description of the self-stabilizing algorithm for spanning tree construction. We briefly present here the specifications of the algorithm.

Each node maintains:

- $color$, the identity of an agent
- $master$, the (sub)tree root to which the node belongs
- $parent$, the node parent within the (sub)tree
- $sons$, the set of sons of the node

An agent is composed of two fields: a color and a root. When two agents are merging on a site, the site have to decide values affected to the fields. So we need a rule to compare agents:

- $T_1 > T_2$ if $(T_1.color > T_2.color) \vee (T_1.color = T_2.color \wedge T_1.root > T_2.root)$
- $T_1 = T_2$ if $(T_1.color = T_2.color) \wedge (T_1.root > T_2.root)$

On timeout, a site flips a coin, generate an agent by a color and the node identity, sends, $local\_state$ to all neighbors. Once all neighbors' $local\_state$ received, if $test\_validity\_state$ is not correct $reset(node)$.

We specify the algorithm behavior by means of overall actions driven by agents and waves. Some sites randomly generate an agent identified by a color that is characterized by the initiators of the agent.

Agent Annexing Mode. Whenever an agent $a_i(color_i, rac_i)$ issued from a node q is annexing (or generated at) node p, which belongs to a (sub)tree (i.e. an agent $a_j(color_j, rac_j)$).

AA1 if $color_i < color_j$, the annexing is stopped and the agent is destroyed.
AA2 else (if $color_i > color_j$), one of the 2 conditions holds:
i One (or more) agent(s) are present on node p
- if agent is the unique biggest, it continues its traversal and all others are destroyed. Node p marked himself with $color \longleftarrow color_i, master \longleftarrow rac_i, parent \longleftarrow q$
- if agent a_i is the biggest but not unique (others agents $a_{j1}, \ldots,$ a_{jd} that have respectly $color_{j1}, \ldots, color_{jd}$ equal to $color_i$, agents $a_i, a_{j1}, \ldots, a_{jd}$ are merged to form the unique agent of identity $i + 1$ rooted in p (i.e. agent $a(i + 1, p)$ is generated).
- if agent a_i is not the biggest, it is destroyed.

ii No other agent on node p. Agent continues its traversal scheme. p marks himself with $color \longleftarrow color_i, master \longleftarrow rac_i, parent \longleftarrow q$

Wave Update Mode. Whenever an agent $a_i(color_i, rac_i)$ reaches a node p with its variable color such that $color < color_i$ a wave is generated.
WU1 The wave is propagated applying a path reversal scheme over the domain identified by color (the domain which p belongs to)
WU2 The wave stops itself when it reaches the p limit.

Termination of the algorithm is realized with a derivation from the Dijkstra-Scholten scheme known as diffusing computation. This termination detection is periodically initiated by nodes that have initiated an annexing agent.

For proof of correctness and stabilization, refer to [BFG$^+$03]

3.4 Random Self-stabilizing Structures

In [BBF04], we use a random walk for the self-stabilizing construction of a collection of spanning tree rooted in each site. As in [Fla01] this computation is achieved using a circulating word which collects the identity of visited site. Each time a site gets the token, it updates its local spanning tree with the one contained in the word. To stabilize the computation, we need to insure the stabilization of the content of the word (for stabilizing tree construction) and the presence of only one circulating word. We correct the word step by step in each site by processing to an internal test that detects local inconsistencies thanks to the neighborhood of the current visited site. A site at the reception of the token, checks the neighborhood relation declared in the word correspond to the neighborhood of the site. The presence of a token is insured by a timeout process on each site that eventually creates new tokens. The merger of the tokens insures the decrease of the number of tokens to one (by the same process as described in [IJ90]). If several tokens hold on a site at the moment, the site would merge them into one, merging topological information. This algorithm is well adapted for unsafe dynamic networks.

3.5 Conclusion on Structuration

We have a way to construct routing tables ([BDDN01,Fla01]) and adaptative structures ([BBF04] for small scale networks. In [BFG$^+$03], we propose another approach for network structuration : distributed random walks. Our future intend to join the two approaches in order to construct control structures for large scale networks.

4 Random Walks and Electrical Networks for the Complexity of Distributed Algorithms

Random walks offer a pleasant framework to design distributed algorithms in a dynamic context or with self-stabilization. However, evaluating the complexity of those algorithms requires the use of many probabilistic tools. It is important to note that, except in very particular case, there is no hope to give hard bound to the complexity of a random walk based distributed algorithm. Indeed, due to their probabilistic nature, there is no way to guarantee that a walk reaches a given site in a given number of steps.

Some quantities are very useful in the evaluation of the complexity of random walk based distributed algorithms. The hitting time h_{ij} is the average number of steps it takes to a random walk starting at i to first reach j. The commute time is the average time for a round-trip from i to j: $\kappa_{ij} = h_{ij} + h_{ji}$. The cover time C_i is the average number of steps for a random walk starting at i to visit all the sites in the network. The cover time $C = \max\{C_i / i \in V\}$.

Random walks and electrical current have in common two properties: the amount of them that enters a node must leave it; it leaves a node through a channel, proportionally to a time-constant quantity attached to this channel (weight or probability for random walks; conductance for the current). This similarity entails a tight link between random walks and resistive networks, detailed in [DS00,CRR$^+$97], for example. In the sequel, we focus on resistive networks.

The conductance is the inverse of the resistance. The effective conductance $\mathcal{C}_{ij}$ of a network, between two of its nodes i and j is the conductance of the resistor to be placed between i and j to ensure the same electrical properties. In other words, it is the current that flows from i to j when a potential difference of $1V$ is imposed between i and j.

In [BBS03,BBBS03], we propose a method based on resistances to automatically compute hitting times, which provides with an evaluation of the complexity of some random walks based distributed algorithms. Our method use the relationship between electrical resistance and random walks established in [DS00,CRR$^+$97]. We provide an automatic way to compute resistances on graphs modeling distributed networks.

A network topology may be viewed as an electrical network (both may be viewed as an undirected graph). Each link (edge) (i,j) is assigned a real value: the resistance $r(i,j)$. Interesting results can be deduced by application of the two Kirchhoff's law and Ohm's law. In particular, through the notion of effective resistance (explained below), we present an innovative method to automatically compute bounds on cover time, exact values of commute time and hitting times, exact values of cyclic cover time and total hitting time for any arbitrary graph.

4.1 Review of Basic Ideas on Electricity

Relation between the potential difference between two nodes, the electrical resistance and the electrical current are given by Ohm's law: $U = R \times I$. Kirchhoff's law states that the sum of currents entering and exiting a node must equal zero. For two resistors (or resistive networks) R_1 and R_2 connected *in series*, we have: $R_{global} = R_1 + R_2$. For two resistors (or resistive networks) R_1 and R_2 connected *in parallel*, we have: $\frac{1}{R_{global}} = \frac{1}{R_1} + \frac{1}{R_2}$.

4.2 Effective Resistance

$R(i,j)$ is the effective resistance (resistance of the electrical network) between i and j, if we replace each edge in the graph by a 1Ω resistor. R denotes the maximal effective resistance between two nodes of the network *i.e.* $R = \max_{(i,j)\in V^2} R(i,j)$.

4.3 Previous Results

A tight relationship between resistances in electric networks and random walks characteristic values such as commute times and cover time, has been established in [CRR+97]. In particular, it has been shown that:

$$\kappa_{ij} = h_{ij} + h_{ji} = 2mR(i,j) \tag{2}$$

where i and j denote two distinct vertices and m, the number of vertices.
From this equation, we have:

$$mR < C < O(mR \log n) \tag{3}$$

In [Tet91], hitting time are expressed only in terms of resistances:

$$h(i,j) = mR(i,j) + \frac{1}{2}\sum_{k\in V} deg(k)[R(j,k) - R(i,k)] \tag{4}$$

4.4 Millman's Theorem

Our method uses the Millman's theorem to compute automatically effective resistance. As shown above, we can compute, thanks to resistances, bounds on cover time, exact values of cyclic cover time, exact values of commute time and total hitting time for any arbitrary graphs.

Theorem 1 (Millman's theorem). *Consider an electrical network, on any node i, we have the following relation:*

$$V_i = \frac{\sum_{j=0,}^{k} \frac{V_j}{r(i,j)}}{\sum_{j=1}^{k} \frac{1}{r(i,j)}}$$

that is

$$\frac{V_i - V_0}{r(i,0)} + \frac{V_i - V_1}{r(i,1)} + \frac{V_i - V_2}{r(i,2)} + \cdots + \frac{V_i - V_k}{r(i,k)} = 0$$

where $1,\cdots,k$ are the neighbors of i, $V_1,\cdots,V_k$ are the voltages on each of these nodes.

By giving the potentials on the two nodes i and j connected to the generator, say $V_i = 0$ and $V_j = 1$, we obtain a single solution. The matrix of the system is obtained from the matrix of the walk by replacing the line i and j by the null line and $M_{ii} = M_{jj} = 1$. Then, we can compute the potential on all the nodes, and the intensity I going out of i. The effective resistance R_{ij} is then $1/I$.

This method is not efficient, since we have to inverse n^2 matrices to obtain all the resistances and hitting times.

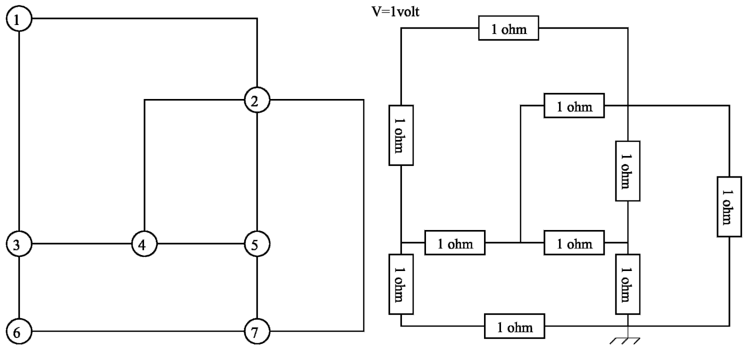

Fig. 1. Example

Example. Let apply Millman theorem to the graph on fig. 1, to compute R_{17}.

Then $V_1 = 1$, $V_7 = 0$.

$$\begin{aligned} V_1 - V_2 + V_4 - V_2 + V_5 - V_2 + V_7 - V_2 &= 0 \\ V_1 - V_3 + V_4 - V_3 + V_6 - V_3 &= 0 \\ V_2 - V_4 + V_3 - V_4 + V_5 - V_4 &= 0 \\ V_2 - V_5 + V_4 - V_5 + V_7 - V_5 &= 0 \\ V_3 - V_6 + V_7 - V_6 &= 0 \end{aligned} \tag{5}$$

From (5), we obtain $V_6 = \frac{2}{7}$, $V_4 = \frac{3}{7}$, $V_5 = \frac{2}{7}$, $V_3 = \frac{4}{7}$ and $V_2 = \frac{3}{7}$. The intensity going out of 1 is $\frac{V_2}{1\Omega} + \frac{V_3}{1\Omega} = 1$, and $R_{17} = 1$. Thus, $h_{17} + h_{71} = 10 \times 1 = 10$.

We can design a more efficient method by noting that we inverse very similar matrices.

Let V_k be the potential on the node k in the graph G. According to the Kirchoff's current law, when a 1A current flows from i to j:

$$\begin{cases} \forall k \in V \backslash \{i, j\}, \sum_{l \in N(k)} c_{kl}(V_k - V_l) = 0 \\ \sum_{l \in N(i)} c_{il}(V_i - V_l) = 1 \\ \sum_{l \in N(j)} c_{jl}(V_j - V_l) = -1 \end{cases}$$

This can be written:

$$\Delta V = v$$

with Δ the matrix built from the conductance matrix by letting the entry $(k; k)$ be $-\sum_{l \in N(k)} c_{kl}$, and v the vector with all entry 0 except the i-th one 1 and the j-th one -1.

However, Δ is not inversible, since the vector $(1, \ldots, 1)$ is in its kernel: the potential is defined up to a constant. The kernel is $(1, \ldots, 1) \times \mathbb{R}$, for if there were other vectors, there would be several steady states in this circuit with one given generator connected to two given nodes, which is false according to electricity laws. Thus, the matrix Δ_2 built by replacing the first line in Δ by $(1, 0, \ldots, 0)$ is inversible, for its electrical interpretation is that the potential on node 1 is given and the Millman's theorem applies on each other node (and the network is connected). Thus, $\Delta_2^{-1} v$ is a solution to $\Delta V = v$.

Knowing the potential on each node, we can compute the effective resistance of the network between i and j: $V_j - V_i$. The effective resistance between i and j is $\Delta_2^{-1}(i,j) - \Delta_2^{-1}(j,j) - \Delta_2^{-1}(i,i) + \Delta_2^{-1}(j,i)$. The hitting times can then be computed thanks to the above formula (4).

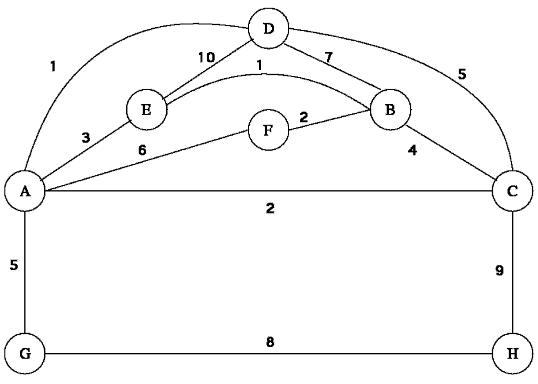

Fig. 2. Example

Thus, this method works with only one matrix inversion, and with no matrices multiplication, which is much better than the previous results we published ([BBS03]). We improved the speed of those computations with a factor of about one hundred, making it possible to apply this on large graphs.

For the graph on figure 2, the matrix Δ_2 is:

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -14 & 4 & 7 & 1 & 2 & 0 & 0 \\
2 & 4 & -20 & 5 & 0 & 0 & 0 & 9 \\
1 & 7 & 5 & -23 & 10 & 0 & 0 & 0 \\
3 & 1 & 0 & 10 & -14 & 0 & 0 & 0 \\
6 & 2 & 0 & 0 & 0 & -8 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & -13 & 8 \\
0 & 0 & 9 & 0 & 0 & 0 & 8 & -17
\end{pmatrix}$$

Δ_2^{-1} is:

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -\frac{102091}{627268} & -\frac{56049}{627268} & -\frac{67339}{627268} & -\frac{110783}{1254536} & -\frac{102091}{2509072} & -\frac{6426}{156817} & -\frac{41769}{627268} \\
1 & -\frac{56049}{627268} & -\frac{84309}{627268} & -\frac{53851}{627268} & -\frac{84937}{1254536} & -\frac{56049}{2509072} & -\frac{9666}{156817} & -\frac{62829}{627268} \\
1 & -\frac{67339}{627268} & -\frac{53851}{627268} & -\frac{89297}{627268} & -\frac{137187}{1254536} & -\frac{67339}{2509072} & -\frac{6174}{156817} & -\frac{40131}{627268} \\
1 & -\frac{110783}{1254536} & -\frac{84937}{1254536} & -\frac{137187}{1254536} & -\frac{391027}{2509072} & -\frac{110783}{5018144} & -\frac{4869}{156817} & -\frac{63297}{1254536} \\
1 & -\frac{102091}{2509072} & -\frac{56049}{2509072} & -\frac{67339}{2509072} & -\frac{110783}{5018144} & -\frac{1356627}{10036288} & -\frac{3213}{313634} & -\frac{41769}{2509072} \\
1 & -\frac{6426}{156817} & -\frac{9666}{156817} & -\frac{6174}{156817} & -\frac{4869}{156817} & -\frac{3213}{313634} & -\frac{21413}{156817} & -\frac{15194}{156817} \\
1 & -\frac{41769}{627268} & -\frac{62829}{627268} & -\frac{40131}{627268} & -\frac{63297}{1254536} & -\frac{41769}{2509072} & -\frac{15194}{156817} & -\frac{98761}{627268}
\end{pmatrix}$$

The resistance matrix is:

$$\begin{pmatrix}
0 & \frac{102091}{627268} & \frac{84309}{627268} & \frac{89297}{627268} & \frac{391027}{2509072} & \frac{1356627}{10036288} & \frac{21413}{156817} & \frac{98761}{627268} \\
\frac{102091}{627268} & 0 & \frac{37151}{313634} & \frac{28355}{313634} & \frac{356259}{2509072} & \frac{2173355}{10036288} & \frac{136335}{627268} & \frac{58657}{313634} \\
\frac{84309}{627268} & \frac{37151}{313634} & 0 & \frac{16476}{156817} & \frac{388515}{2509072} & \frac{2257179}{10036288} & \frac{92633}{627268} & \frac{14353}{156817} \\
\frac{89297}{627268} & \frac{28355}{313634} & \frac{16476}{156817} & 0 & \frac{199467}{2509072} & \frac{2246667}{10036288} & \frac{125557}{627268} & \frac{26949}{156817} \\
\frac{391027}{2509072} & \frac{356259}{2509072} & \frac{388515}{2509072} & \frac{199467}{2509072} & 0 & \frac{2477603}{10036288} & \frac{577827}{2509072} & \frac{532883}{2509072} \\
\frac{1356627}{10036288} & \frac{2173355}{10036288} & \frac{2257179}{10036288} & \frac{2246667}{10036288} & \frac{2477603}{10036288} & 0 & \frac{2521427}{10036288} & \frac{2602651}{10036288} \\
\frac{21413}{156817} & \frac{136335}{627268} & \frac{92633}{627268} & \frac{125557}{627268} & \frac{577827}{2509072} & \frac{2521427}{10036288} & 0 & \frac{62861}{627268} \\
\frac{98761}{627268} & \frac{58657}{313634} & \frac{14353}{156817} & \frac{26949}{156817} & \frac{532883}{2509072} & \frac{2602651}{10036288} & \frac{62861}{627268} & 0
\end{pmatrix}$$

and the hitting times matrix is:

$$\begin{pmatrix}
0 & \frac{6740527}{627268} & \frac{4636113}{627268} & \frac{5079563}{627268} & \frac{6844259}{627268} & \frac{68203479}{5018144} & \frac{1655067}{156817} & \frac{6683885}{627268} \\
\frac{6122939}{627268} & 0 & \frac{1848439}{313634} & \frac{679447}{156817} & \frac{5987869}{627268} & \frac{91460059}{5018144} & \frac{9504503}{627268} & \frac{3771965}{313634} \\
\frac{5986821}{627268} & \frac{2832587}{313634} & 0 & \frac{2140579}{313634} & \frac{7480049}{627268} & \frac{101973699}{5018144} & \frac{7735425}{627268} & \frac{1188563}{156817} \\
\frac{6171859}{627268} & \frac{1106918}{156817} & \frac{2011373}{313634} & 0 & \frac{4373337}{627268} & \frac{100608923}{5018144} & \frac{9680431}{627268} & \frac{3899619}{313634} \\
\frac{10946183}{1254536} & \frac{10468579}{1254536} & \frac{9516347}{1254536} & \frac{3819747}{1254536} & 0 & \frac{98029553}{5018144} & \frac{19278767}{1254536} & \frac{16338531}{1254536} \\
\frac{8632011}{2509072} & \frac{22730653}{2509072} & \frac{20114289}{2509072} & \frac{20465549}{2509072} & \frac{14514859}{1254536} & 0 & \frac{31874379}{2509072} & \frac{30104657}{2509072} \\
\frac{1042971}{156817} & \frac{7673707}{627268} & \frac{3936333}{627268} & \frac{6139751}{627268} & \frac{8562167}{627268} & \frac{95101143}{5018144} & 0 & \frac{3197993}{627268} \\
\frac{5760001}{627268} & \frac{3618817}{313634} & \frac{619915}{156817} & \frac{2891529}{313634} & \frac{8616549}{627268} & \frac{103757699}{5018144} & \frac{4722493}{627268} & 0
\end{pmatrix}$$

The cover time can be expressed in terms of hitting times. To compute the cover time, we need a criterion to determine whether every vertex has been visited by the token. So, a state of the system will be the site the token is currently visiting, but also the set of sites it has visited.

Consider $G = (V, E, \omega)$ the undirected connected weighted graph modeling a distributed system. To formalize the idea above, we build from G an associated graph $\mathcal{G}$ such that the cover time of G may be expressed in terms of hitting times in $\mathcal{G}$.

First let define $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \omega_2)$ where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ a set of directed edges.

- $x \in \mathcal{V}$ is defined by $x = (P, i)$ with $P \in \mathcal{P}(V)$ (*i.e.* P is a subset of nodes of G, representing the nodes in G already visited), and $i \in V$ is a node.
- any edge $(x, y) \in \mathcal{E}$ is of the form $(x, y) = ((P, i), (Q, j))$ with $(x, y) \in \mathcal{V} \times \mathcal{V}$ and $(i, j) \in E$ (is an edge).

Suppose that, initially, the token is at node i in G, and next the token moves to j neighbor of i, and next moves back to i. For the associated graph $\mathcal{G}$, we have the following path $((\{i\}, i); (\{i, j\}, j); (\{i, j\}, i))$.

Note that $\mathcal{E}$ is a set of *directed* edges $((P, i), (Q, j))$. Edges in $\mathcal{E}$ are defined by:

- $((P, i), (P, j))$, where i and j are neighbors ; this case corresponds to a token transmission to the node j that has already been visited by the token.
- $((P, i), (P \bigcup \{j\}, j))$ where i and j are neighbors ; this case corresponds to a token transmission to the node j that is holding the token for the first time.

If $x = (P, i)$ and $y = (Q, j)$ in $\mathcal{G}$ are neighbours, $\omega_2(x, y) = \omega(i, j)$.

The probability to obtain a given path in G is equal to the probability to obtain the associated path in $\mathcal{G}$. Indeed, for $i \in P \subset V$ and $j \in V$, there exists some $Q \subset V$ such that the transition probability from (P, i) to (Q, j) and the transition probability from i to j are the same: $Q = P$ if $j \in P$, else, $Q = P \bigcup \{j\}$.

A token in G has visited every node *iff* the associated token in $\mathcal{G}$ has reached a node (P, i) such that $P = V$. Then, we deduce that the cover time in G is the average time it takes to a random walk token in $\mathcal{G}$ starting from a node i to reach any arbitrary node k while having visited all vertices that is

$$C_i(G) = h_{(\{i\}, i), \{(V, k)/k \in V\}}(\mathcal{G})$$

The token has covered G when the associated token in $\mathcal{G}$ has hit any vertex in $F = \{(V,k)/k \in V\}$. We do not care at which node (V,k) the token reaches in $\mathcal{G}$, then we lump all nodes F into a single one called f (in fact we obtain an absorbing Markov Chain). Now, the cover time in G is obtained by the average number of steps needed before entering f starting in node $(\{i\},i)$.

Indeed, let $\mathcal{N}_o(x)$ be the set of vertices that have an incoming edge from x: $\{y \in \mathcal{V}/(x,y) \in \mathcal{E}\}$.

Now, in any graph (even directed), since f can be reached from any vertex (if not, some of the h_{xf} would be undefined), with $p_{xy} = \frac{\omega(x,y)}{\sum_{n \in \mathcal{N}(x)} \omega(x,n)}$:

$$\begin{cases} \forall x \in V, h_{xf} = 1 + \sum_{y \in \mathcal{N}_o(x)} p_{xy} h_{yf} \\ h_{ff} = 0 \end{cases} \tag{6}$$

6 is a square linear system that has a single solution (since it is square and has at least one solution: the vector $h_{.f}$, which we know to exist!): one can compute the hitting time between all vertices and a given vertex by inverting one matrix.

Thus, we can compute the cover time of any graph G by building $\mathcal{G}$ and computing $h_{(i,\{i\}),f}(\mathcal{G})$, which requires the inversion of an approximatively $2^n \times 2^n$ matrix.

Let G be the graph on figure 3. Then $\mathcal{G}$ is the graph on figure 4.

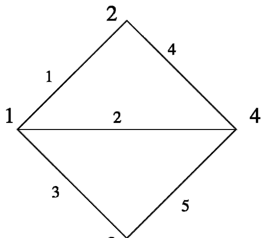

Fig. 3. G

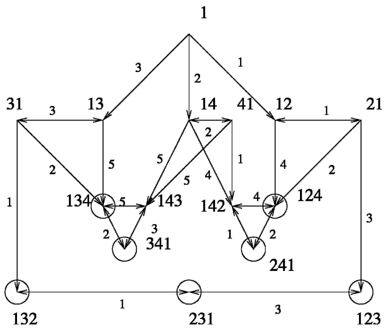

Fig. 4. $\mathcal{G}$

In figure 4 we named sites by giving the set of known vertices, the last mentionned being the current one. We only built the part of $\mathcal{G}$ that corresponds to situations where the token started on vertex 1. We did no write the states in which all vertices are visited: for the sake of legibility, we circled the sites that lead to such a state. Thus, in state 134, the token will reach 2 and finish to cover the graph with probability $\frac{1}{3}$, reach 3 (the state being 143) or 1 (341) also with probability $\frac{1}{3}$.

Since we merge all the states in which the token has covered the graph, every circled state leads to the new site f with an unweighted directed vertex.

The matrix of $\mathcal{G}$ is then:

$$\left(\begin{array}{r|ccccccccccccccccc}
 & 1 & 12 & 14 & 13 & 21 & 124 & 41 & 142 & 143 & 31 & 134 & 123 & 132 & 241 & 341 & 231 & f \\
\hline
1 & 0 & 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
12 & 0 & 0 & 0 & 0 & 1 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
14 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 4 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\
21 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\
124 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 5 \\
41 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 1 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
142 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
143 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 3 & 0 & 0 \\
31 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 \\
134 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 4 \\
123 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 5 \\
132 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 4 \\
241 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\
341 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 1 \\
231 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 & 0 & 0 & 2 \\
f & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right)$$

To obtain the cover time, we have to multiply the inverse of the below matrix by $(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0)$:

$$\begin{pmatrix}
1 & -\frac{1}{6} & -\frac{1}{3} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & -\frac{1}{5} & -\frac{4}{5} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & -\frac{2}{11} & -\frac{4}{11} & -\frac{5}{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -\frac{3}{8} & -\frac{5}{8} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{1}{6} & 0 & 0 & 1 & -\frac{1}{3} & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -\frac{4}{11} & 0 & 0 & 0 & 0 & 0 & -\frac{2}{11} & 0 & 0 & -\frac{5}{11} \\
0 & 0 & -\frac{1}{3} & 0 & 0 & 0 & 1 & -\frac{1}{6} & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -\frac{4}{5} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{5} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -\frac{5}{8} & 0 & 0 & 0 & -\frac{3}{8} & 0 & 0 \\
0 & 0 & 0 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 1 & -\frac{1}{3} & 0 & -\frac{1}{6} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{5}{11} & 0 & 1 & 0 & 0 & 0 & -\frac{2}{11} & 0 & -\frac{4}{11} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -\frac{3}{8} & -\frac{5}{8} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -\frac{1}{5} & -\frac{4}{5} \\
0 & 0 & 0 & 0 & 0 & -\frac{1}{3} & 0 & -\frac{1}{6} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -\frac{1}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & 0 & -\frac{1}{3} & 0 & 0 & 0 & 1 & 0 & -\frac{1}{6} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{2} & -\frac{1}{6} & 0 & 0 & 1 & -\frac{1}{3} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

By solving this system, we obtain that:

$$h_{.f}(\mathcal{G}) = \left(\frac{15154068}{2185469}, \frac{21601}{5423}, \frac{37223}{5797}, \frac{15217}{2431}, \frac{19062}{5423}, \frac{533}{187}, \frac{39698}{5797}, \frac{710}{187}, \frac{1150}{187}, \frac{1321}{221}, \frac{903}{187}, \frac{337}{187}, \frac{267}{187}, \frac{483}{187}, \frac{1063}{187}, \frac{400}{187}, 0\right)$$

Since the first entry is the average time it takes to the token to first reach any state in which all vertices are known, starting from the state in which no site is known and the token is on vertex 1, the cover time is

$$C_1 = h_{1f}(\mathcal{G}) = \frac{15154068}{2185469}$$

that is to say about 7.

5 Conclusion

Peer-to-peer protocols and wireless connections are two important and widespread examples of dynamic distributed systems. In such systems, the lack of knowledge of the topology of the system is the main problem. Current solutions are based on a control layer, that oversees the connections and disconnections, and updates the the routing tables and all the required topological information accordingly. Such a control layer consumes an important amount of bandwidth, which is problem in systems the bandwidth of which is limited.

Random walks offer an interesting alternative to deterministic control layers. Random walks based deterministic algorithms can provide a completely distributed framework to handle the dynamicity of networks, with a lesser consumption of bandwidth. However, the complexity of those solutions, being based on randomized procedures, is difficult to estimate. The main quantities can now be computed, which allows to compare random walk based algorithms to deterministic ones.

References

AKL$^+$79. R. Aleliunas, R. Karp, R. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal traversal sequences and the complexity of maze problems. In *20th IEEE Annual Symposium on Foundations of Computer Science*, pages 218–223, October 1979.

BBBS03. T Bernard, A Bui, M Bui, and D Sohier. A new method to automatically compute processing times for random walks based distributed algorithm. In Marcin Paprzycki, editor, *ISPDC 03, Second IEEE International Symposium on Parallel and Distributed Computing Proceeding*, volume 2069, pages 31–36. IEEE Computer society Press, 2003.

BBF04. T Bernard, A Bui, and O Flauzac. Random distributed self-stabilizing structures maintenance. In Victor Larios Félix F. Ramos, Herwig Unger, editor, *ISADS'04, IEEE Internationnal Symposium on Advanced Distributed Systems Proccedings*, volume 3061, pages 231 – 240. Springer Verlag (LNCS), January 2004.

BBS03. A Bui, M Bui, and D Sohier. Randomly distributed tasks in bounded time. In *Innovative Internet Community System*, volume 2877, pages 36–47. Springer Verlag (LNCS), 2003.

BDDN01. Marc Bui, Sajal K. Das, Ajoy Kumar Datta, and Dai Tho Nguyen. Randomized mobile agent based routing in wireless networks. *International Journal of Foundations of Computer Science*, 12(3):365–384, 2001.

BFG$^+$03. H Baala, O Flauzac, J Gaber, M Bui, and T El-Ghazawi. A self-stabilizing distributed algorithm for spanning tree construction in wireless ad-hoc network. *Journal of Parallel and Distributed Computing*, 63(1):97–104, 2003.

BIZ89. J Bar-Ilan and D Zernik. Random leaders and random spanning trees. In *WDAG89*, pages 1,12, 1989.

Bro89. AZ Broder. Generating random spanning trees. pages 442–447. FOCS89 Proceedings of the 29st annual IEEE Symposium on foundation of computer sciences, 1989.

CRR$^+$97. Ashok K. Chandra, Prabhakar Raghavan, Walter L. Ruzzo, Roman Smolensky, and Prasoon Tiwari. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity*, 6(4), 1997.

DS00. Peter G. Doyle and J. Laurie Snell. *Random Walks and Electric Networks*. 2000.

Fla01. O Flauzac. Random circulating word information management for tree construction and shortest path routing tables computation. In *On Principle Of DIstributed Systems*, pages 17–32. Studia Informatica Universalis, 2001.

IJ90. A Israeli and M Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *PODC90, Proceeding of the ninth ACM Annual Symposium on Principles of distributed Computing*, pages 119–131, 1990.

Lov93. Laszlo Lovasz. Random walks on graphs: A survey. In T. Szonyi ed. D. Miklos, V. T. Sos, editor, *Combinatorics: Paul Erdos is Eighty (vol. 2)*, pages 353–398. Janos Bolyai Mathematical Society, 1993.

Tet91. P Tetali. Random walks and effective resistance of networks. *J. Theoretical Probability*, 1:101,109, 1991.

TW91. P Tetali and P Winkler. On a random walk arising in self-stabilizing token management. In *PODC91, Proceeding of the tenth ACM Annual Symposium on Principles of distributed Computing*, pages 273–280, 1991.