Probabilistic Methods for Routing in Wireless Sensor Networks

Karine Altisen, Stéphane Devismes, Pascal Lafourcade, Clément Ponsonnet

Verimag Research Report n^o TR-2010-18

October 13, 2010

Reports are downloadable at the following address
`http://www-verimag.imag.fr`

Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Probabilistic Methods for Routing in Wireless Sensor Networks

Karine Altisen, Stéphane Devismes, Pascal Lafourcade, Clément Ponsonnet

October 13, 2010

Abstract

In this report we improve routing algorithms mixing random walk and tabu list. We propose 4 new routing protocols based on such ideas. We compare the perfomances of our algoritms using a simulator. For ours tests we propose several classes of network topology in order to show that there is not a best algorithm, it means that for all our algorithms we can construct a topology for which each routing algorithm is not the best one.

Keywords: Routing protocols, Random Walk, Tabu list

Reviewers: Laurent Mounier

Notes: This report is based on the Master thesis realised by Clément Ponsonnet

How to cite this report:

```
@techreport {ADPP-RR-2010,
  title = {Probabilistic Methods for Routing in Wireless Sensor Networks},
  author = {Karine Altisen, Stéphane Devismes, Pascal Lafourcade, Clément Ponsonnet },
  institution = {{Verimag} Research Report},
  number = {TR-2010-18},
  year = {2010}
}
```

Contents

Introduction 3

I Comparison of Different Methods of Probabilistic Routing 9

1 Standard Random Walk *vs.* Random Walk using Local Degrees 11
1.1 Algorithms . . . 11
1.1.1 Standard Random Walk . . . 11
1.1.2 Random Walk Using Local Degree Information . . . 12
1.2 Simulations . . . 12
1.2.1 Improvements . . . 12
1.2.2 Counter-examples . . . 14
1.3 Formal results . . . 16
1.3.1 First counter-example . . . 16
1.3.2 Second counter-example . . . 16

2 Random Walk *vs.* Random Walk with Tabu Lists in Messages 19
2.1 Algorithms . . . 19
2.2 Minimizing the Tabu List Size by Simulation . . . 20
2.3 Comparison of RW and RW+TLM by simulation . . . 22
2.3.1 General case . . . 23
2.3.2 A worse case . . . 24

3 Protocols with Tabu lists in Nodes 29
3.1 Algorithms . . . 29
3.1.1 Tabu List on Links Detecting Cycles . . . 29
3.1.2 Tabu Search on Cycles with Accusation Counter . . . 30
3.1.3 Tabu Search on Cycles with Accusation Counter and Law of Probabilities . . . 31
3.2 Simulation's Results . . . 32
3.2.1 Results of TLLCACLP . . . 33
3.2.2 Results of TLLCAC . . . 33
3.3 Study of conjecture on TLLCAC . . . 33
3.3.1 The algorithm converges to a spanning tree . . . 33
3.3.2 A particular case . . . 35
3.4 Comparison between all protocols . . . 37
3.4.1 Some examples . . . 37
3.4.2 Summary Table . . . 39

II Study of Hybrid Protocols **41**

4 Study of Breath-First Search Tree **43**
4.1 The Algorithm . . . 43
4.2 Theoretical Stabilizing Time of BFST . . . 44
4.3 Experimental confirmation . . . 45

5 Comparison between Hybrid Protocols **47**
5.1 Presentation of our hybrid protocols . . . 47
5.2 Simulations . . . 49
5.2.1 The mickey graphs . . . 49
5.2.2 The "steffi graphs" . . . 50

Conclusion **53**

Bibliography **54**

A The tool "sinalgo" **55**
A.1 Overview . . . 55
A.2 Our use of the simulator . . . 56
A.3 The asynchrony in sinalgo . . . 56
A.4 Assumptions made on the simulated model . . . 57
A.5 The loss of message . . . 57

B Pack of Graphs **59**
B.1 Arbitrary Graphs . . . 59
B.2 Lollipop Graphs . . . 59
B.3 Graphs with an increasing density until sink . . . 60
B.4 Mickey Graphs . . . 61
B.5 4-paths-sink Graphs . . . 61
B.6 3-parts Graphs . . . 62
B.7 City Graphs . . . 62
B.8 Burger Graphs . . . 63
B.9 Flower Graphs . . . 63

Introduction

A Wireless Sensor Network (WSN) consists of hundreds or thousands of low cost nodes which could either have a fixed location or be randomly deployed to monitor the environment. We focus on a crucial problem for WSN: *routing*. That is the problem of transmitting data from a sensor (so called the *source*) to another one (the *sink* or the *destination*), the two sensors can be arbitrarily far in terms of hops (a hop is the direct communication from a node to another) from each other. Typically, a routing protocol encapsulates control data into messages to follow a path from the source to the sink.

Overview: Sensor nodes are extremely basic in terms of interfaces and components. They consist of several sensors, a processing unit with limited computational power and limited memory, a wireless system of communication and a limited capacity power source. The sensors are used to collect physical information such as temperature, sound, vibration, pressure, motion or pollutants. And all these data may have an interest in many areas. For example, we can deploy a network to monitor a forest fire outbreaks or even monitor all movements on a battlefield. The goal of the processing unit is mainly to execute the communication protocols, however this unit is limited in terms of memory and power. Communications are performed by a wireless transmission system, usually radio transceivers. The range of the radio transceivers defined a neighborhood which can be asymmetric (in case of heterogeneous ranges) or symmetric (in case of homogeneous ranges). Here, we always consider that communications can be bidirectional, *i.e.*, if a node n_1 can communicate with a node n_2 then n_2 may also communicate with n_1. Note also that the communication can be unreliable due to the message collisions. However, we assume here that communication primitives are reliable but asynchronous (the alternating bit protocol ([**?**]) can be used to turn unreliable communication into reliable but asynchronous communication). The neighborhoods of nodes define an implicit autonomous network: the WSN. Throughout this report, the network will be always supposed connected.

The routing in WSN is particular mainly because the data flow ends at special nodes called base stations (or one sink in our case). A base station links the sensor network to another network (like a gateway) to disseminate the data sensed for further processing. One of main issue of the routing in WSN is to save energy consumption to extend the lifetime of the network in spite of the small batteries of the nodes. Another issue to increase the lifetime of the network is the load balancing: we want to avoid some nodes to drastically consume more than other ones. To cope with energy saving, protocol designers usually try to reduce the number of hops from the source to the destination, as well as the size of control part in the message to transmit. Moreover, they try to limit the energy consumption by making the nodes alternating between sleep modes and active modes.

A WSN must have several properties according to applications. In the case of a fire starting, the most important is that the information arrives as soon as possible, or if we want to observe natural phenomena precisely, it is better to be sure to receive all messages (see example in Figure 1). Keeping in mind that one of the biggest problems of a WSN is its lifetime, the particularity of this network is its topology which is highly dynamics because there are many events changing WSN continuously. We have the loss

of messages due to collisions, the loss of links due to interferences, for example, the loss is temporary if a truck parks between two nodes or definitive if a wall is built between two nodes. But we can also lose nodes when the battery is too low or empty, or add nodes if we want to extend the network or to correct problems.

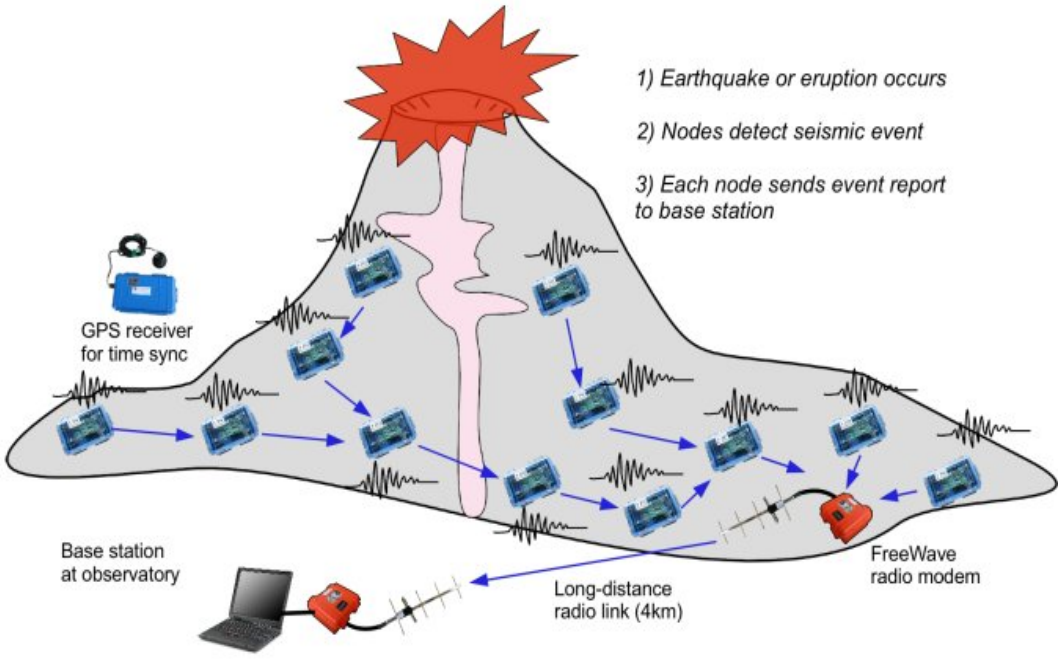


Figure 1: An example of applying a WSN

To sum up routing is a crucial but difficult issue in WSN. Below, we list the main problems to circumvent:

- nodes have limited memory and computing power.
- nodes must limit transfers.
- messages loss.
- links or nodes failures (equivalent to lose all links incident to some node).
- multi-hop network.
- asynchronous communication.

Due to the dynamicity and the limited capability of the nodes, routing paths cannot be computed offline and once for all, that we need to design routing algorithms that are highly distributed. In our distributed algorithms we will always use a all-to-one routing that is, there is only one sink node, and all other nodes may have to route message to the sink. Three routing strategies are typically used in the literature:

- *Flooding* consists of routing the data without any infrastructure. The path is dynamically computed for each message sent. It can be deterministic but the drawback is that many control information have to be stored into the message. In contrast, it can use probabilistic methods but the message delivery is then not guaranteed.

- *Overlay* consists in computing a routing table at each node and then routing the data using it. However, routing tables are difficult to maintain in a dynamic network.

- *Hybrid* mixes the two previous approaches. The idea is to use an overlay when the system is stable. When there are topological changes, the overlay can be temporally perturbated. In this case, techniques derivated from flooding are used to find another path to the destination.

Related works: Now, we present some classical routing techniques.

In [**?**], authors present flooding and gossiping which are two classical mechanisms to relay data in sensor networks without the need for any routing algorithms and topology maintenance. In flooding, each sensor receives a data packet and broadcasts it to all of its neighbors. This process continues until the packet arrives at the destination or the maximum number of hops for the packet is reached. On the other hand, gossiping is a slightly enhanced version of flooding where the receiving node sends the packet to a randomly selected neighbor, which picks another random neighbor to forward the packet to and so on.

In [**?**], authors explain several drawbacks:

- implosion caused by duplicated messages sent to same node.
- overlap when two nodes sensing the same region send similar packets to the same neighbor.
- resource blindness by consuming large amount of energy without consideration for the energy constraints.

Gossiping avoids the problem of implosion by just selecting a random node to send the packet rather than broadcasting. In particular, they use meta-data (high-level data descriptors) negotiations to eliminate the transmission of redundant data throughout the network. However, this cause delays in propagation of data through the nodes.

Among overlay-based protocols, we have hierarchical protocols. The main aim of hierarchical routing is to efficiently maintain the energy consumption of sensor nodes by involving them in multi-hop communication within a particular cluster and by performing data aggregation and fusion in order to decrease the number of transmitted messages to the sink. Cluster formation is typically based on the energy reserve of sensors and sensor's proximity to the cluster head like in [**?**] and [**?**]. In [**?**], LEACH (Low Energy Adaptive Clustering Hierarchy) is one of the first hierarchical routing approaches for sensor networks. The idea proposed in LEACH has been an inspiration for many hierarchical routing protocols [**?**], [**?**], [**?**] and [**?**], although some protocols have been independently developed in [**?**] and [**?**]. The operation of LEACH is separated into two phases, the setup phase and the steady phase. In the setup phase, the clusters are organized and clusterheads are selected. In the steady phase, the actual data transfer to the sink takes place. These two phases alternate with a certain time, which is determinated *a priori*, and when the network goes back into the setup phase, it enters another round of selecting new clusterhead. The advantage of this method is that energy consumption is more evenly distributed between each sensor. Thus the protocol satisfies the load balancing.

A different group of overlay routing protocols is based on a spanning tree idea. The main protocols in this category are the Direct Spanning Tree (DST) routing protocol introduced by [**?**], Spanning Tree proposed by [**?**], Minimum Energy Spanning Tree (MEST) published in [**?**], Shortest Path Tree (SPT) presented in [**?**], and Gathering Load Balanced Tree (GLBT) described in [**?**]. For this type of routing protocols, a tree covering the entire interest area rooted at the sink is built for this network. The tree is constructed from selected nodes that meet specified criteria to be a member. Sender node will choose one available node to forward data to it, and then the data will be directed in the same manner to the sink through the spanning tree.

An other class of overlay-based protocols is called Location-based algorithms and they are studied in [**?**], [**?**] and [**?**]. They rely on the use of nodes position information to find and forward data towards a destination in a specific network region. Position information is usually obtained from GPS

(Global Positioning System) equipment. They usually enable the best route to be selected, reduce energy consumption and optimize the whole network. Na Wang in [**?**] has studied the performance of the geographic based protocols.

In [**?**], the paper propose Hybrid Routing Protocol (HRP), which creates route only when desired by the source node as in case of reactive routing protocols. The propose protocols maintain routing table at each node as in case of proactive routing protocols. Hence they are called hybrid routing protocol. The proposed protocol takes advantage of broadcast nature of mobile ad hoc network which is used to gain maximum routing information at the nodes in the network. HRP uses the Ad hoc On-demand Distance Vector protocol (AODV) [**?**], a highly used reactive routing protocol in Ad hoc network. The results show significant reduction in routing overhead, decreasing end-to-end delay and increasing packet delivery ratio.

Our contribution: We are interested in another hybrid routing protocol more suited to a WSN: it is the protocol proposed by Watteyne in [**?**]. The protocol mixes two strategies:

1. Computing a breath-first search tree rooted at the sink node and then routing messages along the tree.
2. Collecting information into the message to deterministically update the route when some topological changes occur.

The main drawback of the method is its cost. Indeed, heavy control information (list of identifiers) are stored into the message during the routing to guarantee its delivery. We propose to improve this routing protocol by introducing probabilistic methods into the protocol.

Before presenting the strategies chosen to do better than what exists, we describe the methodology followed to study protocols:

1. We implement the protocols studied in the simulator SINALGO (see the description of SINALGO in appendix page 55).
2. We simulate our protocols on a large pack of graphs.
3. We study results to propose conjectures.
4. We propose new graphs to confirm or to invalidate these conjectures.
5. We specify theorems and we prove them.

For the first strategy of our routing hybrid protocols, we keep the breath-first search tree construction using the algorithm in [**?**]. This algorithm constructs a breadth-first Tree in a self-stabilizing manner, *i.e.*, the algorithm satisfies:

- Starting from any state (for example, a state of the system after topological changes), it is guaranteed that the system will eventually reach a correct state (convergence).
- Given that the system is in a correct state, it is guaranteed to stay in a correct state, provided that no fault happens (closure)

Thus this property is very interesting in distributed system. This protocol calculates the shortest path between all nodes and the sink in a WSN.

For the second strategy, we study probabilistic methods. A classic probabilistic method is "Random Walk", *i.e.*, each message is sent to a neighboring node randomly. There are many papers studying this

protocol but we are interested in the improvement proposed by Yamashita in [**?**]. In this article, they propose to change the transition probability matrix from

$$P_{SRW}(u,v) = \begin{cases} \deg(u)^{-1} & \text{if } v \in N(u) \\ 0 & \text{otherwise} \end{cases}$$

for the Standard Random Walk (called SRW) to

$$P_{RWLD}(u,v) = \begin{cases} \dfrac{\deg(v)^{-\frac{1}{2}}}{\sum\limits_{w \in N(u)} \deg(w)^{-\frac{1}{2}}} & \text{if } v \in N(u) \\ 0 & \text{otherwise} \end{cases}$$

where $G = (V, E)$, a graph with V, the set of nodes and E, the set of vertices, $N(u)$ is the set of vertices adjacent to a node u and $deg(u)$ is the number of vertices of u.
This matrix uses the local degree information, and we call Random Walk using Local Degree (RWLD), the routing protocol using this matrix. Authors define the hitting and cover times to test the efficiency of the network protocol. Clearly, the hitting time $H_G(P; u, v)$ is the expected number of transitions necessary for a random walk starting at u to visit v for the first time. Let P be the transition probability matrix, we can also define $H_G(P) = \max_{u,v \in V} H_G(P; u, v)$. The cover time $C_G(P; u)$ is the expected number of transitions necessary for a random walk starting at u to visit all vertices in V and $C_G(P) = \max_{u \in V} C_G(P; u)$. In [**?**], They prove an improvement of the hitting and cover time from SRW to RWLD which pass from $O(n^3)$ to $O(n^2)$ for the hitting time and from $O(n^3)$ to $O(n^2 \log n)$ for the cover time, with n the order of the graph. They also showed in [**?**] that RWLD is a particular case of the random walk using the Metropolis-Hastings algorithm.

We are interested to compare SRW and RWLD by simulation. Experimental results have confirmed the theoretical results for many graph but we have proved (experimentally and theoretically) that there are graphs where SRW is better than RWLD.

After, we improve these methods by prevent cycles using a tabu list. So, we have two possibilities : either store the tabu list in the message, either in the node. First, we introduce a tabu list in the message in protocols SRW and RWLD. We note SRW+TLM, the Standard Random Walk with Tabu List in the Message, and RWLD+TLM, the Random Walk using Local Degree and Tabu List in the Message. The policy in place to fill and empty the list is simple:

- Each message stores node identifiers through where it passed.
- When the list is full, an identifier of the node is removed.

Secondly, the list is stored in the node. More precisely, to choose who to send the message, we have a list associated with each neighboring nodes and we fill the list with the message identifier when we send it through this link to this neighbourg.

To try to prevent cycles, we have implemented a first protocol called Tabu list on Links detecting Cycles (TLLC) that prohibited sending a message on a link by which the message already travelled. But this protocol quickly becomes inefficient when there are many messages in the network. To correct this inconvenient, we add an accusation counter for each links that increases every time a same message returns to the current node. These are the counters associated with the sending link and receiving link. The node transmits the message on a link that has the minimum counter. This new protocol, named Tabu list on Links detecting Cycles with accusation Counters (TLLCAC) is very interesting because it converges to a spanning tree closed to the BFST with a high probability.

We have also studied a Tabu list on Links detecting Cycles with accusation Counters and law of Probability protocol (TLLCACLP) that sends the message on link with a small counter with a higher probability on the link that has a large counter. But it is less efficient than TLLCAC.

Finally, we study the hybrid routing protocol for WSN with our probabilistic methods. Initially, we work on Breath-First Search Tree (BFST). First, we estimate the mean time of complete stabilization of the tree by simulation. Secondly, we show that it is the maximum distance between the sink and a node multiply by the mean time of "a global step". We define the time of a global step by the time taken by each node to perform at least one node step, *i.e.*, all nodes have at least received a message from their neighbors.

After that, we compare BFST with three hybrids protocols using the best different probabilistic methods (*i.e.*, we have the new protocols: BFST+RWLD, BFST+RWLD+TLM and BFST+TLLCAC). We introduce dynamicity in the network to simulate a WSN. We just create a fault with a loss of link and so, we can observe that BFST+RWLD+TLM is the single protocol better than BFST. We explain that because the graph tested is very particular. it is created to slow the restabilization of the tree. But the result is interesting because we can hope that hybrid protocols are better than BFST in a dynamic network. In a future work, we will compare our protocols on networks with several correlated faults, to better model WSN.

Outline: In the first part, we compare several probabilistic methods together. The chapter one is devoted to a comparison between SRW and RWLD, we explain the relationship between the form of graphs and simulation results. In the second chapter, we add a tabu list in the message to the previous two protocols, so we have two new protocols: SRW+TLM and RWLD+TLM. We compare these four routing protocols probabilistic and test the influence of the size of the list in different graphs. In the third and final chapter of this part, we introduce three new protocols: TLLC, TLLCAC and TLLCACLP. They use a tabu list for each link of each node, therefore they are different from protocols "RW" because each node keeps a memory of the passage of previous messages. We will be greatly interested in TLLCAC because this algorithm converges to the computation of a shortest path. And finally we give a complete comparison of all these probabilistic methods.

In the second part, we study the hybrid protocols with in first the study of BFST. We compute the mean time of stabilization for the BFST. We find what it's the best time for a node to use the probabilistic method. Then we compare the following different protocols: BFST, BFST+RWLD, BFST+RWLD+TLM and BFST+TLLCAC on a network where we introduce a fault.

Part I

Comparison of Different Methods of Probabilistic Routing

Chapter 1

Standard Random Walk *vs.* Random Walk using Local Degrees

In this chapter, we compare two probabilistic routing protocols: the Standard Random Walk (SRW) and an improvement proposed by Yamashita [**?**] called Random Walk using Local Degrees (RWLD).

Yamashita proves that the local degree information is powerful enough to enhance the hitting time of a random walk. By local degree information, he means the degrees of a node and its neighbors. Indeed, the hitting time of the RWLD protocol of Yamashita is in $O(n^2)$ while the hitting time of SRW is in $O(n^3)$.

We want to confirm that its results hold in particular for WSNs. So we implemented these two protocols and we compared the simulation's results. We apply these simulations on several classes of graphs. Simulations show that RWLD is better than SRW in most of the cases. For example, we test them on lollipop graphs (see Figure 1.1), which is shown to be a worst case for SRW [**?**]. We note that our results on the lollipop graphs confirms the results of Brightwell and Winkler in [**?**] which states that RWLD is widely better than SRW on lollipop graphs.

However on some simulations, we observed that RWLD does not enhance SRW. This is due to the fact that RWLD has a transition probability matrix which avoids dense areas. To see this, we first exhibit by simulations a class of graphs where RWLD and SRW are equivalent: the regular graphs, that is, graphs where all nodes have the same degree. This result is straightforward to prove because in such graphs the transition matrices of RWLD and SRW are equal.

Then, we found graphs where RWLD is worst than SRW. Intuitively, we need graphs where the routing messages must pass through dense areas before reaching the sink node. After several inconclusive tests, we find a property of graphs ensuring that SRW is better than RWLD. The property is the following: from the source to the sink, nodes have increasing degrees (see Figure 1.5). We formally proved this in Theorem 2.

First of all, we described the algorithms.

1.1 Algorithms

SRW and RWLD are probabilistic, *i.e.*, the hop of a message from one node to another is only decided according to a transition probability matrix.

1.1.1 Standard Random Walk

The principle of SRW is that each node transmits incoming messages by randomly choosing the destination among its neighbors. The probability law is uniform, *i.e.*, each neighbor has the same probability

to be chosen. Thus, the probability that node u relays a message to a node v is:

$$P_{SRW}(u,v) = \begin{cases} \deg(u)^{-1} & \text{if } v \in N(u) \\ 0 & \text{otherwise} \end{cases}$$

where $N(u)$ is the set of neighbors of u and $deg(u)$ is the number of neighbors of u.

1.1.2 Random Walk Using Local Degree Information

The implementation of RWLD is similar to SRW, except that probability weights are associated to neighbors. With the previous notation, RWLD use the following transition probability matrix:

$$P_{RWLD}(u,v) = \begin{cases} \dfrac{\deg(v)^{-\frac{1}{2}}}{\sum\limits_{w \in N(u)} \deg(w)^{-\frac{1}{2}}} & \text{if } v \in N(u) \\ 0 & \text{otherwise} \end{cases}$$

In contrast with SRW, RWLD needs information on the degree of neighboring nodes. According to the transition probability matrix, RWLD transmits messages with a greater probability to a neighboring node of smallest degree. Hence, we can say that RWLD tends to avoid the dense areas of the network. Note that we use this latter property to construct graphs where RWLD is worst than SRW.

1.2 Simulations

We now present results obtained by simulating SRW and RWLD on different graphs. In the simulations, we compute the average number of hops to route the messages from a source node to a sink node and this average is evaluated over the routing of 10,000 messages. We decide to observe the average number of hops because it is a usual measure in the literature to compare routing protocols. Note that the routing time is not significant here because the networks are asynchronous.

1.2.1 Improvements

First, we simulate the protocols on the lollipop graph (see Figure 1.1) which is a worst case for SRW [**?**]. Brightwell and Winkler in [**?**] showed that the hitting time (and also the cover time) of the lollipop

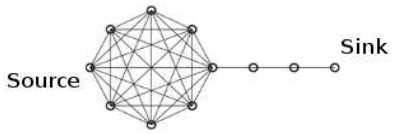


Figure 1.1: A lollipop graph of size 11

is $(1 - o(1))\frac{4}{27}n^3$ where n is the number of nodes. So, in Figure 1.2, we compare the curve $\frac{4}{27}n^3$ to the simulation curve. In the simulation, we placed the sink node at the end of the path and the source node in the complete graph. Knowing that the hitting time of RWLD is in $O(n^2)$, we adjust the coefficient to $\frac{2}{3}$ to have similar curves.

We can observe that our simulations confirm the theoretical bounds and the improvement. On this graph the improvement is due to the fact that RWLD sends the message with a greater probability to nodes that have a low degree. Thus at the average, messages go out of the complete graph more rapidly

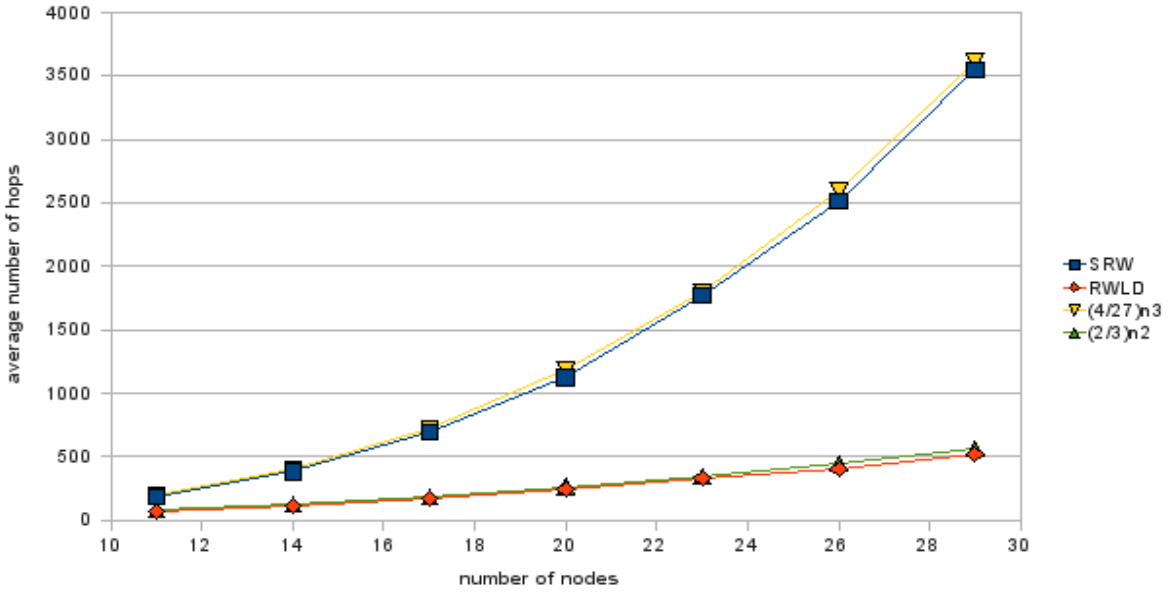


Figure 1.2: Simulation results for lollipop graphs

using RWLD than SRW.

Figure 1.3 gives simulation's results of SRW and RWLD on arbitrary graphs. These results confirm that RWLD is better than SRW in the general case. In all these graphs, the average degree of the nodes is 8.

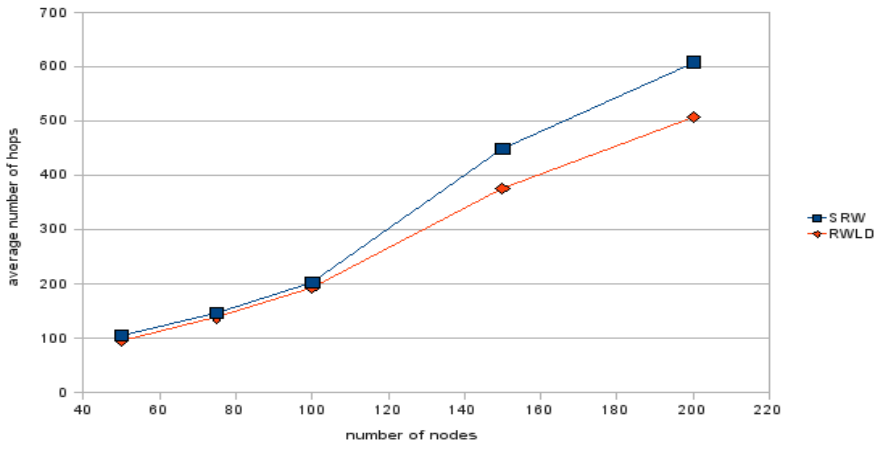


Figure 1.3: Simulation results for arbitrary graphs

We can remark that when the number of nodes is small, the gap between SRW and RWLD is less significant (because the degree of each node is still fairly homogeneous) while it increases when the number of nodes grows (because denser areas appear). To confirm this result, we compute the average number of hops for 1000 messages sent in 50 randomly generated graphs which have the same number of nodes and about the same average degree. More precisely, we consider 3 different type of graphs: (1)

graphs with 400 nodes and an average degree of 15, (2) graphs with 600 nodes and an average degree of 25, and (3) graphs with 1000 nodes and an average degree of 50. Results are given in Figure 1.4. We can observe that the two obtained curves are linear and parallel.

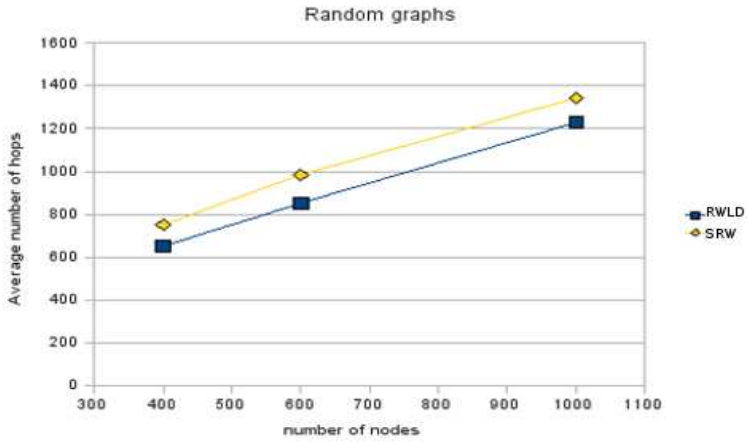


Figure 1.4: Simulation results for random graphs

1.2.2 Counter-examples

Simulation results are strongly dependent on the shape of the graph and the placement of source node and sink node. We now look for counter-examples, that is, classes of graphs where RWLD is not an improvement of SRW. To that end, we use the property of the transition probability matrix of RWLD. We first give counter-example where SRW and RWLD are equivalent. To obtain this result, we simulated SRW and RWLD in regular graphs. In the second counter-example, we force the routing algorithm to follow a path where the encountered degrees increase.

First counter-example

We validate the first counter-example by simulating the two protocols on rings. We formally prove this results in the general case in Theorem 1 (page 16).

Second counter-example

We consider graphs where the degree varies in inverse proportion to the distance of the node to the sink. Figure 1.5 gives an example of such a graph. In the figure, the sink node is in the circle labeled 1 and the source node has label 10.

Figure 1.6 gives the experimental result which confirms that SRW is better than RWLD in graphs with increasing degrees. The property of these graphs is that the message is forced to pass through a path consisting of nodes with an increasing degree to arrive at the sink. Note that it is the only class of graphs we found where RWLD is not an improvement of SRW. Observations confirm that RWLD avoids dense areas while SRW ignores them. We also test graphs with a big dense area between the source and the sink as “Burger graphs” (see Figure 1.7) but when the sink has a degree lower than its neighbors, RWLD is even better.

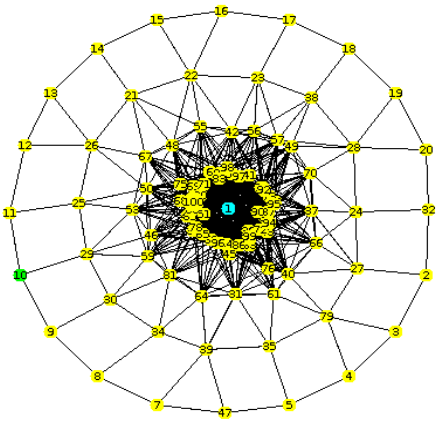


Figure 1.5: Graph with increasing degrees

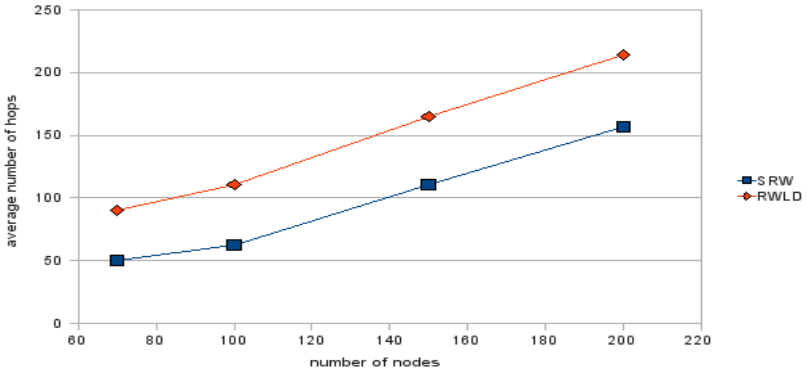


Figure 1.6: Simulation results for graphs with increasing degrees

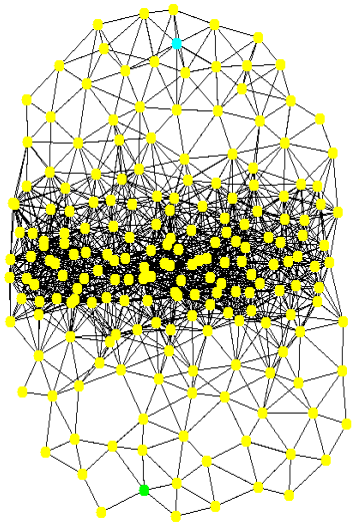


Figure 1.7: Burger graph

1.3 Formal results

In the previous section, we predicted some conjectures. We now prove some of them.

1.3.1 First counter-example

We now prove SRW is equivalent to RWLD in regular graphs. To see this, we show that they have the same transition probability matrix when the network is regular.

Let $G = (V, E)$ be a graph, we express the property of the graph studied: $\forall\, u, v \in V$, $\deg(u) = \deg(v)$.

Theorem 1. *If all nodes of the graph have the same degree then SRW and RWLD are equivalent.*

Proof: To prove this property, we show that our Random Walks protocols have the same transition probability matrix. This matrix for RWLD is (using the same notations of previous section):

$$P_{RWLD}(u,v) = \begin{cases} \dfrac{\deg(v)^{-\frac{1}{2}}}{\sum\limits_{w \in N(u)} \deg(w)^{-\frac{1}{2}}} & \text{if } v \in N(u) \\ 0 & \text{otherwise} \end{cases}$$

Now, if all nodes of the graph have the same degree then $\deg(u) = \deg(v) = \deg(w)$. Thus

$$P_{RWLD}(u,v) = \begin{cases} \dfrac{\deg(u)^{-\frac{1}{2}}}{\sum\limits_{w \in N(u)} \deg(u)^{-\frac{1}{2}}} & \text{if } v \in N(u) \\ 0 & \text{otherwise} \end{cases}$$

$$\Leftrightarrow P_{RWLD}(u,v) = \begin{cases} \dfrac{\deg(u)^{-\frac{1}{2}}}{\deg(u)\deg(u)^{-\frac{1}{2}}} & \text{if } v \in N(u) \\ 0 & \text{otherwise} \end{cases}$$

because we have $|N(u)| = \deg(u)$ by definition thus $\sum\limits_{w \in N(u)} \deg(u)^{-\frac{1}{2}} = \deg(u)\deg(u)^{-\frac{1}{2}}$

$$\Leftrightarrow P_{RWLD}(u,v) = \begin{cases} \frac{1}{\deg(u)} & \text{if } v \in N(u) \\ 0 & \text{otherwise} \end{cases}$$

$$\Leftrightarrow P_{RWLD}(u,v) = P_{SRW}(u,v)$$

we can conclude that SRW $\Leftrightarrow$ RWLD with the previous hypothesis. □

1.3.2 Second counter-example

We exhibited some structural properties of the graph that prevent the improvement given by the use of local degree informations. More precisely: If all possible paths to reach the sink are composed of nodes in such way that degrees increase, then SRW is better than RWLD.

Notations: We now define some notations used. Let $G = (V, E)$ be a graph with $u, v \in V$ nodes and let s be the sink node. We note P, the transition probability matrix where P_{SRW}, the matrix of SRW and P_{RWLD} the matrix of RWLD. We recall that the hitting time $H_G(P; u, v)$ is the expected number of transitions necessary for a random walk starting at u to visit v for the first time. In our protocols, we call that the average number of hops. With a random walk $\omega = (\omega_0, \omega_1, ...)$, it is formally defined by

$H_G(P; u, v) = \mathrm{E}[\inf\{i \geq 1 \mid \omega_i = v\}]$, . Let p be a path from u to v then the distance between two nodes is $d(u, v) = \min(length(p))$ for all paths p possible. We remind that $\deg(u)$ is the number of vertices of u and $N(u)$ is the set of neighbors ($|N(u)| = \deg(u)$). Let $E_l = \{u \in V \mid d(u, s) = l\}$ be the set of nodes which are at distance l from the sink.

Theorem 2. *If G_1 a graph such that $\forall\ u \in E_l$ and $\forall\ v \in E_{l+1}$, we have $\deg(u) > \deg(v)$ then for all x a source node, we have $H_{G_1}(P_{SRW}; x, s) < H_{G_1}(P_{RWLD}; x, s)$*

Proof: $\forall\ u \in E_l$, we can decompose the set $N(u)$ into three disjoint sets as follows:

- $N_1(u) = \{v \in N(u) \mid v \in E_{l-1}\}$ with $|N_1(u)| = n_1$,
  $d_1^{max} = \max\limits_{v \in N_1(u)} \{\deg(v)\}$ and $d_1^{min} = \min\limits_{v \in N_1(u)} \{\deg(v)\}$.
- $N_2(u) = \{v \in N(u) \mid v \in E_l\}$ with $|N_2(u)| = n_2$,
  $d_2^{max} = \max\limits_{v \in N_2(u)} \{\deg(v)\}$ and $d_2^{min} = \min\limits_{v \in N_2(u)} \{\deg(v)\}$.
- $N_3(u) = \{v \in N(u) \mid v \in E_{l+1}\}$ with $|N_3(u)| = n_3$,
  $d_3^{max} = \max\limits_{v \in N_3(u)} \{\deg(v)\}$ and $d_3^{min} = \min\limits_{v \in N_3(u)} \{\deg(v)\}$.

We note that $\deg(u) = n_1 + n_2 + n_3$ and for $i = 1,\ 2$ or 3, first, we compute the transition probability $P_{SRW}(u, v_i)$ and secondly, we bound $P_{RWLD}(u, v_i)$ with $\forall\ v_i \in N_i(u)$.

$$P_{SRW}(u, v_i) = \sum_{v \in N_i(u)} P_{SRW}(u, v)$$

$$\Leftrightarrow P_{SRW}(u, v_i) = \sum_{v \in N_i(u)} \frac{1}{\deg(u)}$$

$$\Leftrightarrow P_{SRW}(u, v_i) = \frac{n_i}{n_1 + n_2 + n_3}$$

And now, we bound $P_{RWLD}(u, v_1)$ knowing it's the same way for $P_{RWLD}(u, v_2)$ and $P_{RWLD}(u, v_3)$:

$$P_{RWLD}(u, v_1) = \sum_{v \in N_1(u)} P_{SRW}(u, v)$$

$$\Leftrightarrow P_{RWLD}(u, v_1) = \sum_{v \in N_1(u)} \frac{\deg(v)^{-\frac{1}{2}}}{\sum\limits_{w \in N(u)} \deg(w)^{-\frac{1}{2}}}$$

$$\Rightarrow \begin{cases} \sum\limits_{v \in N_1(u)} \left(\frac{1}{\sqrt{d_1^{max}}} \times \frac{1}{\sum\limits_{v \in N_1(u)} \frac{1}{\sqrt{d_1^{max}}} + \sum\limits_{v \in N_2(u)} \frac{1}{\sqrt{d_2^{min}}} + \sum\limits_{v \in N_3(u)} \frac{1}{\sqrt{d_3^{min}}}}\right) \leq P_{RWLD}(u, v_1) \\ P_{RWLD}(u, v_1) \leq \sum\limits_{v \in N_1(u)} \left(\frac{1}{\sqrt{d_1^{min}}} \times \frac{1}{\sum\limits_{v \in N_1(u)} \frac{1}{\sqrt{d_1^{min}}} + \sum\limits_{v \in N_2(u)} \frac{1}{\sqrt{d_2^{max}}} + \sum\limits_{v \in N_3(u)} \frac{1}{\sqrt{d_3^{max}}}}\right) \end{cases}$$

$$\Rightarrow \begin{cases} n_1 \times \frac{1}{\sqrt{d_1^{max}}} \times \frac{1}{\frac{n_1}{\sqrt{d_1^{max}}} + \frac{n_2}{\sqrt{d_2^{min}}} + \frac{n_3}{\sqrt{d_3^{min}}}} \leq P_{RWLD}(u, v_1) \\ P_{RWLD}(u, v_1) \leq n_1 \times \frac{1}{\sqrt{d_1^{min}}} \times \frac{1}{\frac{n_1}{\sqrt{d_1^{min}}} + \frac{n_2}{\sqrt{d_2^{max}}} + \frac{n_3}{\sqrt{d_3^{max}}}} \end{cases}$$

$$\Rightarrow \begin{cases} \dfrac{n_1}{\sqrt{d_1^{max}}} \times \dfrac{\sqrt{d_1^{max} d_2^{min} d_3^{min}}}{n_1\sqrt{d_2^{min} d_3^{min}} + n_2\sqrt{d_1^{max} d_3^{min}} + n_3\sqrt{d_1^{max} d_2^{min}}} \leq P_{RWLD}(u, v_1) \\ \\ P_{RWLD}(u, v_1) \leq \dfrac{n_1}{\sqrt{d_1^{min}}} \times \dfrac{\sqrt{d_1^{min} d_2^{max} d_3^{max}}}{n_1\sqrt{d_2^{max} d_3^{max}} + n_2\sqrt{d_1^{min} d_3^{max}} + n_3\sqrt{d_1^{min} d_2^{max}}} \end{cases}$$

We want to prove that $P_{SRW}(u, v_1) > P_{RWLD}(u, v_1)$ and so we compare $P_{SRW}(u, v_1)$ with the upper bound of $P_{RWLD}(u, v_1)$.

$$P_{SRW}(u, v_1) - \frac{n_1\sqrt{d_2^{max} d_3^{max}}}{n_1\sqrt{d_2^{max} d_3^{max}} + n_2\sqrt{d_1^{min} d_3^{max}} + n_3\sqrt{d_1^{min} d_2^{max}}}$$

$$\Leftrightarrow \frac{n_1}{n_1 + n_2 + n_3} - \frac{n_1\sqrt{d_2^{max} d_3^{max}}}{n_1\sqrt{d_2^{max} d_3^{max}} + n_2\sqrt{d_1^{min} d_3^{max}} + n_3\sqrt{d_1^{min} d_2^{max}}}$$

$$\Leftrightarrow \frac{n_1 n_2\sqrt{d_3^{max}}(\sqrt{d_1^{min}} - \sqrt{d_2^{max}}) + n_1 n_3\sqrt{d_2^{max}}(\sqrt{d_1^{min}} - \sqrt{d_3^{max}})}{(n_1 + n_2 + n_3)(n_1\sqrt{d_2^{max} d_3^{max}} + n_2\sqrt{d_1^{min} d_3^{max}} + n_3\sqrt{d_1^{min} d_2^{max}})}$$

The denominator is a product and a sum of positive values so it's positive, and according to the property of the graph G_1, we have $d_1^{min} > d_2^{max} > d_3^{max}$. Thus, we can conclude that $P_{SRW}(u, v_1) > P_{RWLD}(u, v_1)$ and by the same way $P_{SRW}(u, v_3) < P_{RWLD}(u, v_3)$.

To conclude, the probability to approach the sink with SRW is greater than the probability to approach the sink with RWLD, thus $H_{G_1}(P_{SRW}; x, s) < H_{G_1}(P_{RWLD}; x, s)$. We shall formally conclude this proof in a future work.

□

Chapter 2

Random Walk *vs.* Random Walk with Tabu Lists in Messages

In this chapter, we improve the two previous "Random Walk" protocols SRW and RWLD by adding information in the routing messages. This new method avoids the messages to turn back in already visited nodes. Indeed, in the tabu list of each message the identifiers of the nodes traversed during the routing are saved. It is important to note that in such protocols, it may happen that the message arrives at a node having all its neighborhood in the tabu list, in this case the message is routed randomly.

The tabu list is stored in the messages. Thus, minimizing its size is crucial. Indeed, if the tabu list is large, then, the energy cost of the sending messages is also large. In the other hand, if the list is too small, it contains not enough information to significantly enhance the random walk. Hence, there is a trade-off to find. The optimal size for the list is the smallest size of list that enhance the random walk. The goal of this chapter is to determine such a size.

Below, we precisely describe the new algorithms proposed. Next, we use simulations to discover the best size of the tabu list. Then, we show that in the general case, random walk with tabu lists in messages is better than RW, *i.e.*, the average number of hops is smaller. Finally, we exhibit a particular class of graphs where we have the opposite result.

2.1 Algorithms

In the previous chapter, we studied two probabilistic ways to route a message. Thus we have two new protocols: Standard Random Walk with Tabu Lists in Messages (SRW+TLM) and Random Walk using Local Degrees with Tabu Lists in Messages (RWLD+TLM). In these protocols, it is the message which saves the list and so, the list must have a limited size.

In our algorithms, there is a tabu list in each routing message with a size fixed in advance. The purpose of this list is to prevent the message to turn back to already visited nodes. This list is filled with the node identifiers through which the message is passed. Then, when the list is full, we remove the first element and add the identifier of the current node in the list. Hence, we follow the FIFO policy, that is the list is actually a queue. Clearly, the message stores the last S visited nodes (where S is the size of the queue defined at the beginning).

Basically, our protocols behave according to the following two cases when a message m arrives at some node v:

- Some neighbors q of v are not in the tabu list of m. In this case, v is added to the tabu list of m and then m is randomly (with either SRW or RWLD) sent to some q.

- Otherwise, all the neighbors of v are in the tabu list of m. In this case, v is added to the tabu list of m. A neighbor n of v randomly (with either SRW or RWLD) chosen is removed to the tabu list and m is sent to n.

2.2 Minimizing the Tabu List Size by Simulation

We simulate our protocols on several classes of graph. It happens that tabu-based protocols are extremely efficient when the message have to follow paths constituted of nodes of degree 2 (indeed, in this case the protocol becomes deterministic). We remark that a tabu list size of 1 is sufficient in this case. So we look for a class of graphs to reduce this efficiency. For that, we increase the node degrees of the paths that the message have to visit.

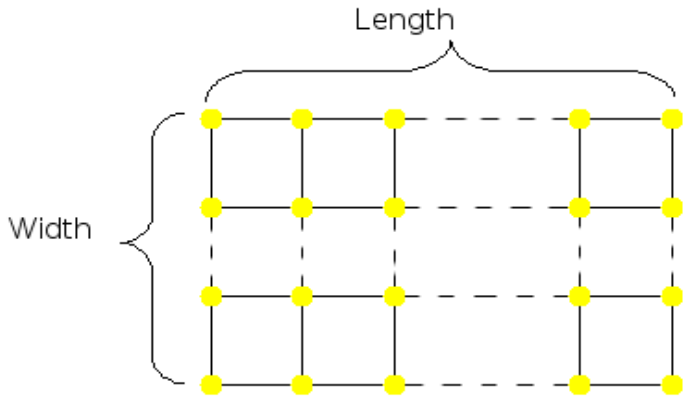


Figure 2.1: A general mesh band

Figure 2.2: A mesh band of length 4 and width 1

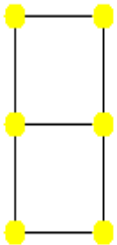


Figure 2.3: A mesh band of length 2 and width 3

We define a mesh band (see 2.1) characterized by a length and a width. We introduce this structure because it generalizes the paths constituted of nodes of degree 2 (see Figure2.2). We suppose that if we

increase the length or the width of a mesh band, then the efficiency of SRW+TLM and RWLD+TLM decrease. The tabu list saves a part of the mesh band. Consequently, by varying the length and the width of the mesh band (see Figure 2.3), we can observe the influence of the tabu list size. We need to simulate the routing protocols on graphs where messages must traverse a mesh band to obtain interesting results. Thus we are interested in a new class of graphs: "dumbbell graphs" (see Figure 2.4). They are composed of two arbitrary graphs with an average degree of 10 and connected by a mesh band.

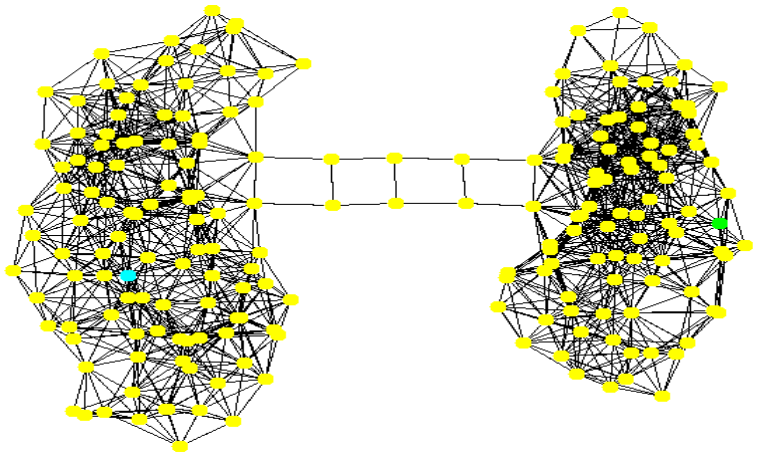


Figure 2.4: A dumbbell graph with a mesh band of length 3 and width 2

We now have good candidates to evaluate the performance of our protocols with different tabu list size. We only present results of the SRW+TLM protocol because the shape of its simulation curves are similar to the shape of simulation curves of RWLD+TLM. We simulate and compute the average over the routing of 10,000 messages. First, we set the width to 2 and vary the length. Figure 2.5 presents results.

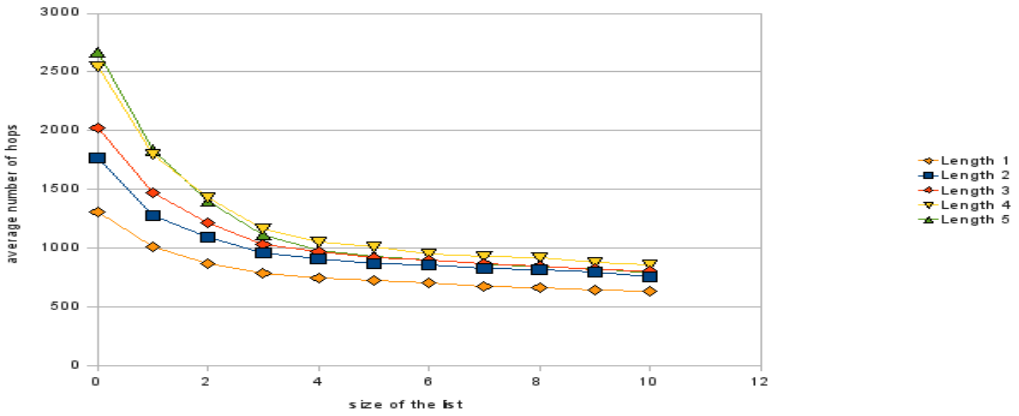


Figure 2.5: Average number of hops for dumbbell graphs (with the width set at 2)

We observe that when the size of the list increases, the number of hops decreases quickly until a size 3 or 4 and then the number of hops decreases slowly. We now vary the width and set the length to 4. We obtain curves of Figure 2.6. We see that when the width becomes too large, the impact of size of the list

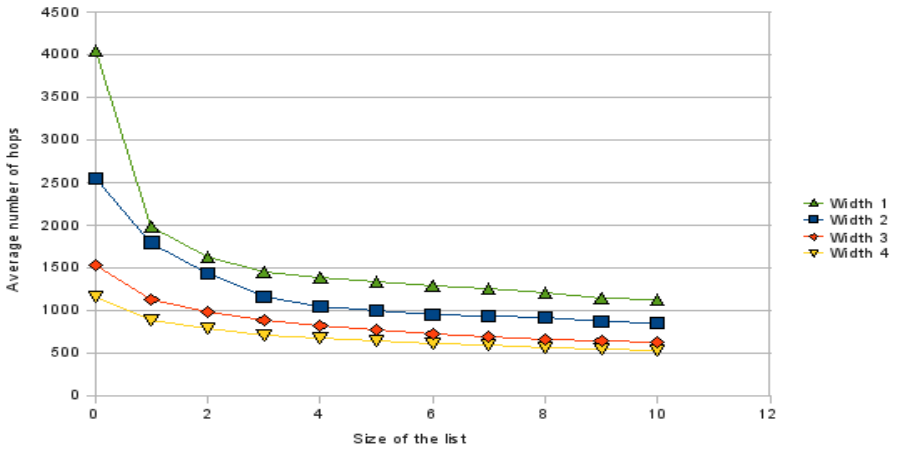


Figure 2.6: Average number of hops for dumbbell graphs (with the length set at 4)

becomes small.

The graph of width 1 has a huge hop between a list size 0 and 1 because:
- When the list size is 0 then, the protocol is strictly RW.
- When the list size is 1 then, the protocol becomes deterministic.

Thus to have a better idea of optimal size, we draw the curve for each graph except for the graph of width 1. We construct these curves as follows: In ordinate, we compute the difference between the average number of hops with the size of the list $S+1$ and the average number of hops with the size of the list S; and in abscissa S. Results are given in Figure 2.7.

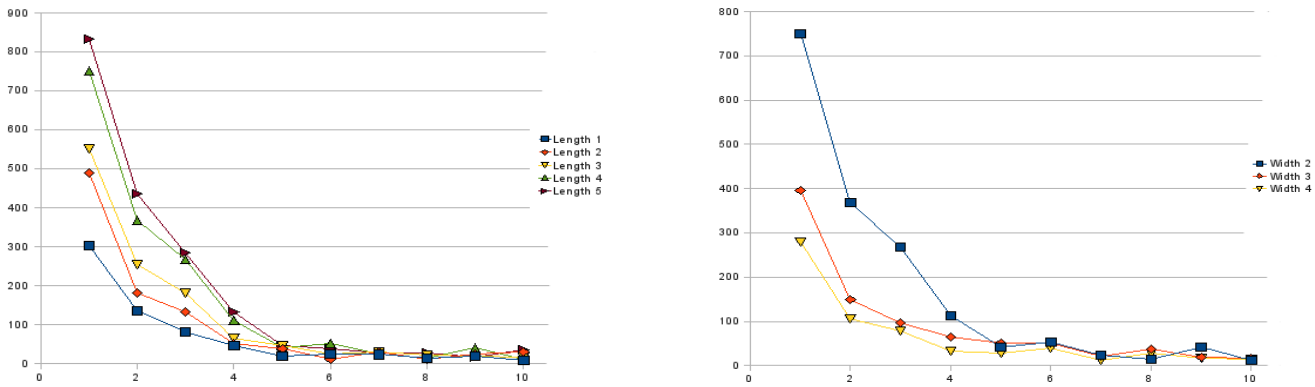


Figure 2.7: Simulation results for dumbbell graphs

We can conclude that the optimal size of the list is 4 because with a lqrger size, the gain is too small on this particular shape of graph which are designed to give bad results for the protocols.

2.3 Comparison of RW and RW+TLM by simulation

Due to the previous observation, we want to show that RW+TLM with a tabu list size of 4 is better than RW.

2.3.1 General case

We simulate SRW+TLM and RWLD+TLM on arbitrary graphs. These graphs are randomly built with an arbitrary topology and an average degree of 8 (see Figure B.1 page 59). Figure 2.8 shows that SRW+TLM is better than SRW (resp. RWLD+TLM is better than RWLD). Curves have the same shape except that the average number of hops is always smaller when we add a tabu list in the protocol.

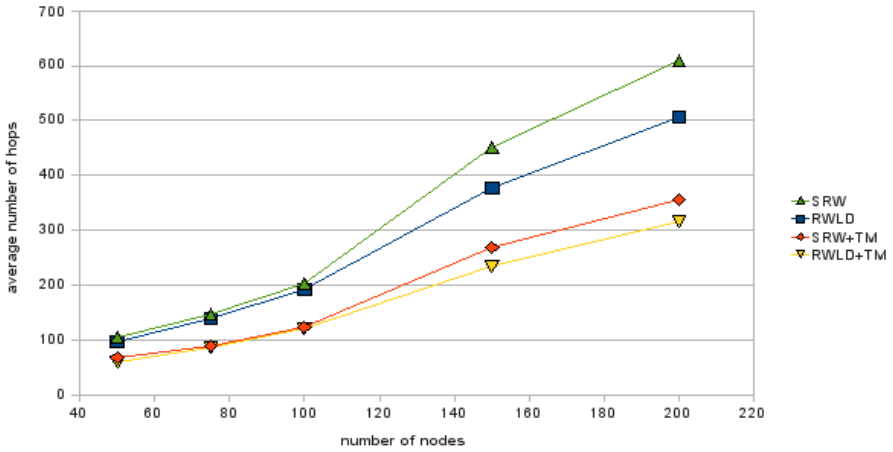


Figure 2.8: Simulation results for arbitrary graphs

We call "mickey graph" the class of graphs where graphs are composed of 2 arbitrary parts connected by several paths constituted of nodes of degree 2 (see Figure 2.9). We present results of "mickey graph" in Figure 2.10. The improvement is even more significant because the list forces the message to cross the paths without turning back.

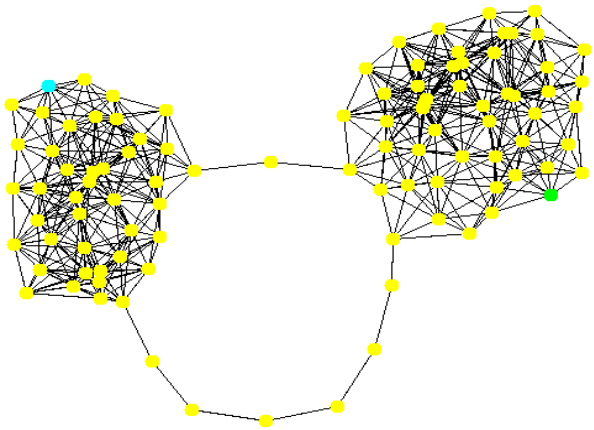


Figure 2.9: Mickey graph

It is interesting to remark that for both previous case, SRW+TLM and RWLD+TLM have closed efficiency. As if the tabu list attenuated the improvement of RWLD compared to SRW.

To conclude, simulation results show that for the vast majority of graphs, RW+TLM is better than RW, especially when there are paths or strips in the graph. We found examples where the result is reversed.

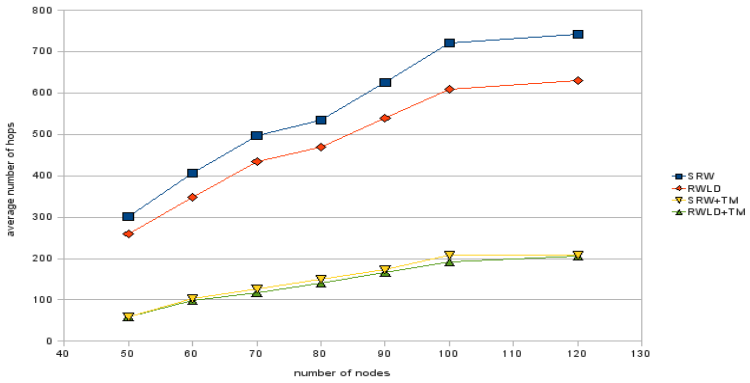


Figure 2.10: Simulation results for “mickey graphs”

2.3.2 A worse case

We want to build a graph where RW is better than RW+TLM. We propose the “flower graphs” (see Figure 2.11 and 2.12) where the source node is in the green circle labeled 10 and the sink node is in the blue circle labeled 1. We call a “petal”, a ring where all its nodes are connected to the source node u which is the only node connected to the sink.

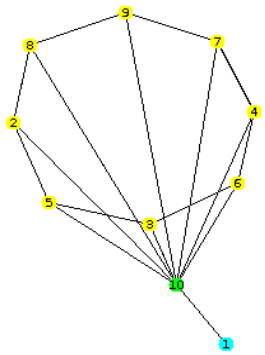


Figure 2.11: Flower graph with one petal of size 8

This graph is designed to slow down RW+TLM. RW and RW+TLM sends the first message to one of neighbors of u with the same probability and RW+TLM adds u in its tabu list. When the message routed by RW is in any petal, it returns to u in an average of three hops. Whereas when the message routed by RW+TLM is in a petal, it has two ways to return to u: if the number of nodes in the petal is greater than the list size, then the minimum number of hops is the list size, else the minimum number of hops is the number of nodes in the petal. We now assume that RW+TLM has a tabu list size strictly greater than the size of the petal. So when the message is returned to u, it can not be sent to a node of the petal because their identifiers are in the tabu list. But it can randomly goes to any other petal. Consequently, we add enough petals to the flower so that RW is better than RW+TLM.

We now compare SRW and SRW+TLM on the class of flower graphs with one petal. We draw the curve of SRW+TLM with graphs which have different size of petal and we present simulation results in

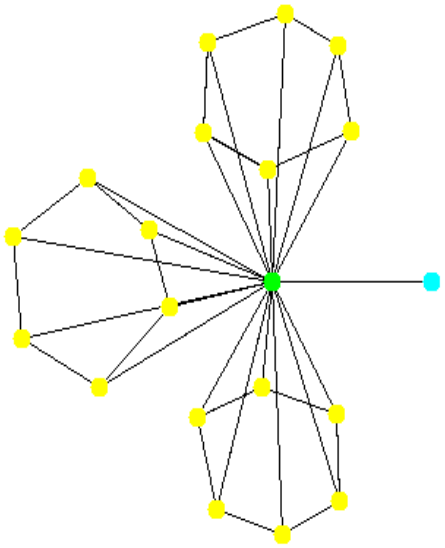


Figure 2.12: Flower graph with 3 petals of size 6

Figure 2.13. Results of SRW can be read for a size of list of zero.

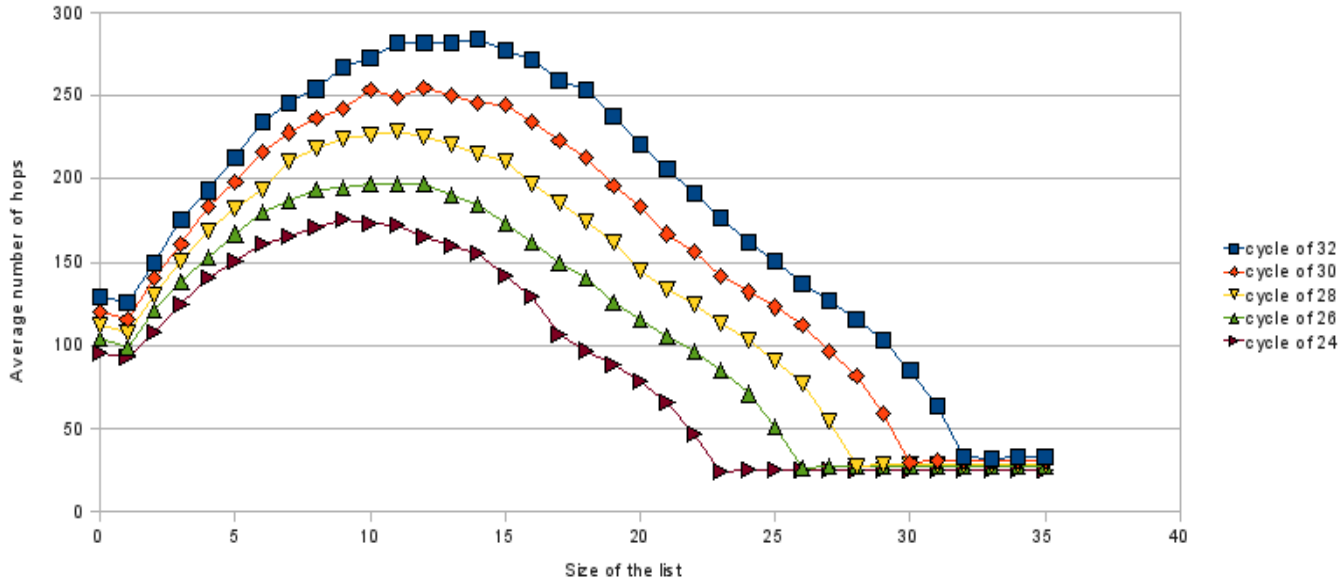


Figure 2.13: Simulation results for flower graphs with one petal

First, with a size of list of 1, results of SRW+TLM are better than SRW because we avoid the message to return immediately to the node that comes to pass. After, SRW+TLM becomes less effective than SRW until a worst result which is approximately when the size of the list is equal to half the petal. To explain this result, we explain why results are better when the size is less than half and when the size is more than half. So the average number of hops increases when size increases up to half the number

of nodes of the petal because the source node is removed earlier from the tabu list and so the message is more likely to leave petal quickly. And the average number of hops decreases when size increases from half the number of nodes of the petal because when the message arrives at the source node, it is less and less likely to return to the petal. Finally, it is necessary that the message records about 75% of the petal for SRW+TLM becomes better than SRW. For a size of list greater than or equal to the size of the petal, the average number of hops stabilizes.

We want to reduce the inefficiency of SRW+TLM in this kind of bad cases; for that we choose to change the output of the list property: from a FIFO list to a random exit of the list. Figure 2.14 gives simulation results. We simulate SRW and SRW+TLM with 2, 3, or 4 petals and each petal has a size of 8 nodes.

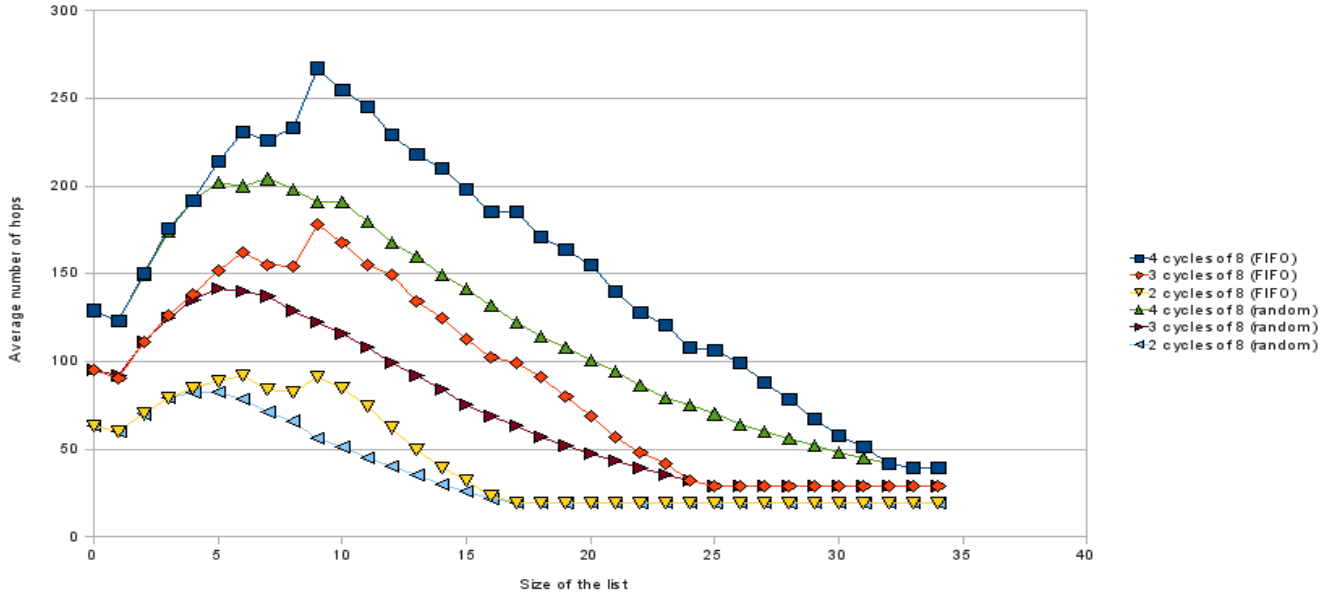


Figure 2.14: Simulation results for flower graphs with 2, 3 and 4 petals

We have always a little improvement with a size of list of 1. Then for the FIFO list, we observe a little hollow during the rise of the number of hops. There is this hollow because with a size of list of 7 or 8, the message recorded the complete petal and so, it is forced to go to the source node. The worst case is with a size of list of 9 because one petal and the source are saved, the message should go another round and in all case, there is always the source node in the tabu list. After, the curve decreases until the stabilization, we remark that there are levels when the size of the list is equal to a multiple of the size of the cycle.

It is interresting to see that in this case of graph, a “random list” is better than a “FIFO list”. But it is a logical result because the problem of this graph is the source node which is an mandatory passage and a “random list” permits to remove it earlier from the tabu list that a “FIFO list”. We performed simulations with a “random list” on all graphs that we have but we found no other cases for which a random policy improves the protocols.

We can conclude this chapter by saying that RW+TLM often improves RW but this amelioration is expensive due to the fact that the message saves visited nodes. We also show by simulation that the optimal size is 4 or inferior. We can note that in a WSN, a node already knows who sends the message. So an improvement without additional cost is to prohibit an immediate return to the same node (this

corresponds to a list size of 1).

Chapter 3

Protocols with Tabu lists in Nodes

We saw that the size of the messages must be limited in order to save energy. So, instead of storing a tabu list in each message, we can try to store the tabu list in the node. However, we must remember that node memory is of limited capacity. So, we must still optimize the size of the tabu list in order to take this constraint into account.

Storing tabu lists in the nodes instead of the messages leads to slightly different properties. In contrast with tabu lists in messages, tabu lists into nodes allow messages to share information. However, as we will see later, this shared information becomes significant after a finite period of convergence.

The general scheme of our algorithms is the following: each node associates a tabu list with each of its incident links. These lists are initially empty. Then, each time, a message m is sent for the first time through a link l, the node stores the message identifier into the corresponding list. Hence, if the message m returns to the same node, the node detects that the message follows a cycle, this cycle contains the edge l. In consequence, the node can prevent the message m to follow the same cycle by routing m through another link. As previously, when a node has several possible choices to route the message, it makes its choice using one of the previous random policies (SRW and RWLD). We obtain a first new protocol: Tabu List on Links detecting Cycles (TLLC). We remark that all links of a node can quickly be banned. Consequently, we introduce an accusation counter for each links (associated with the list) that increases when a cycle is detected. Thus, the link is "good" (avoids cycles) when the accusation counter is small. Hence, we present two new routing protocols: one that sends the message to the link with the minimum counter and another that sends the message according to a law of probability favoring the links with small counters. We call the first Tabu List on Links detecting Cycles with Accusation Counters (TLLCAC) and the second Tabu List on Links detecting Cycles with Accusation Counters and Law of Probability (TLLCACLP). Exact algorithms of our three protocols with tabu lists in nodes are given in the first section.

In the next sections, we studied simulation results of these protocols and we compared all our routing probabilistic protocols. We noted several interesting results such as TLLCAC converges to a spanning tree, or such as RW+TLM has the best results for the first routed message in general cases.

3.1 Algorithms

3.1.1 Tabu List on Links Detecting Cycles

First, we explain the data structure: for each neighboring connection of each node (except the sink), we create a list associated with this connection. This list contains a finite number of message identifier and we shall see later how the list is filled. Typically, a message identifier is a pair of integers composed of a node identifier and a sequence number. The node identifier corresponds to the identifier of the source

node, *i.e.*, the identifier of the node which sends the message. The sequence number corresponds to -th message (the number of message) sent by the source. With m, the number of edge in our network and m_s, the number of edge of the sink. The protocol creates $2m - m_s$ tabu lists.

Then we present policy of filling the list, called FIFO: we put the message identifier in the list when the node sends the message through this connection. If the list is full, we remove the first introduced element and add the last in the queue.

We now present the operation, each node transmits the information message such that:

- if the message identifier is in none of the list of a node, then the node randomly sends the message to a neighbor according to a transition probability (SRW or RWLD).
- if the message identifier is in all lists of a node, then the node randomly sends the message to a neighbor according to a transition probability and the message identifier is removed from all lists of the node.
- if the message identifier is in some lists of a node, then the links associated with these lists are forbidden and the node randomly sends the message to a neighbor according to transition probability.

3.1.2 Tabu Search on Cycles with Accusation Counter

This protocol uses the same list that TLLC but we have more information because we add an accusation counter, i.e. we create a counter associated with each list. We fill the list in the same way but when the node receives the message which is already in the list, then we increment the counter associated with this link and delete the message identifier from this list. And here, the transmission of information message is as follows: The node randomly sends the message to a neighbor which has the minimum of counters.

After several attempts, we find an improvement: when a node receives a message that is already in a tabu list of its links, then the counter associated with the link where the message arrived is incremental. We describe precisely the process when a source node sends messages regularly in the Algorithm 1. Notations are given below.

Notations: Let $G = (V, E)$ be a graph with V, the set of nodes noted i (s for the sink and u for the source) and E, the set of links. For each nodes we have N_i, the set of i's neighbors. Let $L_{i,j}$ be the tabu list of messages on each link from a node i to a node j. We remark that $j \in N_i$. We recall that the tabu list has an arbitrary maximum capacity. Let $C_{i,j}$ be the accusation counter on each link from a node i to a node j.

To write clearly the algorithm, we need several functions. We have the basic functions of:
- a network protocol:
  - send(Message m, Node n): The node sends the message m to the node n.
  - receive(Message m, Node n): The node receives the message m from the node n.
- processing a list:
  - remove(Message m, ListofMessage L): Remove the message m from the list L.
  - push(Message m, ListofMessage L): Add the message m to the top of the list L and if the list is full then we remove the last message of the list.
- a random selection:
  - random(ListofNodes L): randomly select a node n from the list L and return n.

Algorithm 1 TLLCAC

Code for each node i except the sink

Initialization:
for all $j \in N_i$ **do**
  $L_{i,j} = \{\}$
  $C_{i,j} = 0$
end for
start task

Task:
upon receive (m, n)
for all $j \in N_i$ **do**
  if m $\in L_{i,j}$ **then**
    $C_{i,j} \leftarrow C_{i,j} + 1$ */*to accuse the link by which the message has been sent*/*
    if $j \neq n$ **then**
      $C_{i,n} \leftarrow C_{i,n} + 1$ */*to accuse the link by which the message is back*/*
    end if
    remove(m, $L_{i,j}$)
  end if
end for
$r \leftarrow$ random($\{j \in N_i \mid C_{i,j} = \min_{k \in N_i}(C_{i,k})\}$)
push(m, $L_{i,r}$)
send(m, r)

Code for the source node u

loop
  $r \leftarrow$ random($\{j \in N_u \mid C_{u,j} = \min_{k \in N_u}(C_{u,k})\}$)
  push(m, $L_{u,r}$)
  send(m, r)
end loop

3.1.3 Tabu Search on Cycles with Accusation Counter and Law of Probabilities

TLLCACLP is almost similar to TLLCAC. It uses the same information as TLLCAC, *i.e.*, we have an accusation counter and a tabu list for each link. Moreover, the list is filled in the same way. In contrast to TLLCAC that is deterministic because it sends the message to the neighbor which has the minimum counter. TLLCACLP sends randomly the message. It use a transition probability favoring links which have a small counter. Thus we define the probability that node u relays a message to v such as:

$$P(u,v) = \begin{cases} \frac{C(u,v)^{-1}}{\sum_{w \in N(u)} C(u,w)^{-1}} & \text{if } v \in N(u) \\ 0 & \text{otherwise} \end{cases}$$

where $N(u)$ is the set of vertices adjacent to a vertex u and $C(u,v)$ is the accusation counter of vertices from u to v.

3.2 Simulation's Results

We quickly abandoned the protocol TLLC because simulations showed that TLLC and TLLCAC are strictly equivalent for the first message sent. But for the following messages, TLLCAC is widely better than TLLC because TLLCAC keeps informations of passage of messages (accusation counter) and not TLLC.

We note that the protocol TLLCAC may not finish, *i.e.*, messages can turn indefinitely in a cycle. By example if the size of the tabu list is S and we have a cycle of size N, we suppose that the N nodes have a single minimum counter then with $S + 1$ messages in the cycle, the protocol does not detect the problem because the last message remove the first message from the list. We want that the protocol operates correctly. Thus, we must assume that the size of the list is always greater than the number of message in the network at a time t.

We compare TLLCAC and TLLCACLP over time, for that, we put in abscissa the order of the "$i - th$" message sent and in ordinate the average number of hops taken by the message. Figure 3.1 is a simulation on an arbitrary graph of 50 nodes and an average degree of 10.

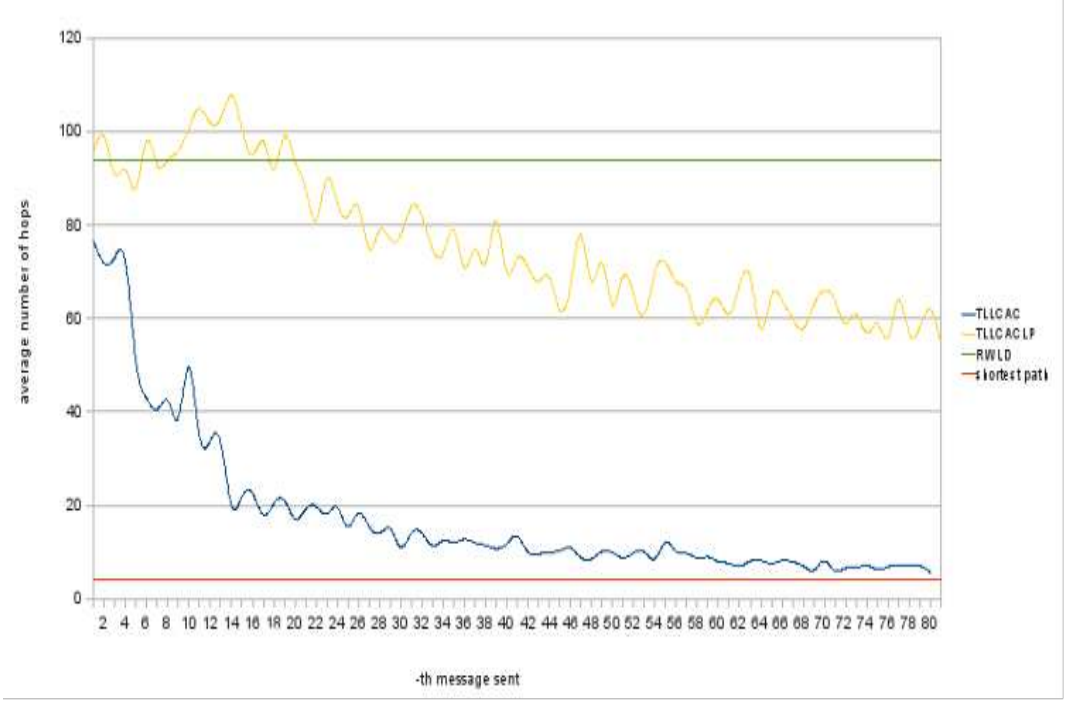


Figure 3.1: Average number of hops for arbitrary graphs

It seems that TLLCAC converges quickly to a spanning tree close to being the Breath-First Spanning Tree (BFST) and it is always better than TLLCACLP and RWLD. The average number of hops of TLLCACLP decreases slowly and from the 20-th message, it's better than SRW. To simulate TLLCACLP which is a probabilistic protocol, we compute the average of hops on all 1000 messages. Then, we simulate TLLCAC with the average of hops on the first messages.

3.2.1 Results of TLLCACLP

Figure 3.2 shows that TLLCACLP has good results because the average number of hops of RWLD is much greater. We observe that over time, the average of hops decreases fairly quickly at the beginning and more slowly after.

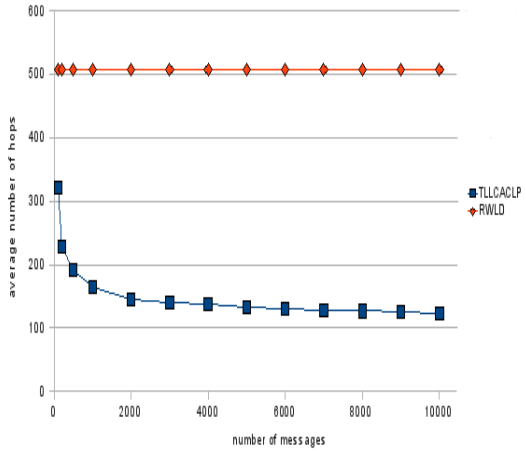


Figure 3.2: arbitrary graphs

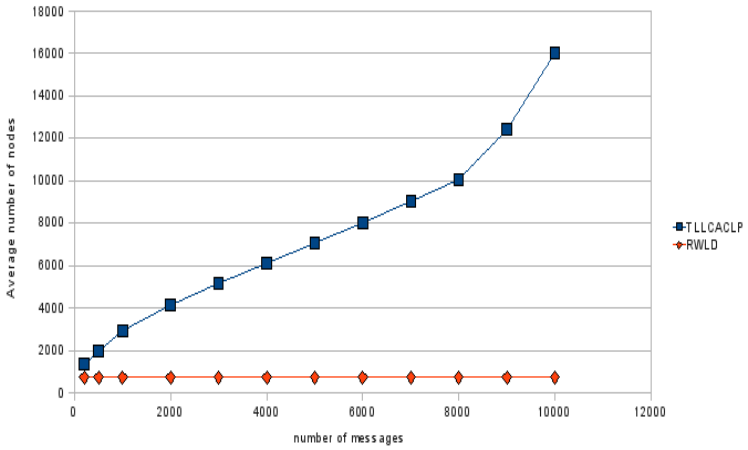


Figure 3.3: mickey graphs

Figure 3.4: Simulation results

Instead, Figure 3.9 shows a big problem on a mickey graph (see Figure 2.9 page 23) because the average number of hops drastically improves and RWLD is better than TLLCACLP. We can explain this result saying that the first messages are blocked in the arbitrary graph part including the source. Then, counters increase without detecting paths constituted of nodes of degree 2. So they lose all their meaning.

3.2.2 Results of TLLCAC

Simulation results of TLLCAC are given by Figure 3.5. They are really interesting because the average number of hops decreases extremely quickly. Simulations show another important result: TLLCAC converges to a spanning tree closed to being the BFST. In the next section, we studied precisely this conjecture.

3.3 Study of conjecture on TLLCAC

We observed that TLLCAC builds the shortest path between the sink and the source in most examples but not all. So, in a first step, we prove that the routing protocol TLLCAC converges to a spanning tree. In a second step, we present a simple case where TLLCAC does not converge to the BFST at each time.

3.3.1 The algorithm converges to a spanning tree

TLLCAC is a protocol based on the principle of accusation counter. We accuse the links of a node increasing the counter associated with the tabu list of each link. The counter of a link l increases when a message sent through l is back to the node of l. The sink node sends no message so neighboring nodes of the sink always have the counter of the link connecting the sink to its neighbor equal to zero.

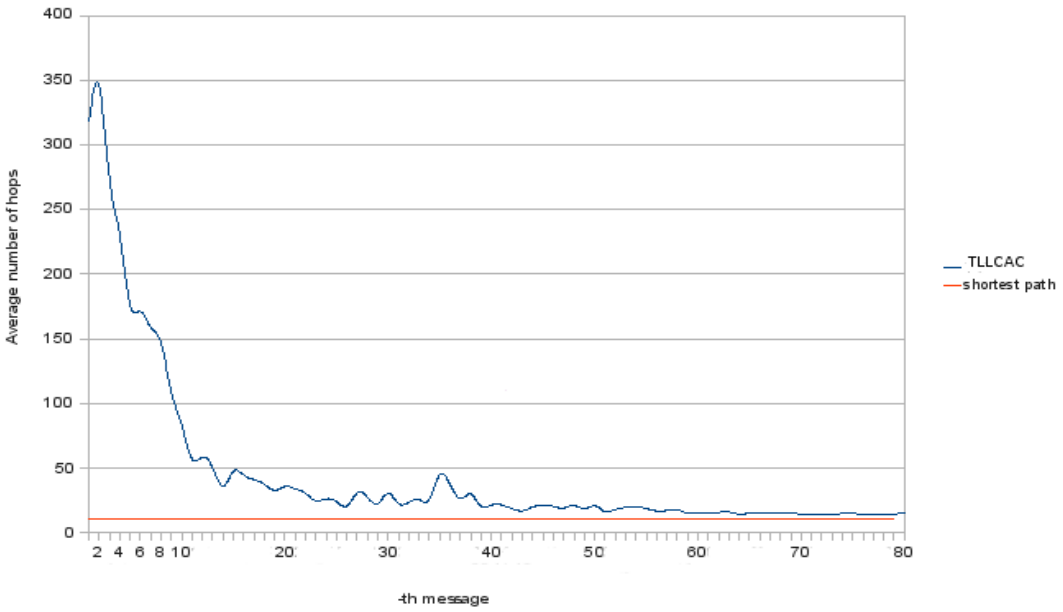


Figure 3.5: Simulation results of TLLCAC

Consequently, after a finite time, neighboring nodes of the sink have bounded counters. All messages arriving in the neighborhood of the sink are directly routed to the sink. Therefore, counters of links joining the neighborhood of the sink can not increase and they become bounded after a finite time. We have the same operation as previously which is repeated by the algorithm of TLLCAC until the source node. We deduce that after a finite time, all counters are bounded. We prove that a message is always delivered because a message can not turn in a network during an infinite time due to the fact that counters are bounded. Consequently, after a finite time, we have that messages always follow an elementary path between the source and the sink. Assuming that all nodes are source nodes, we obtain that TLLCAC converges to a spanning tree (Theorem 3). Afterwards, this Theorem 3 is formally proved. But this spanning tree may not be a BFST as shown latter by the counter example.

Preliminary : we assume that the size of the tabu list is big enough, i.e. size is always greater than the number of message in the networks (messages sent by the source but not yet received by the sink). Let $d(x, y) = \min(length(p))$ be the distance between two nodes with p, a path from a node x to a node y. Let $m(u, i)$ be the i-th message sent by the source u (s is the sink node). Let $p(m)$ be the path followed by the message m. Let $E_d = \{x \in V \mid d(x, s) = d\}$ be the set of nodes which are at distanced l from the sink. The source node is the node which periodically sends data messages. Let $C^i_{x,y}$ be the accusation counter of the link relaying nodes x and y after the passage of the i-th message. Hence, if all nodes are source node then all nodes periodically send data messages. We need to prove Lemma 1 using Property 1 and Property 2 to conclude with Theorem 3.

Property 1. *After a finite time, all counters are bounded.*

Proof. We have an inductive property on the natural $d \geq 1$:

$$\mathrm{P}(d) : \exists\, i \in \mathbb{N} \text{ such that } \forall\, j \geq i, \forall\, x \in E_d, \forall\, y \in E_{d-1},\ C^i_{x,y} = C^j_{x,y}$$

Initialization: if $d = 1$ then the message sent on the link $l(x, y)$ (here $y = s$) reaches s that sends none message, thus the counter $C^i_{x,y} = C^j_{x,y} = 0$.

Induction: We suppose that $\mathrm{P}(d)$ is true and we want to prove $\mathrm{P}(d+1)$ by contradiction. For that, let z be the node such that $x \in E_{d+1}$, we suppose that $C_{z,x}$ increases to infinity. However, according the algorithm 1, for the counter increases, we have two possibilities:
- The node z sends a message on the link $l(z,x)$ and this same message is back to the node z by any link.
- The node z sends a message on any link and this same message is back to the node z by the link $l(z,x)$.
So, $C_{z,x}$ increases to infinity if and only if at least one of the two possibilities is realized infinitely. Consequently, we will prove that these two possibilities can not be done infinitely and we have therefore proved that $\mathrm{P}(d+1)$ is true by contradiction.

First case: The message arrives at the node x which sends the message on any link and this message is back to the node z, and may also back to the node x implying that counters of x increase. As this operation is carried out an infinite number of times, counters of x increase to infinity but according to the induction hypothesis, the counter $C_{x,y}$ is bounded. Thus after some time, $C_{x,y}$ is the minimum counter and so, it is impossible that the message is back to z, $C_{z,x}$ is not increasing.

Second case: The node x sends an infinity of message by the link $l(z,x)$, so the counter $C_{x,z}$ must increase infinitely but it's impossible because the node sends the message on the link which has the minimum counter and as $C_{x,y}$ is bounded, after some time, it was $C_{x,y}$ the minimum counter.

Conclusion: $\mathrm{P}(d)$ is true, so all counters are bounded. □

Now we want to prove that messages are delivered with a finite number of hops

Property 2. *Messages are delivered with a finite number of hops.*

Proof. We suppose that a message turns in the network during an infinite time. In this case, the message must come back in nodes already visited. So, according the algorithm 1, counters increase. As the message passes an infinite number of times in a same node, then counters increase infinitely often. This contradicts the property 1. Thus messages are delivered with a finite number of hops. □

Lemma 1. *After a finite time, all messages follow an elementary path until sink.*

Proof. Since Property 1 and Property 2, we deduce that after a finite time, all messages follow an elementary path until sink. □

Theorem 3. *If all nodes are source nodes which send messages periodically, then TLLCAC converges to a spanning tree.*

Proof: In our network, all nodes are source nodes. Since Lemma 1, messages routed by TLLCAC follow an elementary path until sink after a finite time. Here, messages are sent by each node and so, the elementary paths cover all the network. Consequently TLLCAC has built a spanning tree in a finite time. We can say that TLLCAC converges to the spanning tree. □

3.3.2 A particular case

We remark that several simulations on the same graph give different number of hops between the source and the sink when TLLCAC has converged. We observe proceedings of TLLCAC step by step and we identify a simple structure of the graph where the spanning tree is not the BFST. The class of graphs is composed of two (or more) disjoint paths between the source and the node. For example, Figure 3.6 shows a graph with disjoint paths of size 1 and 3. This graph is a counter example because TLLCAC

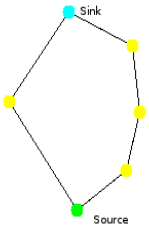


Figure 3.6: Disjoint paths graph

can stabilize on the path of length 3 between the sink and the node.

We now have a class of graphs for which TLLCAC does not converge to the BFST each time but the number of nodes of different level between the spanning tree built and the breath-first tree is one. We derive a class of graphs where the TLLCAC can converge to a spanning tree with a chosen number of nodes of different level compared to the BFST by construction (see Figure 3.7) because we repeat the construction of Figure 3.6.

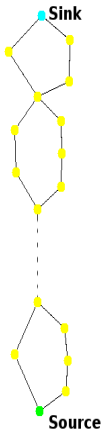


Figure 3.7: Worst extensible case

We are interested in the number of times when TLLCAC does not converge to the shortest path in class of graphs such that Figure 3.6 because this result can be generalized for graphs of Figure 3.7. So, we compute the percentage of times that the protocol converges to the longest paths. We vary the length of the paths between the sink and the source. We draw curves for the length of the short path and in abscissa, we have the difference between the short path and the long path.

Results on Figure 3.8 show two things:

- More the difference between the short path and long path is large, more the protocol is likely to converge to the shortest path.

- More the two paths are long, more the protocol is likely to not converge to the shortest path. But it appears that the error rate (the number of times when the protocol converge to the longest path) is bounded by 50% when we have two disjoint paths.

We simulate TLLCAC on graphs with a number n of disjoint paths. Simulations results allow us to propose a following conjecture.

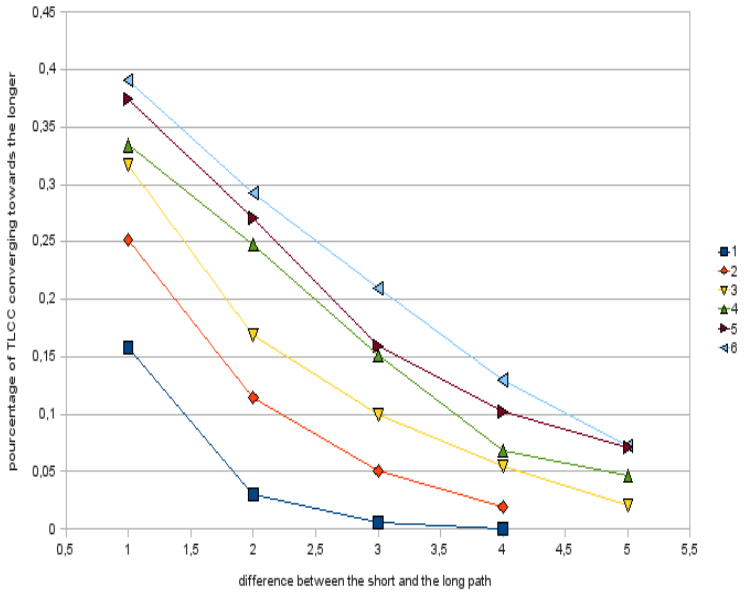


Figure 3.8: Simulation results for disjoint paths graphs

Conjecture 1. *The probability that TLLCAC converges to the shortest path is between* 1 *and* $\frac{1}{n}$.

3.4 Comparison between all protocols

It is difficult to compare the last two protocols with the previous because we have seen that these two protocols had a memory. Therefore we choose to compare the average number of hops of the first message. We must keep in mind that TLLCAC and TLLCACLP are more effective for the next messages in a "static network" but not necessarily in a "dynamic network".

3.4.1 Some examples

Figure 3.9 gives results on the lollipop graph. TLLCAC and TLLCACLP are just better than SRW but not others. We can explain this result saying that the lollipop is the worst case of SRW and protocols TLLCAC and TLLCACLP are equivalent as long as accusation counters are equals (for example when the message does not yet pass through the same node). It is this property that can explain that TLLCAC is better than TLLCACLP because the counter has more influence in TLLCAC.

We now present test on an arbitrary graph which conserves the same degree (see Figure 3.10). This time, TLLCAC is better than RWLD and TLLCACLP is roughly equal to RWLD but the tabu method in the message is always the best.

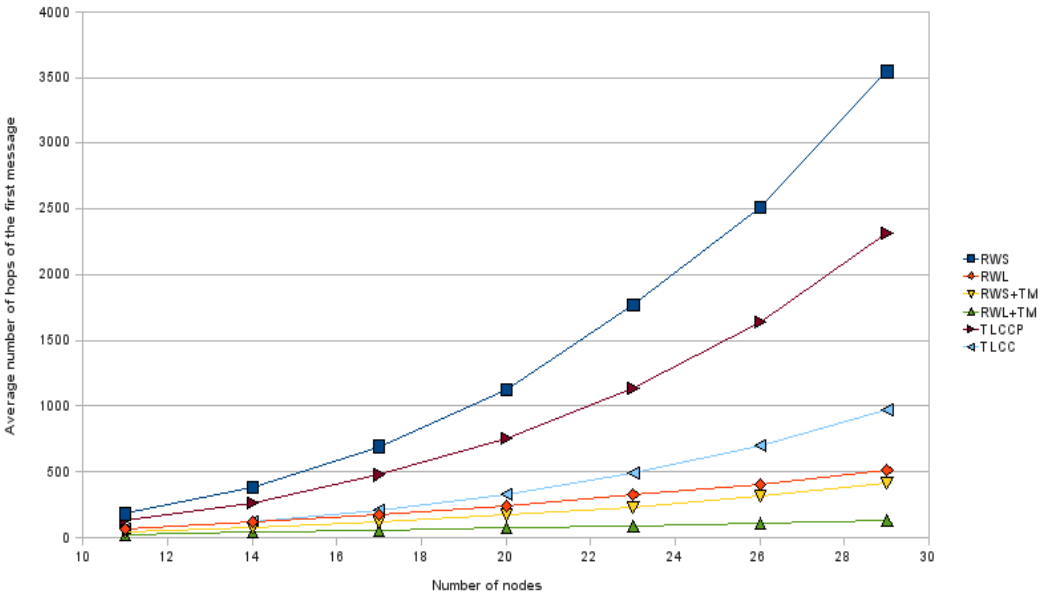


Figure 3.9: Simulation results for lollipop graphs

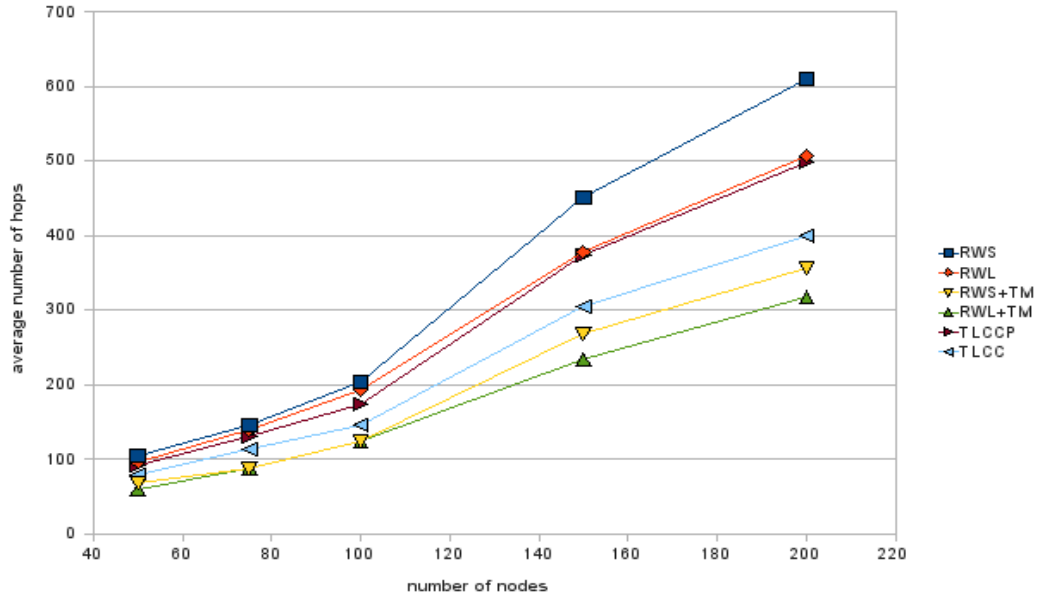


Figure 3.10: Simulation results for arbitrary graphs

3.4.2 Summary Table

We summarize simulations results carried out on all of our test graphs. The pack of class of graphs is given in the appendix from page 59 to page 64. We give the results with a number, number 1 being the most efficient algorithm and so on. When the average number of hops is equivalent, we put the same number. Some graphs are divided into different categories according the density, more details on the average degree in parentheses where we noted d the average degree of nodes. Protocols RW+TLM are tested with a size of list of 4.

Graphs/Protocols	SRW	RWLD	SRW+TLM	RWLD+TLM	TLLCAC	TLLCACLP
arbitrary ($8 < d$)	6	4	3	1	2	5
arbitrary ($d < 8$)	6	5	2	1	3	4
lollipop	6	3	2	1	4	5
4-paths-sink	6	3	2	1	4	5
mickey ($11 < d$)	6	3	2	1	4	5
mickey ($d < 6$)	6	5	2	1	3	4
mickey ($6 < d < 11$)	6	4	2	1	3	5
3-parts	6	4	2	1	3	5
chain	5	5	1	1	3	4
city	5	5	1	1	3	4
burger ($12 < d$)	6	2	3	1	4	5
burger ($8 < d < 12$)	6	3	2	1	4	5
burger ($d < 8$)	6	4	2	1	3	5
increasing degrees	4	6	1	5	2	3
flower	3	6	5	4	1	2

Part II

Study of Hybrid Protocols

Chapter 4

Study of Breath-First Search Tree

This part is devoted to the study of hybrid protocols. An hybrid protocol mixes two routing strategies. Here, when the network is stable, we use a routing along a spanning tree. Otherwise, after a topological change, the network is perturbated, then we use a probabilistic protocol until restabilization of the spanning tree. We need an algorithm which builds a spanning tree because it is used for the routing. We want to limit the number of hops because we are in a WSN. The protocol Breath-First Search Tree is adapted because it computes the minimum tree height. Moreover, we can use the BFST alone to route a message. The message is routed along the tree until the root that corresponds to the sink node in a WSN. Besides, we compare our routing hybrid protocols with the routing protocol BFST in the next chapter.

Since information on the stabilization is only local, we want to study precisely the stabilizing time of BFST. First we give the implemented algorithm of BFST. Then we formally prove Theorem 4 and Theorem 5 on the mean time of stabilization of the tree. Finally, we confirm them by simulations using the fact that the simulator offers a global view of network.

4.1 The Algorithm

The construction of the spanning tree by the BFST is described in [**?**]. To implement it, we give a level and a parent for each node. We have a timer that sends regularly the node level to its neighbors. We explain more precisely the update information:

- Nodes are initialized: level of sink with 1, other nodes with a random number different of 1 and the parent identifier to 0.
- Each node sends periodically its level to its neighbors according to a same initial period.
- When a node receives this message, it updates its information: its level takes the minimum of neighboring levels added one and its parent becomes the neighboring node with the minimum level and the minimum identifier.

We observe that the tree is stabilized after a certain time. Thus we create a detector of global stabilization in the simulator. It is a timer which periodically verifies the two following properties:

1. All nodes have the minimum of neighboring levels added one.
2. All nodes have a parent with a level less one.

This algorithm becomes a routing protocol because the data message is transmitted to the parent node until the sink node (the root).

4.2 Theoretical Stabilizing Time of BFST

We give a theorem on the stabilizing time of BFST. First, we set the notation, then, we are doing preliminary work on the algorithm and the property of stabilization. After, we write Theorem 4 working on "a global step" (see Definition 2), and prove it. Finally, we define "a global step" with the time and we express Theorem 5 based on the mean time.

Notation Let i be a node with s the sink node and n the number of nodes. The set of neighbors of i is noted $N(i)$. Let $L(i)$ be the level of the node $i \neq s$ and $i \in [2 \dots n]$. Let $P(i)$ be the parent of i, so $P(i) \in N(i)$. We have $L(s) = 1$ and $P(s) = \varnothing$. We define the distance between two nodes by $d(x, y) = \min(length(p))$ with p, a path from the node x to y.

Preliminary The algorithm stabilizes if and only if $BFST$ is true.

$$BFST \equiv \forall i \neq s, L(i) = L(P(i)) + 1 \wedge L(P(i)) = \min_{j \in N(i)} \{L(j)\}$$

A node i is stable if and only if $BFST(i)$ is true.

$$BFST(i) \equiv L(i) = L(P(i)) + 1 \wedge L(P(i)) = \min_{j \in N(i)} \{L(j)\}$$

Notice that $BFST \Leftrightarrow \forall i \neq s, BFST(i)$
The algorithm gives us the next node step for each node $i \neq s$:

- When a message is received from neighbor j, i update the know value $L(j)$ and i performs its level such that if $\neg BFST(i)$ then we set $P(i)$ to $\arg\min_{j \in N(i)}\{L(j)\}$ and we set $L(i)$ to $L(P(i)) + 1$.
- Broadcast $L(i)$ periodically, with a period P_o.

Definition 1. *Node step: The node has received a message from all its neighbors. A node step begins when the node receives a first message from a neighbor and it finishes when the node receives a message from its last neighbor. During this time, the node can receive several other messages but it ignores them. Another node step starts again when the node has received the last message of the first node step.*

Definition 2. *Global step: Each node performs at least one node step, i.e., all nodes have at least received a message from their neighbors.*

Furthermore, we know that whatever are the initial values for $L(i)$ the algorithm converges, *i.e.*, $BFST$ becomes true in a finite number of steps. For compute the mean time of stabilization, we need to evaluate how many steps and the mean time for a step.

Theorem 4. *Let d_{max} be $\max_{i \in [1\ ..\ n]}\{d(s, i)\}$, the maximum distance of a node from the sink. Then after d_{max} global steps, the stabilization is reached.*

Proof: Let an inductive property on the natural k :

$$\mathrm{P}(k) \begin{cases} \forall i \text{ such that } d(s,i) < k, \text{ we have } BFST(i) \wedge L(i) \leq k+1 \ (\star) \\ \forall i \text{ such that } d(s,i) = k, \text{ we have } BFST(i) \wedge L(i) = k+1) \ (\star\star) \\ \forall i \text{ such that } d(s,i) > k, \text{ we have } L(i) > k+1 \ (\star\star\star) \end{cases}$$

Like $\mathrm{P}(d_{max}) \Rightarrow \forall i \neq s, BFST(i) \Leftrightarrow BFST$, we show that $\mathrm{P}(k)$ is inductive with $k \in [0, d_{max}]$ and so that in at most d_{max} global steps, the stabilization is reached.

Initialization : $\mathrm{P}(0)$ is trivially true because it's the initialization phase of the algorithm.

Induction : If $\mathrm{P}(k)$ is true and the algorithm performs a global step, then we show that $\mathrm{P}(k+1)$ is true.
Since $\mathrm{P}(k)$, $\forall i,\ d(s,i) \leq k$, we have $BFST(i)$ true and $\forall i,\ L(i)$ and $P(i)$ do not change and thus $BFST(i)$ still holds after the node step.($\star$)
Let j be a node such that $d(s,j) = k+1$:
- Since $\mathrm{P}(k)$, we have $L(j) > k+1$
- Since $d(s,j) = k+1$, j has at least one neighbor i at distance k, thus $d(s,i) = k$ and $L(i) = k+1$ according $\mathrm{P}(k)$ and for others neighbors m, we must have $L(m) \geq k+1$ because $\forall m,\ d(s,m) \geq k$
Note that $L(i)$ is known by j since it has been at last updated and broadcast during the previous global step. And therefore $P(j)$ is set to i and $L(j)$ is set to $L(i)+1 = k+2$. This implies $BFST(j) \wedge L(j) \leq k+2$ is true.($\star\star$)
Let z be a node such that $d(s,z) > k+1$, then z is neighbor of others nodes at distance $\geq k+1$ and thus after the update of $L(z)$ if it occurs leads to $L(z) \geq k+2$.($\star\star\star$)
We can conclude that $\mathrm{P}(k+1)$ holds because ($\star$), ($\star\star$) and ($\star\star\star$) are satisfied. □

Theorem 5. *The mean time for stabilization* $= d_{max} \times (P_o + \mathrm{E}(d) + \mathrm{E}(t_{trans}))$

Proof: **Rating of the mean time of a global step:**
Let t_{trans} be the random variable for the time of transmission of a message and let d be the random variable for the drift time of a node (a drift time is the lag when a clock does not run at the exact right speed compared to another clock).
The mean time for a global step $= \mathrm{E}(P_o + d + t_{trans})$
$= P_o + \mathrm{E}(d) + \mathrm{E}(t_{trans})$
And with Theorem 4, we can conclude that the mean time for stabilization is $d_{max} \times (P_o + \mathrm{E}(d) + \mathrm{E}(t_{trans}))$ □

4.3 Experimental confirmation

We confirm the theoretical result. We simulate BFST on classic classes of graphs. We compute the mean time of stabilization from the first phase because all nodes have an arbitrary level. We present the basic case of chain graph in Figure 4.1. Then we show some other results for arbitrary graphs with the same average of degree in Figure 4.2. They clearly confirm that the mean time of stabilization of BFST is proportional to the maximum level of the tree. The diameter of a graph is the greatest distance between any pair of vertices. So the maximum level of the tree is always bounded by the diameter of graph.

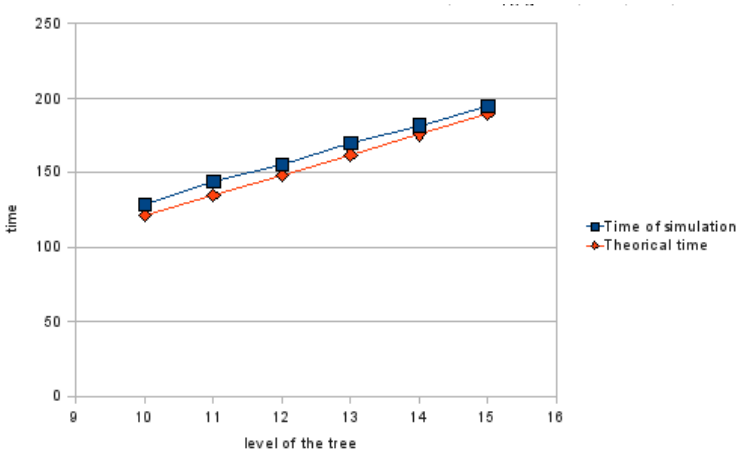


Figure 4.1: Simulation results for chain graphs

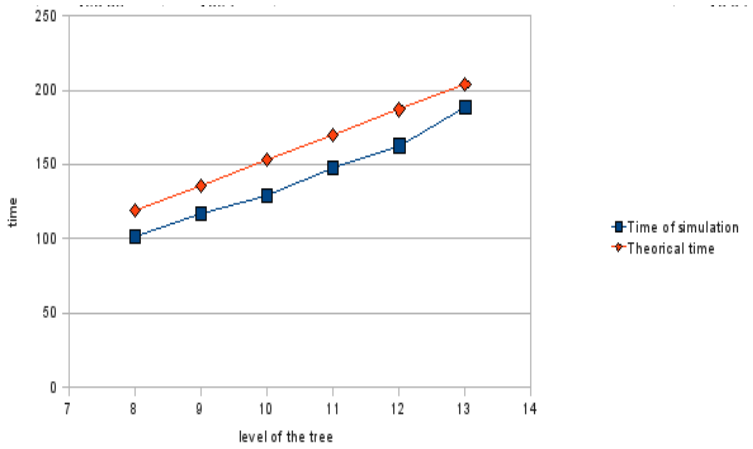


Figure 4.2: Simulation results for arbitrary graphs

Chapter 5

Comparison between Hybrid Protocols

In this chapter, we study hybrid routing protocols that is protocols mixing at least two routing strategies. Here we consider hybrid routing protocols using only two strategies:

1. Routing along a spanning tree.
2. Random walk routing.

We use these protocols in dynamic networks where links or nodes can be removed (knowing that a loss of node is equivalent to a node which loses all its links). In our protocols, the choice between the two strategies is made according to the stability of the network: when the network is stable and a spanning tree is available, messages are routed following the first strategy. Upon a topological change, the protocol temporarily switches to the second strategy.

Note that such kinds of hybrid protocols already exist in the literature. For example, in [**?**], Watteyne *et. al.* proposes a hybrid protocol mixing the breath-first search spanning tree protocol presented in [**?**] with a deterministic flooding. For the flooding protocol, they use heavy control information into the message to deterministically update the route. It is an important drawback for a WSN.

Here, we also use the breath-first search spanning tree protocol of [**?**] because it allows to route message along the shortest path when the network is stable. Moreover, it is self-stabilizing, so, it converges in a finite time after any topological changes. However, we replace the deterministic flooding by the previously introduced probabilistic methods. This latter allows to reduce message sizes, and consequently allows to save energy, which is crucial in WSN.

In this chapter, we propose and compare several hybrid protocols. For that, we need a network which is no static because in this case, all our protocols use BFST to route the message. So, we need to simulate protocols on a network implying the restabilization of the tree. First, we present the hybrid protocols studied. Secondly, we present simulation results on graphs where there is a single topological change. We observe that the BFST is better than our hybrid protocols excepted in a very particular case.

5.1 Presentation of our hybrid protocols

In the previous part, we have studied several probabilistic protocols using different methods. We choose to mix the best protocols of each method. Consequently, the first method is to modify the transition probability matrix. We proved by simulation that RWLD is better than SRW in most graphs, and we also characterized the class of graphs for which the reverse was true. So we propose the hybrid routing protocol mixing a Breath-First Search Tree with Random Walk using Local Degree (BFST+RWLD). The second method consist in forbidding the last visited nodes with this information carried by the message. We saw that this protocol is better than random walk protocols. We will test the new protocol Breath-First Search Tree and Random Walk using Local Degree with Tabu Lists in Messages

(BFST+RWLD+TLM). The size of the list will be 4 because previous simulations showed that it is certainly the optimal size. Finally, the last method is to put tabu lists in the node to avoid cycles saving links already traversed. This method is not the best but the protocol TLLCAC has interesting properties: it is better than SRW, and especially, it converges to the spanning tree closed to being the BFST in most cases. It is possible that TLLCAC has a better fault tolerance than BFST and more, it is the data message that constructs the stable path between the source and the sink. We present a third hybrid routing protocol named Breath-First Search Tree and Tabu Search on Cycles with Accusation Counter (BFST+TLLCAC).

In our three different hybrid protocols (BFST+RWLD, BFST+RWLD+TLM, and BFST+TLLCAC), we always mix BFST with probabilistic protocols because BFST is the best protocol when the network is static because it allows to route messages by the shortest path. And when we have a highly dynamic graph, the best protocols are probabilistic protocols. So when the network is static and BFST has converged to the tree. Our hybrid protocol routes the message with BFST. When there are faults causing recalculation of the tree, our hybrid protocol chooses to route with a probabilistic method. BFST always converges to the spanning tree with minimum level and supports errors, i.e. when the tree is found (protocol stabilized), we can have an error like the loss of a link then the protocol detects the error and correct it in a finite time. Consequently, we must evaluate the time taken by BFST to stabilize (or restabilize) because we need to switch between BFST and probabilistic protocols to minimize the routing time with probabilistic methods.

Information on the stabilization of the BFST is local for each node, *i.e.*, a node is considered stable if and only if its level and its parent node have not changed during a node step (see Definition 1 page 44). We remind that a node step is when the node has at least received a message from its neighbors.

We suppose that the BFST is in a stable phase. When an error induces a changing network topology then the BFST passes in a restabilization phase. At this moment, our hybrid protocols have two possibilities to transmit the fact that the BFST is not locally stable. The first strategy is to carry the information by the data message. A flag is added in the data message which specifies a node if it must use the BFST or the probabilistic protocols. The second strategy is to give the choice to the node. If the node has detected the restabilization of the tree, then it uses the probabilistic protocols during a certain period and after it reuses the BFST. We have quickly abandoned the first strategy because when the message arrives at a "stable" node (a stable part of the tree), the hybrid protocol uses the probabilistic method instead of using the BFST. Now, we evaluate the best swapping time between BFST and probabilistic methods.

The message is routed by a protocol decided by the node. We saw that the node knows locally its stability during the node step. So we want to find an estimation on the time T of a node step. Let P be the period of each node when they broadcast its levels (we call information messages, these messages). So a node waits at most a time P before to send the information messages. Let t_{trans} be the time of transmission of messages in the network. The node receives an information message after at most a time of $P + t_{trans}$. The communications are asynchronous because each node has a drift time d (a drift time is the lag when a clock does not run at the exact right speed compared to another clock). We do not know d or t_{trans} for each neighbor nodes of the node for which we want to evaluate T. So a reasonable idea is to take the average on all nodes of the network. Finally, we give $P + \mathrm{E}(d) + \mathrm{E}(t_{trans})$ like approximation of T, the time of a node step.

We assume that the network is in a stable phase. We sum up the operation of our hybrid protocols. We present steps of each node v when a topological change occurs:

1. v routes data messages using the BFST while v does not detect the instability of the tree.
2. If v detects the instability of the tree with a change of one attribute (level or parent), then v routes data messages using probabilistic protocols during a time $P + \mathrm{E}(d) + \mathrm{E}(t_{trans})$.
3. If one of attribute of v changes again before the end of the timer, then n restarts the timer to $P + \mathrm{E}(d) + \mathrm{E}(t_{trans})$.

4. After the end of the timer, v routes data messages using the BFST.

5.2 Simulations

For having interesting simulations, we need a loss of link having influence, especially, when we simulate protocols with a single fault. For example we can directly eliminate the links that does not connect a node and its parent node because the tree is not affect. Or if we disconnect a link connecting the node v and its parent p but v has a neighbor node with the same level of p, then BFST is immediately restabilized. Consequently, in general cases, a single loss of link has no influence and so, BFST is always better than our probabilistic protocols. We propose simulation results on networks where the loss of a link having a great influence on the restabilization phase of the spanning tree (see graphs on Figure 2.9 page 23 and Figure 5.2 page 50).

5.2.1 The mickey graphs

We simulate our hybrid protocols on a network where we have an unique loss of link. So, the network becomes a few dynamic. We choose the class of mickey graphs (see Figure 2.9 page 23) because the loss of the link on the short path constituted of nodes of degree 2 implies a long restabilization phase. We compute the average number of hops of the -th message arrived after the introduction of the fault. Results are given by Figure 5.1.

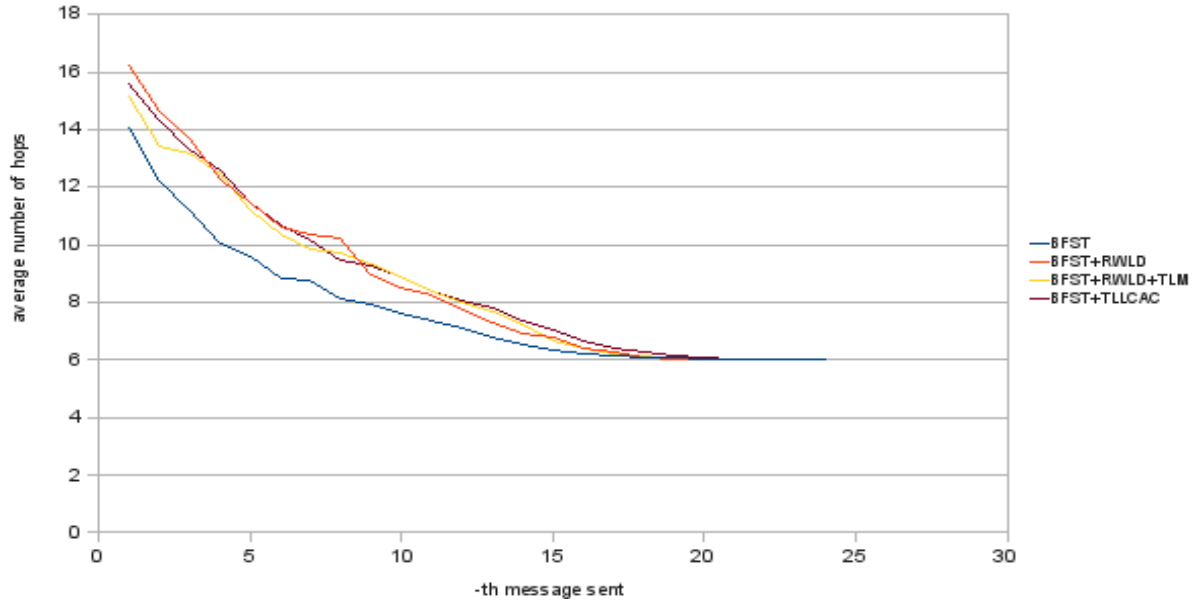


Figure 5.1: Simulation results for mickey graphs

We observe that the BFST is better than our hybrid protocols in this case. We obtain these results because BFST restabilizes quickly after one single fault. We need a network more dynamic because we want to show the best protocols in a WSN. In a future work, we will compare our protocols in a network with several correlated faults.

5.2.2 The “steffi graphs”

Here, we are interested in a particular class of graphs that we called steffi graphs (see Figure 5.2). The link that we disconnect is the dotted link. We constructed these graphs in order to slow the restabilization of the BFST. We saw that when the link is disconnected, the fault is quickly propagated to other nodes. So, we can suppose that BFST is slow enough and the swapping in our hybrid protocols is quickly efficient.

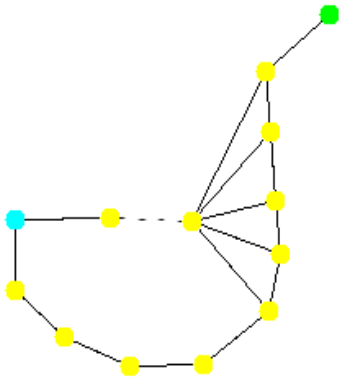


Figure 5.2: A steffi graph

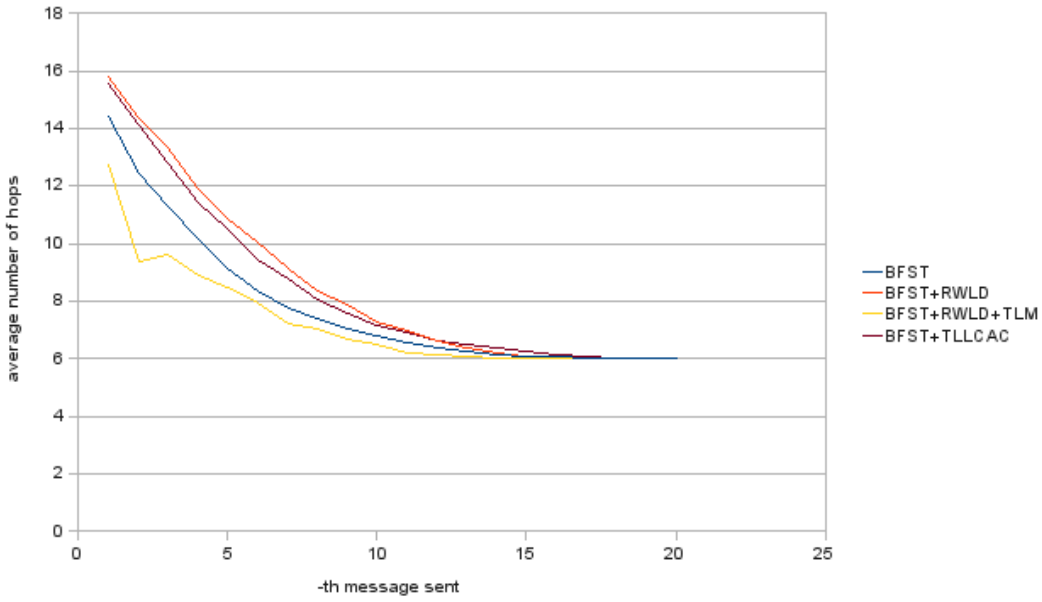


Figure 5.3: Simulation results for steffi graphs

Figure 5.3 shows simulation results. We see that BFST+RWLD+TLM is better than BFST but no the other hybrid protocols. We explain this result because the probabilistic protocol RWLD+TLM saves the “instable nodes” and so it is deterministic until the stabilized part of the network.

Simulations give contradictory results. It seems that the BFST is better than our hybrid protocols in most cases but we still managed to find an example of graphs with a hybrid protocol (BFST+RWLD+TLM)

which is better than BFST. So we need to introduce more faults to have a network more dynamic. But we must have correlated fault in terms of space and time because otherwise it is like having a single fault. We propose the following conjecture: if we introduce many correlated faults in a network, then, the network becomes sufficiently dynamic for that our hybrid protocols becomes better than the BFST.

Conclusion

We investigated the probabilistic routing methods for WSN. We used simulations to confirm that RWLD proposed by Yamashita in [**?**] is better than SRW in a vast majority of graphs. This is certainly due to the fact that RWLD avoids dense areas of graphs. Hence, we conjectured that SRW is better than RWLD on graphs where messages have to pass in a dense area. We formally proved that if the graph has nodes with an increasing degrees to the sink node, then SRW is better than RWLD. We can note that in a regular graph, SRW and RWLD are equivalent because they have the same transition probability matrix.

The drawback of random walk protocols is that messages can follow cycles in the network. Hence, we enriched random walks with tabu list to try to improve them. Indeed the aim of this technique is to prevent the message to follow cycles. We have two possibilities for storing information in tabu list to avoid cycles. The tabu list can be stored either in the messages or in the nodes. In the former case, we forbid the messages to traverse the nodes having their identifiers in the list. In the latter case, there are tabu list for each possible destination from the node. In each tabu list, we stored the message identifiers of the S last messages that traverse the corresponding edge, using this tabu list, the node can detect when a message follow a cycle. When tabu lists are stored in messages, we showed by simulations that RW+TLM with a tabu list size of 4 is better than RW in general cases. When tabu lists are stored in nodes, we propose two strategies using an accusation counter on links. The first strategy is to route messages through the links with the minimum accusation counter. This TLLCAC protocol converges to a spanning tree closed to the BFST with a high probability. The second strategy is to randomly route messages according a transition probability favoring links with a small accusation counter. After a finite time, this strategy presents good results in most of the graphs. But if routed messages have to traverse a path which only contain nodes of degree 2, then the average number of hops drastically increase.

We were also interested in hybrid routing protocols for WSN that mix two routing strategies. The first strategy uses probabilistic methods previously studied when the network is static and the second uses overlay-based methods when the network is dynamic.

Our hybrid routing protocols use the BFST when the spanning tree is stable because it allows messages to follow the shortest path. We evaluated the time of probabilistic routing when a topological change in the network is locally detected. Finally, we presented and compared simulation results of our probabilistic protocols on a network where a topological change occurs after an unique loss of link. These results are not convincing enough to conclude that our hybrid protocols are more efficient than the BFST in a dynamic network.

Consequently, an aim for future work will be to introduce several faults because we saw that an unique loss of link has not enough influence on the BFST except for a very particular graph. Moreover, these faults must be correlated in terms of space and time because otherwise it is like having a single fault.

We shall prove the different conjectures and simulation results of this work:

- The proof of Theorem 2 page 17 needs to be formally concluded.
- Prove that RW+TLM has better hitting and cover times than RW.
- Find exactly the property of graphs where TLLCAC converges to the BFST.
- Evaluate the probability that message routed by TLLCAC follows the shortest path.

- Estimate the average number of messages from which TLLCAC and TLLCACLP have better hitting and cover times than RW.

We will also continue the work on hybrid protocols. It will be interesting to simulate and to compare our hybrid protocols with the BFST when the network is highly dynamic and close to a real WSN, *i.e.*, a network in which occurs:

- Several correlated losses of links.
- Several losses of nodes (corresponding to a loss of all links of the node).
- Several losses of messages.
- Several discoveries (or rediscoveries) of links (or nodes).

We shall study the problem of the energy consumption, *i.e.*, to know what is the protocol most energy efficient and how can we optimize our protocols to save energy.

The author would like to thank its supervisors: Karine Altisen, Stéphane Devismes, and Pascal Lafourcade for their helpful comments.

Bibliography

[Avr99] D.R. Avresky. Embedding and reconfiguration of spanning trees in faulty hypercubes. In *IEEE Transactions on Parallel and Distributed Systems*, pages 211–222, 1999. (document)

[Bei] Nicklas Beijar. Zone routing protocol (zrp). (document)

[BJ98] A. Buczak and V. Jamalabad. Self-organization of a hetero geneous sensor network by genetic algorithms. *Intelligent Engineering Systems Through Artificial Neural Networks*, 8:259–264, 1998. (document)

[BSW69] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Commun. ACM*, 12(5):260–261, 1969. (document)

[BW90] G. Brightwell and P. Winkler. Maximum hitting time for random walks on graphs. *Journal of Random Structures and Algorithm*, 3:263–276, 1990. 1, 1.2.1

[CAD10] R.S. Mangrulkar Makarand R. Shahade C. A. Dhote, M.A.Pund. Article: Hybrid routing protocol with broadcast reply for mobile ad hoc network. *International Journal of Computer Applications*, 1(10):131–136, February 2010. Published By Foundation of Computer Science. (document)

[CC06] Tzung-Shi Chen and Chih-Ping Chu. Gathering-load-balanced tree protocol for wireless sensor networks. In *Proceeding of the 2006 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, 2006. (document)

[HC92] Shing-Tsaan Huang and Nian-Shing Chen. A self-stabilizing algorithm for constructing breadth-first trees. *Inf. Process. Lett.*, 41(2):109–117, 1992. (document), 4.1, 5

[JZ07] Wu Chengdong Zhang Yunzhou Jia Ji, P. and Zixi. Research of directed spanning tree routing protocol for wireless sensor networks. In *Proceeding of 2007 International Conference on Mechatronics and Automation*, 2007. (document)

[KKAB09] Azmi Halasa Khalid Kaabneh and Hussein Al-Bahadili. An effective location-based power conservation scheme for mobile ad hoc networks. *American Journal of Applied Sciences 6*, 9:1708–1713, 2009. (document)

[LG97] C.R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal on Selected Areas in Communications 15*, 7:1265–1275, 1997. (document)

[LR02] S. Lindsey and C.S. Raghavendra. Pegasis: power efficient gathering in sensor information systems. In *Proceedings of the IEEE Aerospace Conference*. Big Sky (Montana), March 2002. (document)

[MA01] A. Manjeshwar and D.P. Agrawal. Teen: a protocol for enhanced efficiency in wireless sensor networks. In *Proceedings of the 1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*. San Francisco, April 2001. (document)

[MA02] A. Manjeshwar and D.P. Agrawal. Apteen: a hybrid protocol for efficient routing and comprehensive information retrieval in wireless sensor networks. In *Proceedings of the 2nd International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile computing*. Ft. Lauderdale (Florida), April 2002. (document)

[Moh05] Dumont M. Teng-Sheng Moh. Moh, M. Evaluation of dynamic tree-based data gathering algorithms for wireless sensor networks. In *Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology*, pages 170–175, 2005. (document)

[MYA02] M. Youssef M. Younis and K. Arisha. Energy-aware routing in cluster-based sensor networks. In *Proceedings of the 10th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS2002)*. Fort Worth, October 2002. (document)

[PD03] E. Royer C. Perkins and S. Das. Ad hoc on-demand distance vector (aodv) routing. *RFC 3561*, July 2003. (document)

[SH88] A. Liestman S. Hedetniemi. A survey of gossiping and broadcasting in communication networks. *Networks 18*, 4:319–349, 1988. (document)

[SIY09] Izumi Kubo Satoshi Ikeda and Masafumi Yamashita. The hitting and cover times of random walks on finite graphs using local degree information. *Theoretical Computer Science*, 410:94–100, 2009. Contents lists available at ScienceDirect. (document), 1, 1.2.1, 5.2.2

[SK00] L. Subramanian and R.H. Katz. An architecture for building self configurable systems. In *Proceedings of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing*. Boston, August 2000. (document)

[SLS01] C.S. Raghavendra S. Lindsey and K. Sivalingam. Data gathering in sensor networks using the energy delay metric. In *Proceedings of the IPDPS Workshop on Issues in Wireless Networks and Mobile Computing*. San Francisco, April 2001. (document)

[TWD07] Isabelle Augé-Blum Thomas Watteyne, David Simplot-Ryl and Mischa Dohler. On using virtual coordinates for routing in the context of wireless sensor networks. In *18th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (pimrc'07)*, Athens, Greece, September 3-7 2007. IEEE. (document), 5

[WC09] Na Wang and Chorng Hwa Chang. Performance analysis of probabilistic multi-path geographic routing in wireless sensor networks. *International Journal of Communication Networks and Distributed Systems*, 2:16–39, 2009. (document)

[WHB99] J. Kulik W. Heinzelman and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, pages 174–185. Seattle, August 1999. (document)

[WHB00] A. Chandrakasan W. Heinzelman and H. Balakrishnan. Energy-efficient communication protocol for wireless sensor networks. In *Proceeding of the Hawaii International Conference System Sciences*. Hawaii, January 2000. (document)

[YNY10] Kunihiko Sadakane Yoshiaki Nonaka, Hirotaka Ono and Masafumi Yamashita. The hitting and cover times of metropolis walks. *Theoretical Computer Science*, 411:1889–1894, 2010. Contents lists available at ScienceDirect. (document)

[ZKW09] Tong Libiao Zheng Kai and Lu Wenjun. Location-based routing algorithms for wireless sensor network. *ZTE Communications*, 2009. (document)

[ZZ06] L.y. Zhang Zhenjiang. An energy-efficient redundant nodes tree mechanism for wireless sensor networks. In *Proceeding of 2006 International Conference on Systems and Networks Communication (ICSNC '06)*, 2006. (document)

Appendix A

The tool “sinalgo”

A.1 Overview

Sinalgo is a simulation framework for testing and validating network algorithms. Unlike most other network simulators, which spend most time simulating the different layers of the network stack, Sinalgo focuses on the verification of network algorithms, and abstracts from the underlying layers : It offers a message passing view of the network, which captures well the view of actual network devices. Sinalgo was designed, but is not limited to simulate wireless networks.

The key to successful development of network algorithms is a comprehensive test suite. Thanks to the fast algorithm prototyping in JAVA, Sinalgo offers itself as a first test environment, prior to deploy the algorithm to the hardware. Prototyping in JAVA instead of the hardware specific language is not only much faster and easier, but also simplifies debugging. Sinalgo offers a broad set of network conditions, under which you may test your algorithms. In addition, Sinalgo may be used as a stand-alone application to obtain simulation results in network algorithms research.

Sinalgo’s view of network devices is close to the view of real hardware devices (e.g. in TinyOS): A node may send a message to a specific neighbor or all its neighbors, react to received messages, set timers to schedule actions in the future, and much more.

Some of the key features of Sinalgo :

- Quick prototyping of your network algorithms in JAVA
- Straight forward extensibility to cover nearly any simulation scenario
- Many built-in, but still adjustable plug-ins
- High performance - run simulations with 100000s of nodes in acceptable time
- Support for 2D and 3D
- Asynchronous and synchronous simulation
- Customizable visualization of the network graph
- Platform independent - the project is written in Java
- Sinalgo is for free, published under a BSD license

To guarantee easy extensibility, Sinalgo offers a set of extension points, the so called models. The following list gives an overview of the available models, to each of which you may add your own extension. To facilitate your life, Sinalgo ships with a set of frequently used models.

- The mobility model describes how the nodes change their position over time. Examples are random waypoint, random walk, random direction, and many others.
- The connectivity model defines when two nodes are in communication range. The best known examples are the unit disk graph (UDG) and the quasi-UDG (QUDG).
- The distribution model is responsible to initially place the network nodes in the simulation area. E.g. place the nodes randomly, evenly distributed on a line or grid or according to a stationary regime of a mobility model.
- Use the interference model to define whether simultaneous message transmissions may interfere.
- The reliability model is a simplified form of the interference model and lets you define for every message whether it should arrive or not. E.g. drop one percent of all messages.
- Last but not least, the transmission model lets you define how long a message takes until it arrives at its destination.

A.2 Our use of the simulator

We mainly use the default models which are already implemented by sinalgo (we see an exception just after). Thus we are interested in the behavior of a node and to achieve a routing protocol, we are implemented classes “node”, “message”, “timer” and exceptionnaly “edges”, essentially to visualize our protocols. These classes and mainly methods are already defined abstractly in sinalgo.

The class Node It’s the most important class, here, we implement the node behavior. In first time, we initialize the node, after, the most interessant, we characterize its actions when it receive a message, i.e. we can send messages, compute a hash table, start a timer, ... And finally we choose how draw the node on the GUI (graphic mode). Each node is represented by an unique identifier (ID) and we can collect a collection of all edges outgoing from this node, thus a collection of all the neighboring nodes.

The class Message These objects allow communication between the nodes, they contain informations that we want to convey in our network.

The class Timer A timer is an object that allows a node to schedule a task in the future. When the task is due, the timer wakes up the node and performs the given task, it’s the method void fire(), which contains the task this timer needs to perform.

A.3 The asynchrony in sinalgo

Presentation The simulator is event-driven, i.e. there is an event queue that will happen in the future, this queue is ordered by the date of future events. When two events must arrive at the same time, they are strictly ordered by their (unique) ID, there are only two events upon which nodes react : Arriving messages and timer events.

Algorithm while (the queue is not empty)
do {
  extract the event with the lowest priority in the queue
  the current time of sinalgo is updated by the date of event
  the event is processed and it generates possibly other events
}
end while

The problem The asynchronous model of sinalgo is totally deterministic : events that occur in the same simulation time are ordered in the same way (by the ID) all the time. Thus isn't really an asynchronous model.

Our solution We don't change the mechanism of the simulator, we play on the transimission time of a message by being careful to preserve the character FIFO of a communication canal. For that, for each message sent, we choose at random the time of transmission of this message. And to maintain the order of sending messages, we store in the node which send the arrival date of the last message sent, this date must be newer than the following date.

A.4 Assumptions made on the simulated model

- Perfect detector participants : never says a neighbor is no longer there whereas it is still, and always ends by informing a neighbor is no longer present (in finite time).
- "fair lossy" : to an infinity of sending corrrespond an infinity of reception, this assumption is necessary to validate the alternating bit protocol (see 3.1).
- Unsilent link failure : if a link is broken then the nodes around realize it in time 0 or 1.
- The loss of node is equivalent to the loss of all its connections.
- The network must be connected and it must conserve its connectedness.
- Asynchronous communication.
- No mobility.
- No interference but we can lose message.
- All messages are "broadcast".

A.5 The loss of message

To resolve this problem, we choose to implement the alternating bit protocol. We hide this one because it is a different problem that our routing problem. For that we implement a class node between the abstract class node of sinalgo and others class node of our routing protocols and thus we have a new method "send" using the alternating bit and a new method "action" called when the node receive a message using the alternating bit.

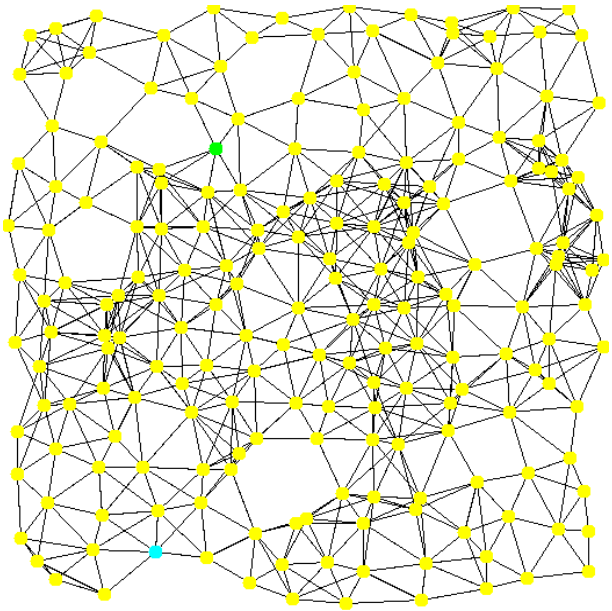
Appendix B

Pack of Graphs

In this chapter we describe the graphs we tested, the sink node is always in blue and the source node is always in green.

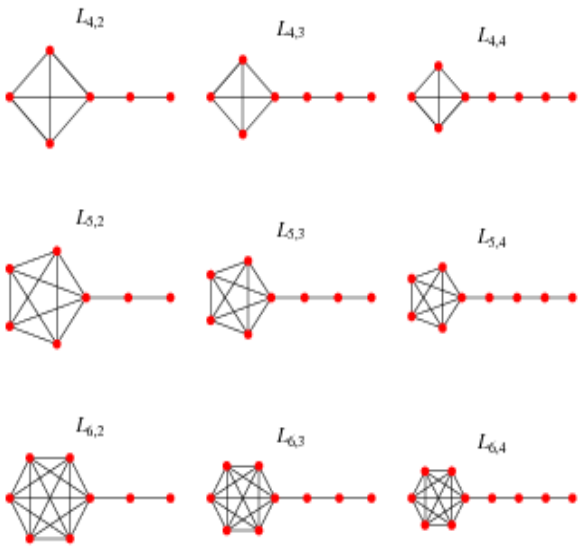
B.1 Arbitrary Graphs

These graphs are randomly built with an arbitrary topology and an average degree of 8

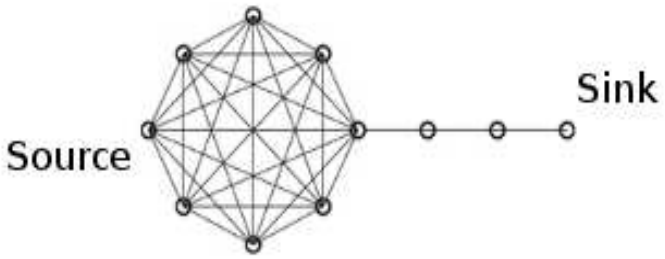


B.2 Lollipop Graphs

A lollipop graph is a complete graph with a tail (i.e. a path graph), by example:

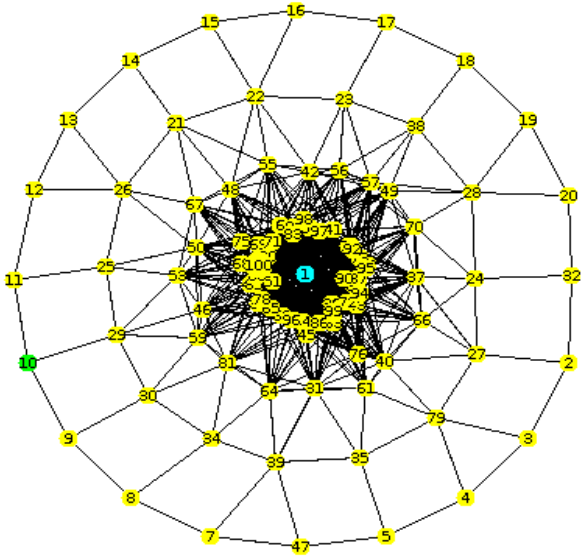


But to have the maximum of the hitting time among all graphs and so the upper bound of the hitting time, we respect the following proportions for a lollipop graph of order n : the complete graph is of order $\frac{2n}{3}$ and the path graph is of order $\frac{n}{3}$. And to perform experiments we place the sink node at the end of the path graph and the source node in the complete graph. Example with a lollipop of order 11:



B.3 Graphs with an increasing density until sink

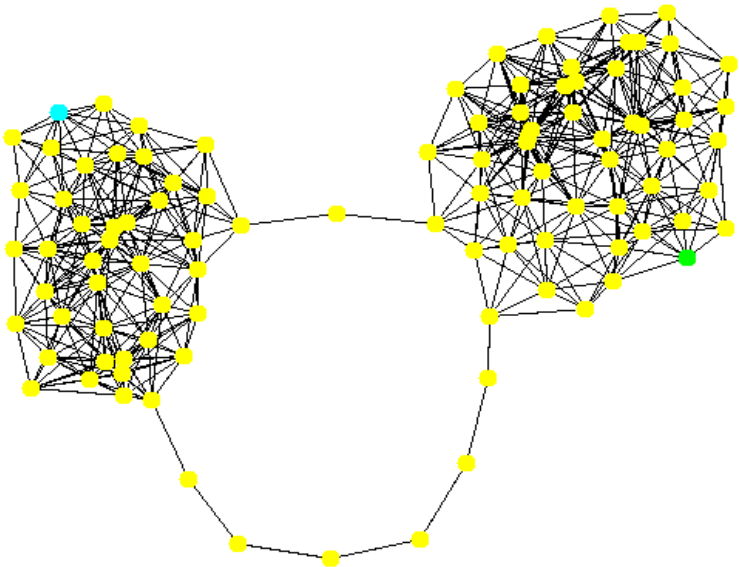
This graph has a circular shape because the connectivity model is the Unit Disk Graph connectivity, and we want to avoid the side effects. Recall that the sink node is in the center with the number 1 and is blue whereas the source node is green with a number 10, this node periodically send messages to the calculate the average number of hops. By example:



We note this graph: “increasing density”.

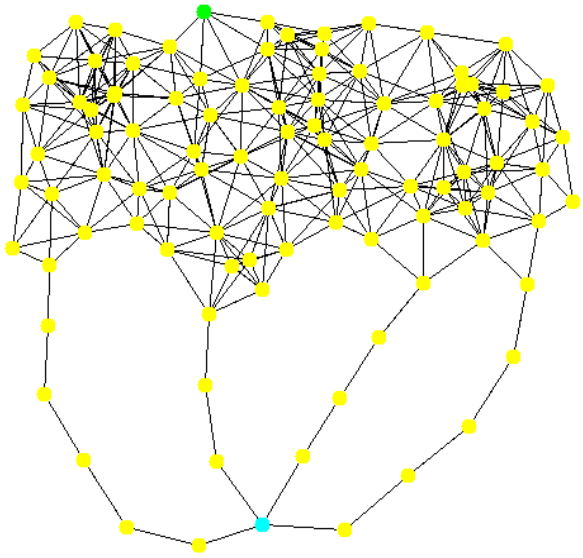
B.4 Mickey Graphs

There are 2 parts with one which has the sink and the other has the source, connected by a short path and a long path. For example:



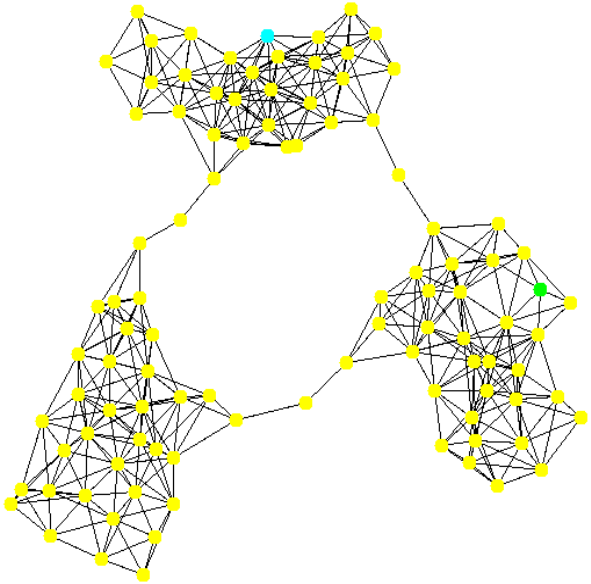
B.5 4-paths-sink Graphs

There are 4 paths between a classic part (general graph) and the sink. For example:



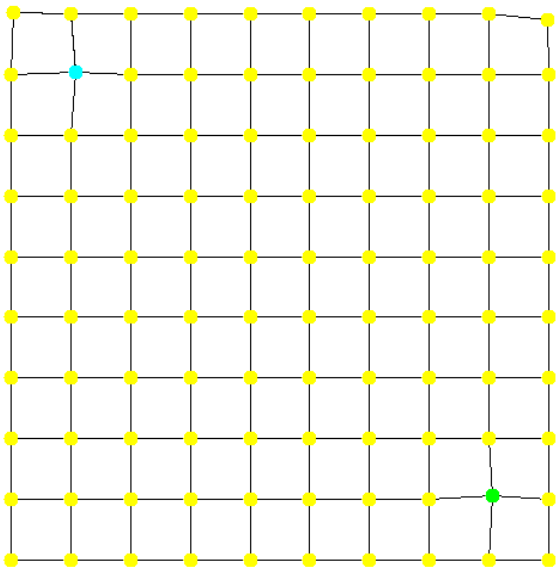
B.6 3-parts Graphs

There are 3 parts connected by paths. For example:



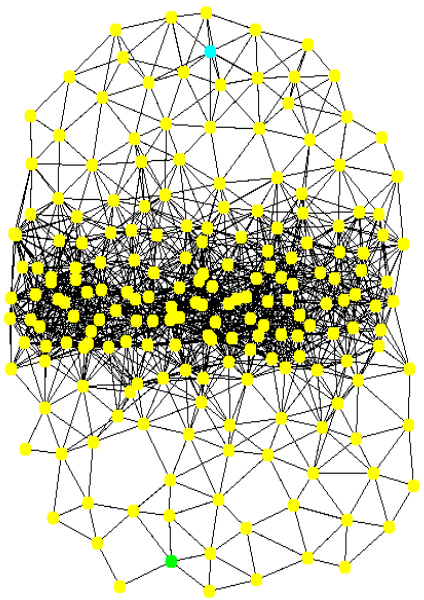
B.7 City Graphs

This graph is often studied in theory of graphs, it's a grid. For our simulations, the place of the sink and the source is important because if there are on the side, RWLD is slightly better than SRW and if there are about the center, SRW is slightly better than RWLD. In our summary table we suppose that SRW and RWLD are equivalents. An example of "city" graph:



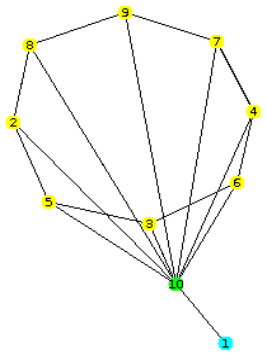
B.8 Burger Graphs

It is a graph with big dense area between the sink node and the source node.



B.9 Flower Graphs

This graph is explain in the chapter on “RW+TLM” because it’s a counter-example, we just show the picture.


9
8
7
4
2
6
5
3
10
1