

DoTRo: A New Dominating Tree Routing Algorithm for Efficient and Fault-Tolerant Leader Election in WSNs and IoT Networks

Ahcène Bounceur, Madani Bezoui, Loïc Lagadec, Reinhardt Euler, Abdelkader Laouid, Mohammad Hammoudeh

▶ **To cite this version:**

Ahcène Bounceur, Madani Bezoui, Loïc Lagadec, Reinhardt Euler, Abdelkader Laouid, et al.. DoTRo: A New Dominating Tree Routing Algorithm for Efficient and Fault-Tolerant Leader Election in WSNs and IoT Networks. Renault É., Boumerdassi S., Bouzefrane S. (eds). Mobile, Secure, and Programmable Networking. MSPN 2018, 11005, Springer, Cham, pp.42-53, 2019, Lecture Notes in Computer Science, 978-3-030-03100-8. 10.1007/978-3-030-03101-5_5 . hal-02501870

HAL Id: hal-02501870
https://hal.univ-brest.fr/hal-02501870

Submitted on 14 Mar 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DoTRo: A New Dominating Tree Routing Algorithm for Efficient and Fault-Tolerant Leader Election in WSNs and IoT Networks*

Ahcène Bounceur[1,2,5], Madani Bezoui[1,3], Loic Lagadec[1,4], Reinhardt Euler[1,2], Laouid Abdelkader[5,6], and Mohammad Hammoudeh[7]

[1] Lab-STICC CNRS UMR 6285
[2] Université de Bretagne Occidentale, Brest, France
[3] Department of Mathematics, Université de Boumerdes, Boumerdes, Algeria
[4] ENSTA Bretagne, Brest
[5] LIMED Laboratory, Bejaia, Algeria
[6] University of El-oued, Algeria
[7] Manchester Metropolitan University, Manchester, UK
Ahcene.Bounceur@univ-brest.fr

Abstract. A leader node in Ad hoc networks and especially in WSNs and IoT networks is needed in many cases, for example to generate keys for encryption/decryption, to find a node with minimum energy or situated in an extreme part of the network. In our work, we need as a leader the node situated on the extreme left of the network to start the process of finding its boundary nodes. These nodes will be used to monitor any sensitive, dangerous or non-accessible site. For this kind of applications, algorithms must be robust and fault-tolerant since it is difficult and even impossible to intervene if a node fails. Such a situation can be catastrophic in case that this node is the leader. In this paper, we present a new algorithm called DoTRo, which is based on a tree routing protocol. It starts from local leaders which will start the process of flooding to determine a spanning tree. During this process their value will be routed. If two spanning trees meet each other then the tree routing the best value will continue its process while the other tree will stop it. The remaining tree is the dominating one and its root will be the leader. This algorithm turns out to be low energy consuming with reduction rates that can exceed 85%. It is efficient and fault-tolerant since it works in the case where any node can fail and in the case where the network is disconnected.

Keywords: Wireless Sensor Network · IoT · Leader Election · Distributed algorithms · Dominating Tree Routing.

1 Introduction and related work

This paper comes within the context of secured sites where one needs to find the boundary nodes of wireless sensor and IoT networks. Many algorithms exist

* This project is supported by the French National Research Agency ANR PERSEPTEUR - REF: ANR-14-CE24-0017.

in the literature. A recent algorithm, called D-LPCN [1], can be used for this purpose. This algorithm starts from the node which is on the extreme left of the network, that we suppose to be embedded in the plane with nodes being identified by their coordinates. To find this particular node, one can use any Leader Election algorithm, which is a process of electing one particular node in a network. Usually, the leader process is required to play a particular role for coordination or control purposes. There is no solution for this problem in the case of anonymous systems [2], i.e., systems with nodes having no identifiers. In other words, there is no way to differentiate a process p_i from another process p_j. Due to this problem, we assume in this paper that each process p_i has a unique identifier id_i. Moreover, it is assumed that the identifiers can be compared with each other.

In the literature, one can define two main families of leader election methods. The leader election for ring topologies and the leader election for arbitrary topologies (e.g., ad hoc networks).

Regarding the first case of ring topologies, authors of [3] have presented an algorithm for bidirectional rings, where each process has a left and a right neighbor, and where it can send and receive messages from any neighbor. Authors of [4] have presented an algorithm for unidirectional rings, in which the channels are receiving messages in FIFO mode. As in [3], initially, all processes compete to be elected as a leader, and execute consecutive rounds to that end. During each round, at most half of the processes that are competitors remain competitors in the next round.

Regarding the second case of arbitrary topologies, in [5] and [6], two leader election protocols have been presented for static networks. In these algorithms, several minimum-weight spanning trees are established, which will be reduced to only one spanning tree. Then the root will be the leader. In [7], two leader election algorithms have been proposed for mobile ad hoc networks. The algorithms assume that each connected component of the graph has exactly one leader. They are based on a routing algorithm TORA presented in [8]. Another algorithm is presented in [9, 10] for asynchronous mobile ad hoc networks AEFA (Asynchronous Extrema Finding Algorithm), which is a weakly self-stabilizing algorithm in which each node possesses some weight representing the criteria to elect the best-node. It constructs and maintains a spanning tree using the diffusion computation to elect a leader. The paper [11] introduces a leader election algorithm for mobile ad hoc networks, which is based on an extrema-finding concept that elects a unique node and, on the basis of specific characteristics, outperforms all the other nodes in the network. Another algorithm is proposed in [12] for the election of a leader in an asynchronous network with dynamically changing communication topology. In the last decade, several clustering and leader election algorithms has been developed and tested to address the challenges of communication efficiency and fault tolerance in Wireless Sensor Networks (WSN) [13, 14].

In this paper, we propose a new algorithm for arbitrary networks. This algorithm starts with a given set of local minima, each of which will start as a

root the process of flooding in order to determine a spanning tree on which to route its value. If two spanning trees meet, the one routing the better value will continue the flooding process and the other one will stop it. After a given time, only one spanning tree will remain and its root will be the leader.

The remainder of the paper is organized as follows: Section 2 presents the classical Minimum Finding algorithm. Section 3 introduces the Local Minima Finding algorithm. Section 4 is dedicated to the proposed DoTRo algorithm. The used simulator is briefly presented in Section 5 and the simulation results are presented in Section 6. Finally, Section 7 concludes the paper.

2 The Minimum Finding Algorithm

In this section, we will present a distributed algorithm that allows to determine a leader node representing the node with minimum or maximum value v. This value can represent the local energy of the battery, the residual energy, the x-coordinate in a network, etc. Let us first define in Table 1 the functions used in the algorithms that will be presented in this paper.

Table 1. Functions of the proposed algorithms.

Function	Definition
`getId()`	returns the node identifier
`delay(t)`	waits t milliseconds before going to the next instruction
`stop()`	stops the execution of the program
`send(a,b)`	sends the message a to the sensor node having the identifier b or in a broadcast (if $b = *$)
`read()`	waiting for receipt of messages. This function is blocking, which means that if there is no received message any more, it remains blocked in this instruction
`read(t)`	waiting for receipt of messages. If there is no received message after t milliseconds then the execution will continue and go to the next instruction

The Minimum Finding Algorithm presented in [15, 16] relies on the tree-based broadcast algorithm. Indeed, it can also be used to find the maximum value. The principle of this algorithm can be described as follows. First, each node of the network assigns its local value to the variable x_{min} assumed to represent the minimum value of the network (the leader). Then it will broadcast this value and wait for other x_{min} values coming from its neighbors. If a received value x_{min} is less than its local x_{min} value then this one will be updated and

broadcasted again. This process is repeated by each node as long as a received value is less than its local x_{min} value. After a certain time t_{max}, there will be only one sensor node that has not received a value that is smaller than its local x_{min} value. This node is the leader. The pseudo-code of this process is given by Algorithm 1, where t_0 is the time of the first execution of the algorithm, which can correspond to the first powering-on of a sensor node, t_c the current local time of a sensor node, and t_{max} the maximally tolerated running time of the algorithm from the first execution to the current time of a sensor node.

Algorithm 1 *MinFind*: The pseudo-code of the classical Leader Election Algorithm

Input: wt, x
Output: *leader*
1: $leader$ = true
2: $x_{min} = x$
3: send(x_{min}, *)
4: **while** (true) **do**
5: x_r = read(wt)
6: **if** (x_r == null) **then**
7: stop()
8: **end if**
9: **if** ($x_r < x_{min}$) **then**
10: $leader$ = false
11: $x_{min} = x_r$
12: send(x_{min}, *);
13: **end if**
14: **end while**

3 The Local Minima Finding algorithm

A local minimum node, also called *Local Leader*, is the node which has no neighbor with a value smaller than its own value. But, this value is not necessarily a global minimum.

The Local Minima Finding (LMF) Algorithm uses the same principle as the previously presented *MinFind* algorithm to determine if a node is a local minimum or not, with the exception that each node will send its coordinates only once, and after reception of messages from all its neighbors, it decides if it is a local minimum or not in case it has received a smaller value than its own. The algorithm of finding local minima is given as follows:

Algorithm 2 *LMF*: The pseudo-code of the Local Minima Finding Algorithm

Input: wt, v
Output: $local_min$
1: $local_min$ = true;
2: $x_{min} = v$;
3: send(x_{min}, *);
4: **while** ((x = read(wt))$\neq$ null) and $local_min$) **do**
5: **if** ($x < x_{min}$) **then**
6: $local_min$ = false;
7: **end if**
8: **end while**

4 The proposed DoTRo algorithm

4.1 The DoTRo algorithm

The DoTRo algorithm is is based on a tree routing protocol. It starts from local leaders which will run, as a root, the process of flooding [19] to determine a spanning tree. During this process the value of the leader (root) will be routed. If two spanning trees meet each other then the tree routing the best value will continue its process while the other one will stop it. Based on the example of Figure 1, we present in the following the main steps of the DoTRo algorithm, where we assume that the leader is the node having the maximum value:

1. *Step 1:* For the network of Figure 1(a) we run the LMF algorithm (cf. Algorithm 2) to determine the local minima. The obtained result is shown by Figure 1(b) where we have two local minima: 1 and 4 because they are the only nodes that don't have any neighbor with a value smaller than their own value.
2. *Step 2:* Each local minimum will start the flooding process to route the leader value (local minimum) over the tree (cf. Figure 1(c)).
3. *Step 3:* If two trees meet, as is the case of the center node in Figure 1(c), the red tree chooses the center node with value 1 and the blue tree will choose the same center node with value 4. Since 1 is less than 4, the center branch of the blue tree will stop the flooding process, whereas the other branch will continue, and the center branch of the red tree will continue the flooding process. The obtained result is shown by Figure 1(d). Figures 1(e) and (g) show another meeting and Figures 1(f) and (h) show the result of the DoTRo algorithm after this meeting.
4. *Step 4:* Each local minimum will wait for a given time, assumed to be sufficient to finish the process of flooding. If after this time there is no received message anymore, the corresponding local minimum will become the leader.

4.2 The DoTRo pseudo-code

The pseudo-code of the DoTRo algorithm is given by Algorithm 3. It is composed of three main parts that are: 1) the initialization part (lines 1 to 4) where each

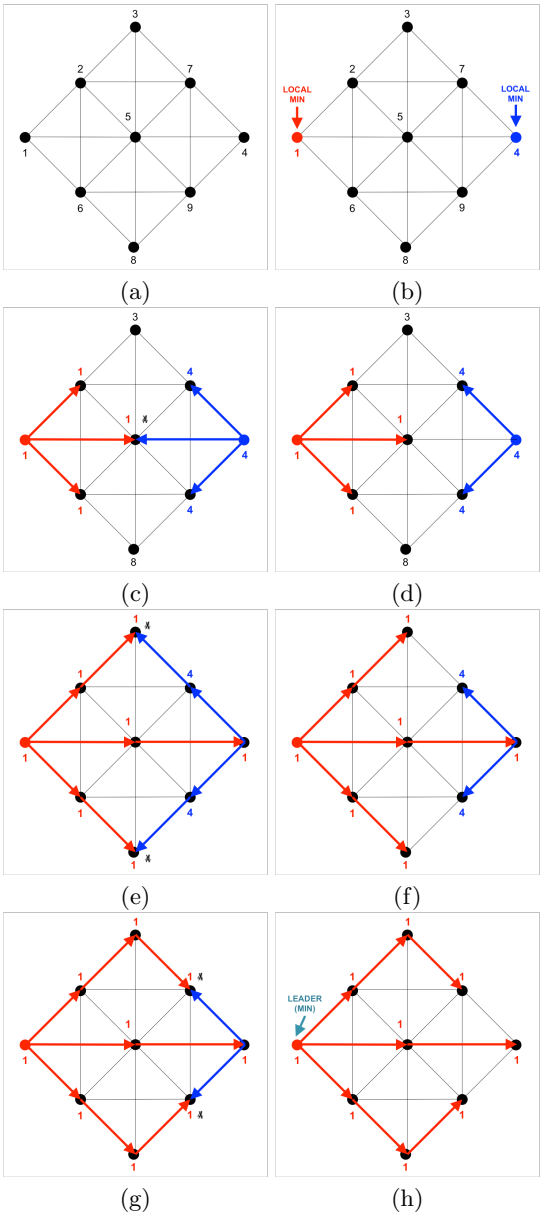


Fig. 1. DoTRo algorithm illustration.

node is considered as a leader (line 4), 2) the LMF algorithm (line 5 and from line 8 to line 18) and 3) the flooding process (from line 21 to line 30). The second part is explained above and in the third part, each node will wait for a message representing the value of the local minimum routed during the flooding process. If the received value is less than its own value then it will route it to continue the flooding process, and it will be considered as a non-leader node. Otherwise, the node will do nothing, which will stop the process of flooding.

Algorithm 3 *DoTRo*: The pseudo-code of the DoTRo algorithm

Input: x, wt_1, wt_2
Output: *leader*

```
 1: id = getId()
 2: x_min = x
 3: step = 1
 4: leader = true
 5: send(x, *)
 6: while (true) do
 7:     if (step == 1) then
 8:         rx = read(wt_1)
 9:         if (rx == null) then
10:             step = 2
11:             if (leader == true) then
12:                 send(x, *)
13:             end if
14:         else
15:             if (rx < x) then
16:                 leader = false
17:             end if
18:         end if
19:     end if
20:     if (step == 2) then
21:         rx = read(wt_2)
22:         if (rx == null) then
23:             stop()
24:         else
25:             if (rx < x_min) then
26:                 leader = false
27:                 x_min = rx
28:                 send(rx, *)
29:             end if
30:         end if
31:     end if
32: end while
```

5 CupCarbon simulator and SenScript

The simulation of networks is an essential tool for testing protocols and their prior-performance deployment. Indeed, such an establishment may be costly and challenging, especially when a large number of nodes are to be distributed at a large scale. This is why the simulation of networks is essential. CupCarbon is a Smart City and Internet of Things Wireless Sensor Network (SCI-WSN) simulator. Its objective is to design, visualize, debug and validate distributed algorithms for monitoring, tracking, collecting environmental data, etc., and to create environmental scenarios such as fires, gas, mobiles, and generally within educational and scientific projects. It can help to visually explain the basic concepts of sensor networks and how they work; it may also support scientists to test their wireless topologies, protocols, etc. (cf. Figure 2).

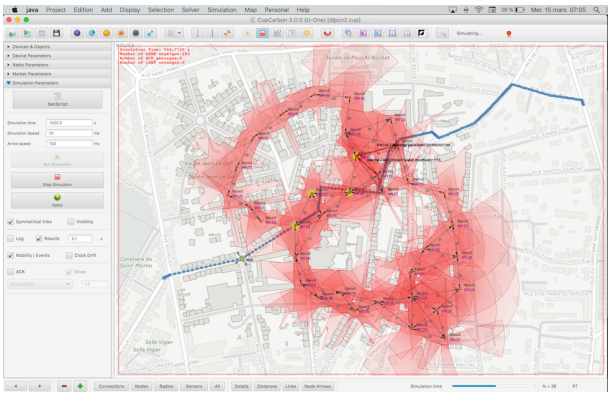


Fig. 2. CupCarbon User Interface.

Networks can be designed and prototyped by an ergonomic and easy to use interface using the OpenStreetMap (OSM) framework to deploy sensors directly on the map. It includes a script called SenScript, which allows to program and configure each sensor node individually. The energy consumption can be calculated and displayed as a function of the simulated time. This allows to clarify the structure, feasibility and realistic implementation of a network before its real deployment. CupCarbon offers the possibility to simulate algorithms and scenarios in several steps. For example, there could be a step for determining the nodes of interest, followed by a step related to the nature of the communication between these nodes to perform a given task such as the detection of an event, and finally, a step describing the nature of the routing to the base station in case that an event is detected [17, 18]. SenScript is the script used to program sensor nodes of the CupCarbon simulator. It is a script where variables are not declared, but can be initialized. For string variables, it is not necessary to use the quotes. A

variable is used by its name (e.g., x), and its value is determined by $ (e.g., $\$x$). Algorithm 4 shows an example of a SenScript code. The command **atget id cid** of line 1 allows to assign to variable *cid* the identifier of the current node. The command **loop** allows to start the loop section, where all the code situated after will be executed infinitely. The command textbfwait will allow to wait for a received message. This command is blocking and the next code will not be executed until a message is received. The command **read** of line 4 will assign the received message in the buffer to x. In the line 5, we test if the received message (an identifier) is less than the value of the current identifier cid. If is the case, the line 6 will be executed, and the node will be marked (**mark 1**), otherwise (line 7), the line 8 will be execute, where the current sensor will be unmarked (**mark 0**).

Algorithm 4 *SenScript example*

```
1: atget id cid
2: loop
3:    wait
4:    read x
5:    if($x < $cid)
6:       mark 1
7:    else
8:       mark 0
9:    end
```

6 Simulation results

To compare our algorithm with the classical *MinFind* algorithm, we have generated 9 networks in a rectangular area of $(z \times z)m^2$, where z is varied from 200 to 1000 with n randomly generated nodes. The value of n is fixed so that the density of the nodes in each network remains the same. We have fixed it to 10 $nodes/hm^2$ (hm: hectometer), i.e., 10 nodes in an area of $100m \times 100m$. Note, that we consider symmetric communications between nodes.

We have considered two cases. In the first case (case 1), each node generates a value representing its x-coordinate. In the second case (case 2), the considered value represents a random value. For each network, we have calculated the number of transmitted and received messages (exchanged messages) in order to compare their energy consumption which is directly related to this metric. We have obtained the graph of Figure 3. In both cases, we have executed the algorithm MinFind [15]. The obtained results are shown by the black curves of Figure 3 labeled as MinFind1 (case 1) and MinFind2 (case 2) and the red curve for DoTRo1 (case 1) and the blue curve for DoTRo2 (case 2). As we can see, the DoTRo algorithm is less energy consuming than the MinFind algorithm. This is confirmed by Figure 4 which shows the reduction rate between the MinFind

algorithms for both cases defined above. In the case where the leader represents the smallest x-coordinate value, we can see that the reduction rate reaches 83% for a network with 1000 nodes and it is growing for larger networks. This kind of leader is needed in the case of the D-LPCN algorithm [1] which starts from the node situated on the extreme left. In the case of other kinds of leaders (random, id, etc.) the proposed algorithm can reach a reduction rate of 30%. Altogether, we can conclude that the proposed algorithm is less energy consuming than the classical algorithm MinFind.

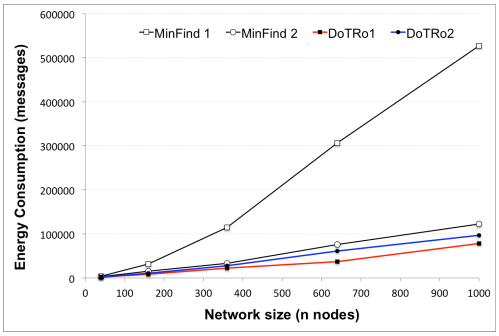


Fig. 3. Simulation results.

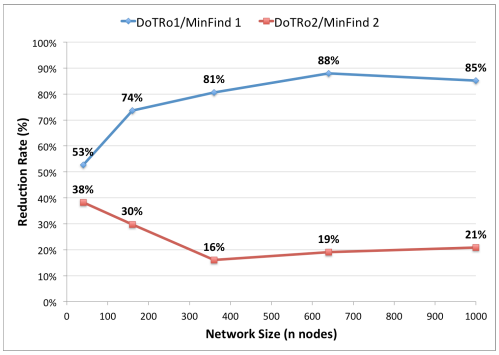


Fig. 4. Reduction rate.

7 Conclusion

We have presented a new algorithm called DoTRo (Dominating Tree Routing) which starts from the local minima found using the LMF algorithm. Then each local minimum starts as a root the process of flooding in order to determine the spanning tree while routing its value over it. If two spanning trees meet, the one routing the better value will continue the flooding process and the other one will stop it. After a given time, only one spanning tree will remain and its root will be the leader. The obtained results show that the proposed algorithm is less energy consuming with rates that can exceed 85% when searching the minimum x-coordinate and 30% when searching the minimum random value. Another advantage of the proposed algorithm is that it is fault tolerant since it starts even when there are failing nodes and it also finds the leader of each connected component of the network.

References

1. Saoudi, M., Lalem, F., Bounceur, A., Euler, R., Kechadi, M. T., Laouid, A., Madani, B., and Sevaux, M., D-LPCN: A Distributed Least Polar-angle Connected Node Algorithm for Finding the Boundary of a Wireless Sensor Network, Ad Hoc Networks, Elsevier, Volume 56, 1 March 2017, Pages 56-71.
2. M. Raynal. Distributed algorithms for message-passing systems, volume 500. Springer, 2013.
3. D. S. Hirschberg and J. B. Sinclair. Decentralized extrema finding in circular configuration of processors. Communications of the ACM, 23(11):627-628, 1980.
4. D. Dolev, M. Klawe, and M. Rodeh. An O(n log n) unidirectional distributed algorithm for extrema finding in a circle. Journal of Algorithms, 3(3):245-260. 1982.
5. R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. ACM Transactions on Programming Languages and systems (TOPLAS), 5(1):66-77, 1983.
6. D. Peleg. Time-optimal leader election in general networks. Journal of parallel and distributed computing, 8(1):96-99, 1990.
7. N. Malpani, J. L. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications, pages 96-103. ACM, 2000.
8. V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In the Proceedings of the Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution., volume 3, pages 1405-1413. 1997.
9. S. Vasudevan, B. DeCleene, N. Immerman, J. Kurose, and D. Towsley. Leader election algorithms for wireless ad hoc networks. In the Proceedings of the IEEE DARPA Information Survivability Conference and Exposition, 2003. Volume 1, pages 261-272, 2003.
10. S. Vasudevan, J. Kurose, and D. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In the Proceedings of the 12th IEEE International Conference on Network Protocols. ICNP 2004, pages 350-360. 2004.
11. A. Boukerche and K. Abrougui. An efficient leader election protocol for mobile networks. In Proceedings of the ACM international conference on Wireless communications and mobile computing, pages 1129-1134, 2006.

12. R. Ingram, P. Shields, J. E. Walter, and J. L. Welch. An asynchronous leader election algorithm for dynamic networks. In IEEE International Symposium on Parallel & Distributed Processing, IPDPS'09, pages 1-12, 2009.
13. M. Hammoudeh, Modelling clustering of sensor networks with synchronised hyperedge replacement, International Conference on Graph Transformation, Springer, Berlin, Heidelberg, 490-492, 2008.
14. M. Hammoudeh and T. Alsbou'i, Building programming abstractions for wireless sensor networks using watershed segmentation, Smart Spaces and Next Generation Wired/Wireless Networking, Springer, Berlin, Heidelberg, 587-597, 2011.
15. Santoro, N., Design and analysis of distributed algorithms, Vol. 56, John Wiley & Sons, 2007.
16. N. A. Lynch, Distributed algorithms, Morgan Kaufmann, 1996.
17. CupCarbon simulator, http://www.cupcarbon.com
18. Mehdi, K., Lounis, M., Bounceur, A., and Kechadi, T. CupCarbon: A Multi-Agent and Discrete Event Wireless Sensor Network Design and Simulation Tool, In IEEE 7th International Conference on Simulation Tools and Techniques (SIMUTools'14), Lisbon, Portugal, 2014.
19. Tanenbaum, Andrew S.; Wetherall, David J. (2010-03-23). Computer Networks (5th ed.). Pearson Education. p. 368-370.