Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 19 (2013) 330 – 339

Procedia Computer Science

The 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013)

SARP : Synchronous Adaptive Routing Protocol for MANETs

Mohamed Amine ABID[a], Abdelfettah BELGHITH[a], Khalil DRIRA[b]

[a]*HANA Research Group*
University of Manouba, Tunisia
[b]*LAAS-CNRS, France*
Univ. de Toulouse, France

Abstract

The aim of a routing protocol is to compute valid routes allowing every couple of nodes in the network to communicate at anytime. When the network topology is evolving over time, routing decisions should be constantly reconsidered. The main goal is to ensure a valid routing through time at the lowest possible cost. Conventional proactive routing protocols periodically recompute their routing tables; but due to their inherent nature based on shortest paths, they select longer links that ensure faster routing but are amenable to rapid breakages as nodes move around. Using short periods certainly allows a better tracking of the topology changes; however, it induces a higher control signaling overhead. An adequate trade-off between the routing period size and the traffic overhead should be found.

In this paper, we propose a new mechanism that keeps sensing the network mobility level to properly adjust the routing period size. It relies on a distributed algorithm that collects the network cartography which is then used to self-regulate the routing period size. Simulation results show that our proposed scheme correctly tracks changes and properly adjusts the current routing period size leading to much better performances.

Selection and peer-review under responsibility of Elhadi M. Shakshuki

Keywords: Autonomous routing, Adaptive routing, Mobility, Load

1. Introduction

So far, little consideration has been given to the impact of the used strategies and the applied topology updates frequency on the network performance. In [1] and [2], the authors focused rather on how to reduce the routing overhead. In fact, in [1], the author integrated the *Fisheye State Routing (FSR)* main idea [3] into OLSR in a way to reduce the routing overhead. While in [2], the authors proposed different updating strategies in order to maximize the routing period, and consequently reducing the signaling overhead. A generic investigation on the impact of certain topology updating strategies on the performance of proactive routing was conducted in [4, 5, 6].

However, none of these aforementioned research efforts and studies have paid enough attention to the routability and the reliability of the used proactive routing protocol, that is the correctness of the established routes within the routing tables, and its impact on network performance. In fact, the validity of computed routes plays a central and decisive role to improve the performance of the network; because forwarding through incorrect routes not only causes traffic wandering inside the network without ever being able to

Selection and peer-review under responsibility of Elhadi M. Shakshuki
doi:10.1016/j.procs.2013.06.046

reach their ultimate destinations, but also vainly over consumes valuable network resources leading to a tangible increase in the network perceived workload which causes the formation of congestion situations and conducts to poor performances. To reach the ultimate goal of better performance of our network, an adequate trade-off should then be found between the size of the routing period which controls the routing overhead and the routability of the used proactive routing protocol. The question naturally arises as to how to calibrate this trade-off between reliability, routing overhead and overall network performance. In this paper, we propose a new proactive routing protocol, called *SARP (Synchronous Adaptive Routing Protocol)*, that collects the cartography of the entire network. This cartography is then used to properly and dynamically tune the size of the routing period in an autonomous way, leading to a self-regulation of the routability of the protocol and hence attaining better performances.

The rest of the paper is organized as follows. In section II, we define the network cartography and how to measure its correctness. Section III is devoted to present a distributed mechanism to collect this network cartography. Section IV proposes an innovative approach based on the collected cartography that dynamically induces the dynamics of the network and accordingly adjusts the size of the current routing period so as to minimize the induced routing overhead and maximize the routing pertinence. The evaluation of this proposal is made in section V, where we compared it to the well known OLSR in terms of the network throughput. Section VI provides concluding remarks and some future investigations.

2. Network Cartography

2.1. The Cartography Definition

The network cartography is the set of every node's position within the network area. Our aim now is to design a distributed algorithm that allows each node to compute and build the cartography of the entire underlying network. Recall that link state based proactive routing protocols already collect the topology of the entire network, but not its cartography. The cartography is the geographic localization and connectivity of the different nodes throughout the network while the topology is restricted to the mere connectivity among the nodes. On the other hand, distance vector based proactive routing calculates the different routes without any knowledge or need of the entire topology. In this section, we develop a distributed algorithm to build the entire network cartography based on a distance vector proactive routing protocol.

To make nodes build their cartographies, we can rely on the assumption that each node is location capable (capable of knowing its own geographical location). Such an assumption can be easily justified by the recent availability of small and inexpensive low power GPS receiver devices [7]. Or, if these devices are not available or are not applicable (in door networks case for instance), we can rely on many other approaches based on relative coordinates. A large set of GPS less or restrained techniques for localization could be found in the literature [8, 9].

2.2. Definition of the Validity of the Cartography

The cartography provides the geographic position of each known node within the network at a given time. At any time, the correctness of the collected cartography is measured against the real actual cartography of the network. The validity of the cartography reflects how far it is from the reality. Note that the actual instantaneous cartography of the network can be extracted from the simulator but it cannot be known in practice. Since nodes are mobile they are continuously changing their positions. The correctness, hereafter named the validity, of the collected cartography falls in time until a new wave of routing updates is launched. It is then interesting to quantify the validity of the collected cartography as we get farther from the start of the routing period.

Consider a target node N. When N advertised itself, it was at position (x_0, y_0). In the cartography structure of a node A that had already heard N's Hello, a new entry for N was created, showing (x_0, y_0) as N's coordinates. Since node N is mobile, its position varies as a function of time, and consequently it will be at position (x_t, y_t) at time t during the same current routing period. We say that N's position, as indicated by A's current cartography (the entry relative to N), is valid as long as the distance between the recorded position (x_0, y_0) and the actual current position (x_t, y_t) is less than a tolerated predefined value

denoted by d. That is: $\sqrt{(x_t - x_0)^2 + (y_t - y_0)^2} \leq d$. The validity of the cartography, as perceived by any given node, represents the percentage of nodes having valid positions among all nodes. d is a tuning parameter whose value is relative to the transmission range used, and is in general a small fraction of this range.

3. Synchronous Cartography Collecting Protocol

Our proposal is based on a synchronous cartography collecting protocol. This proposal requires an adequate synchronization framework at the MAC layer similar to the one proposed in [10].

3.1. The Proposal Description

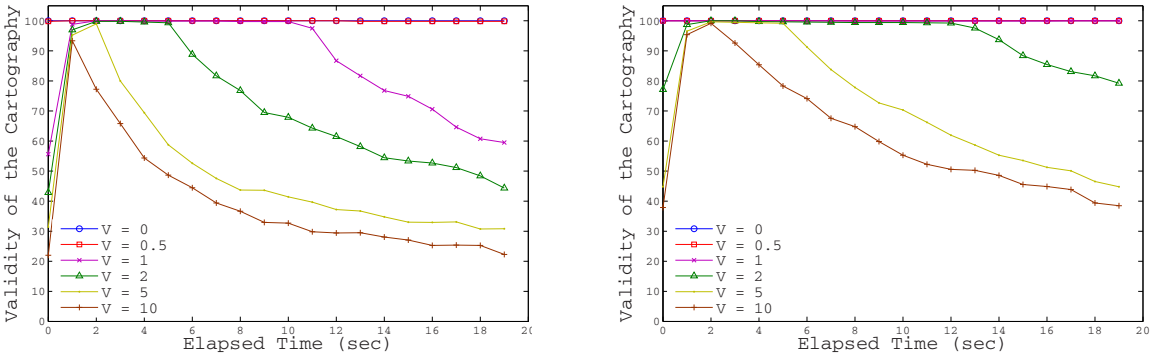
The cartography collection process can be integrated to any proactive routing protocol, especially distance vector routing protocols, with no additional signaling traffic apart from a very few additional fields in routing announcement messages (Hello). All what we need to make nodes collect the cartography, is just to disseminate nodes positions in the exchanged messages. At the start of each routing period, nodes will get a new cartography that reflects the real geographical distribution of nodes in the network (too close to reality). This collected structure could then be used for different purposes such as performing the routing function. Synchronization, however, is needed to make nodes announce themselves at almost the same time (within a small interval of time: 1 to 2 seconds). We intend to get homogeneous collected cartographies; i.e. the collected cartographies by all the nodes should have entries dating back almost to the same moment. Recall here that the cartography collection is spread over a small time interval depending on the network size, density and the applied data load. A generated and sent message Hello can take some time to reach every node in the network. As a consequence, a node can't get a coherent (homogeneous) cartography unless the latest Hello message generated and sent at a given routing period, reaches every node in the network. To speed up the collection process, Hellos are given transmission priority over any other awaiting data packets.

3.2. Simulation Set Up

To ascertain the validity of the cartography of the network as a function of the mobility, the traffic load and the elapsed time since the start of the current routing period, we conducted an extensive set of simulations. We have considered a simulation area of $1000m$ by $1000m$, with 120 mobile nodes using the Random Waypoint mobility model [11]. We used a transmission range and a carrier sense range both equal to $250m$. In all scenarios, we used a network capacity equals to $54Mbps$ and a maximum retransmission count equals to 3. We used a priority IP [12, 13, 14, 15] module at the network layer to enforce that Hellos are treated before any awaiting data packet. The simulation transient regime is evaluated to 100 seconds, and the routing updating period (generation of control traffic) is set to 20 seconds. One central node generates *CBR* (Constant Bit Rate) data streams; one stream to each one of the remaining 119 nodes at a rate of ρ packets per second per stream. The data packet size used is 200 bytes. Finally the *MaxDurationPeriod*, representing the life time of an entry in the routing table, is set to 30 seconds.

3.3. Simulation Results

1a and 1b represent the validity of the cartography collected by a random witness node, using a tolerance d of 10 meters 25 meters respectively. This validity is plotted as a function of the elapsed time since the start of a routing period and for different node speeds. We shall recall that the network load has a very little impact on the validity of the collected cartography independently of the considered speeds. This is mainly due to the use of our priority IP [12, 13, 14, 15] handling scheme that sends Hello messages in priority avoiding all substantial waits at sending queues that could eventually be caused by data packets. As shown these two figures (Fig.1a and Fig.1b), with a static network (no mobility), we get a validity of 100% during the whole period. We can see that the validity of the cartography gets at its maximum around the instant $2sec$ taking the start of the period as a time origin, which is the time required to get the maximum of Hellos throughout the network. More interestingly, these figures show that once the validity of the cartography reaches its ultimate maximum value, it stays there for a while before decreasing as time gets farther from the start of the routing period. The duration of this stay, however, depends on the node mobility. For example, figure 1a



(a) ***tolerance = 10m*** **(b)** ***tolerance = 25m***

Fig. 1: Validity of the Cartography as a function of the elapsed time for a null load

portrays a stay of 10 seconds at the maximum validity level when nodes speed is $1m/s$; while the speed of $10m/s$, a rather quick stay at a maximum validity of ninety four percent. It is of course obvious that when we use a larger tolerance value, the stay at the maximum validity level for each applied speed will be greater than when using the $10m$ tolerance. In fact, by using a larger tolerance value, $25m$ for instance, we give our cartography more resiliency to the network dynamics as it is shown in figure 1b. We will always get the same general behavior, but a better resistance to the nodes mobility.

Since it shows a high level of validity, especially for low and moderate speeds, this collected cartography can simply be used as a link state protocol does: compute shortest paths (in terms of a given metric) towards all nodes within the network, and just rely on these decisions to route data packets. However, it can be used otherwise. Recall here that as the elapsed time gets farther from the start of a routing period, the collected cartography gradually recedes from reality (it loses its validity). It would then be interesting if one can dynamically measure or estimate the point at which this validity starts decreasing below a certain predefined threshold (i.e. a tolerated remoteness from reality). Such a point provides the size of the routing period to use if we want to drive the network with valid routes or at least with the best possible percentage of valid routes above the fixed threshold. Recall that non valid routes cannot deliver any traffic, yet they deteriorate the network performances. This is investigated next.

4. Routing Period Dynamic Self Regulation Protocol

As it was shown in the previous section, the evolution of the validity of the collected network cartography as time progresses since the start of the current routing period goes through three identifiable phases: one first phase of nodes position gathering starting at instant 0 of the routing period. During this phase, the validity of the collected cartography incrementally increases until reaching the maximum validity level (around $2sec$ later) and announcing by the same way the end of this first phase and the start of the second one. The second phase, called the stay phase, is the one where the validity of the collected cartography is maintained at this maximum level. It ends at the start of the third and last phase where the validity starts decreasing as time progresses until the end of the current routing period. For a fixed tolerance value, the durations of these phases are closely related to nodes speeds. Particularly, using a tolerance of $10m$, the stay phase varies from a long stay of $10sec$ at a speed of $1m/s$, to a very quick stay at speed of $10m/s$. With a tolerance d equals to 25 meters (figure 1b), the same evolution is detected but with a better resilience as the tolerance is now larger. The stay phase is then larger compared to the $10m$ tolerance case.

In the other hand, the validity of the proactive routing is intimately related to the validity of the network cartography. It is then important to maintain the collected cartography valid, or equivalently a high routability, beyond a certain predefined level since invalid routes not only will not succeed in delivering their traffic, but more drastically over consume vainly valuable network resources as each packet is retransmitted

up to the retransmission count limit, defined by the underlined MAC protocol, before being rejected. To this end, we propose to dynamically sense the point at which the validity starts decreasing below a predefined threshold. Such a point in time depends on the used nodes' speed; and will provide the adequate size of the routing period to be used if we want to drive the network with the best possible valid routes above this fixed threshold.

A logical question that arises at this level is, how to measure the validity of the cartography in practice knowing that we ignore the real cartography of the network evolving during the period. In the aforementioned discussion about the validity of the cartography, we assumed the presence of an oracle providing us with the actual nodes positions at any instant (the actual network cartography). To circumvent this problem, we propose that at the start of each routing period (i.e. at instant 0 of each period) and just before triggering the new wave of Hello messages, a node saves its perceived cartography, denoted by C_0, which represents the maximum of collected cartography at the previous period. Based on the previous simulation results, we observed that the collecting process ends at instant 2 seconds from the beginning of the current routing period. As such, 2 seconds after saving C_0, this node finishes collecting its new perceived cartography, denoted by C_2. This later cartography, at this very instant of 2 seconds, may adequately represent the real unknown network cartography we are seeking for. Consequently, by comparing C_0 to C_2, the node gets the validity of C_0 and can therefore increase, decrease or keep the same size of its current routing period.

Let us consider the i^{th} routing period and a node A having a cartography C_0 at instant $t = 0sec$ (at the start of the current period), and a new collected cartography C_2 at instant $t = 2sec$ in this same period. Let N be an advertised node saved at position (x_0, y_0) in C_0 and position (x_2, y_2) in C_2. Node A considers that the position of node N as indicated in its C_0 as valid if the distance between (x_0, y_0) and (x_2, y_2) is less than or equal to the assumed tolerance value d; that is if $\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2} \leq d$.

The validity of our cartography C_0 at the routing period i, denoted by $V_i(C_0)$, is then the percentage of nodes having valid positions among all nodes. Depending on the value of $V_i(C_0)$, node A will adjust in a autonomous way, its current routing period size $T(i)$ relatively to the previous routing period size $T(i-1)$. The tuning decisions are made such that if $V_i(C_0) \geq P_1$, then $T(i)$ can be set larger than $T(i-1)$. The second case is when $V_i(C_0) \leq P_2$, then $T(i)$ should be set smaller than $T(i-1)$. $T(i)$ can be kept same as $T(i-1)$ if $P_2 \leq V_i(C_0) \leq P_1$. Where P_1 and P_2 are two validity percentages ($P_2 \leq P_1$). Let T_{min} and T_{max} denote respectively the minimum and maximum size of the routing period, that is : $\forall i,\ T_{min} \leq T(i) \leq T_{max}$

In this work, we deliberately choose ($P_1 = 95$, $P_2 = 80$), $T_{min} = 3s$ and $T_{max} = 20s$. The dynamic regulation of the routing period size is then governed by the followings rules: (I) if $V_i(C_0) \geq P_1$ then $T(i) = T(i-1) + 1$, (II) if $V_i(C_0) \leq P_2$ then $T(i) = T(i-1) - 1$ and (III) if $P_2 \leq V_i(C_0) \leq P_1$ then $T(i) = T(i-1)$.

Recall that our proposed protocol relays on a synchronous collected cartography. Such a requirement could be met by the use of an adequate synchronization framework at the MAC layer similar to the one proposed in [10]. He have considered the routing update period number as the sequence number to use. This suppose that upon entering to the network, a node should know the current period number, its time origin and its duration. These three quantities could be easily provided by the routing layer of the station emitting the Beacon and from which we got the association at the 802.11 MAC Layer. Note also that the above regulation rules implicitly assume that the maximum of cartography is collected within 2 seconds from the start of the current routing period as portrayed on all previous figures. This amounts to say that all nodes starts a new wave of Hello messages at the same exact time.

As the regulation decision is made locally by each node a part, the questions that naturally arise are: First, how to maintain the same period size for all nodes? And second, how to force these periods to start at the same instant? Otherwise, the time origin of a given routing period may differ from one node to another and consequently Hello messages will be spread over the time axis (even the condition of the cartography collection within $2sec$ will no longer be respected).

This is indeed accomplished by Algorithm 1, under the tacit hypothesis that $(P_1 - P_2)$ is large enough to prevent the case where one node increments its current routing period while another node decrements its own. This hypothesis is usually met since Hello messages are treated in priority leading to almost the same

cartography of the network, built by every node. This algorithm firstly determines both the current period size $T(i)$ and the start instant of the next routing period (the time origin of $T(i+1)$). It also enforces the sizes of the i^{th} routing periods at all nodes to be of at most one second difference. Moreover, for any couple of nodes, only a shift of at most one second on the time origins of their i^{th} routing periods is permitted.

For now on, and since the size of the i^{th} routing period as well as its time origin may both differ by one second from one node to another, the validity of the cartography should be calculated by comparing the old cartography of instant one second before the start of the current routing period denoted by (C_{-1}) against that of instant 3 seconds after the start of the current routing period denoted by (C_3). This is by the way the very reason for which we set the value of T_{min} to 3 seconds. Let $V_i(C_{-1})$ denote the validity of the old cartography (C_{-1}) that will be compared against $(C_{T_{min}})$ of routing period i. Algorithm 1 is then provided with $V_i(C_{-1})$ instead of $V_i(C_0)$.

Algorithm 1 CalculateCurrentPeriodSize($Tmin, Tmax, P1, P2, Vi(C-1)$)

1: *The Algo. determines the current period size *newPeriod* and schedules the next Hello updating Process (i.e. the time origin of the next period)*
2: *CurrentUpdateInstant* ← *updateInstant*
3: *lastPeriod* ← *newPeriod*
4: **if** *Shift* = **false then** // No time origin shift is detected and consequently we adjust according to *Vi* (*C*0)
5: *freezingPeriods* ← 0
6: **if** *Vi* (*C*−1) < *P*2 **then** // decrement the routing period
7: *newPeriod* ← max (*lastPeriod* − 1, *Tmin*)
8: *Up* ← **false**
9: **else if** *Vi* (*C*−1) ≥ *P*1 **then** // increment the routing period
10: *newPeriod* ← min (*lastPeriod* + 1, *Tmax*)
11: *Up* ← **true**
12: **else**
13: *newPeriod* ← *lastPeriod*
14: *Up* ← **false**
15: **end if**
16: *updateInstant* ← *updateInstant* + *newPeriod*
17: **else** // a time origin shift is detected
18: *Shift* ← **false**
19: **if** *Up* = **true then**
20: **if** *Vi* (*C*−1) ≥ *P*1 and *freezingPeriods* < 1 **then**
21: *newPeriod* ← *lastPeriod*// keep same period size
22: *updateInstant*← *updateInstant* + *newPeriod* − 1// keep Shift=1
23: *freezingPeriods* ++
24: **else** // decrement both period size and updateInstant
25: *Up* ← **false**
26: *freezingPeriods* ← 0
27: *newPeriod* ← max (*lastPeriod* − 1, *Tmin*)
28: *updateInstant*← *updateInstant* + *newPeriod* − 1
29: **end if**
30: **else** // Up = **false**
31: *newPeriod* ← max (*lastPeriod* − 1, *Tmin*)// decrement the period size twice
32: **if** *newPeriod* = *Tmin* **then** // limiting case: recuperate the shift within two periods
33: *updateInstant*← *updateInstant* + *newPeriod* + *Tmin* − 1
34: **else** // recuperate the shift upon the next period
35: *updateInstant*← *updateInstant* + *newPeriod* − 1
36: **end if**
37: **end if**
38: **end if**
39:
40: **if** *updateInstant* > *CurrentUpdateInstant* + *Tmin* + 1 **then**
41: Schedule saving the cartography at *updateInstant*-1
42: **else**
43: Save now the current cartography
44: **end if**
45: // schedule the new updating process trigger time:
46: **return** max (*Now*, *updateInstant*)

This algorithm, firstly presented in [16], is executed by each node in the network T_{min} from the start of the current period, after finishing the collection of C_3. It computes the current period size (Notice this period has already started $T_{min} = 3$ seconds ago) as well as the time origin of the next period (i.e. the remaining time in the current period which is also the instant when the new updating wave is triggered). The meaning of the different presented variables used in Algorithm 1 is as follows:

- *UpdateInstant*: when the algorithm is executed by a node, *UpdateInstant* firstly represents the time origin of the current routing period (already computed at the last execution of the algorithm during the previous period). The node starts by saving its value in the local variable *CurrentUpdateInstant*. Upon the termination of the algorithm execution, *UpdateInstant* contains the time origin of the next routing period.
- *newPeriod*: at the start of the algorithm, *newPeriod* contains the size of the last routing period (the previous one). *newPeriod* is then saved in the variable *LastPeriod*. Upon the termination of the algorithm execution, *newPeriod* contains the size to be adopted for the current routing period.

- $Shift$: at the start of the algorithm, $Shift$ can be either set to $True$ or $False$ depending on whether a shift in time origins has been detected or not. A node detects a shift if receives a Hello message originated within one second before the start of its current routing period. Note that no Hello messages can be generated at any node, more than one second before the start of the routing period of any other given node.
- $freezingPeriods$: a node cannot increment its current period for more than once if another node has kept its own current period unchanged. That is if a shift in the time origin is detected ($Shift = True$), then this node increments its current routing period only if $V_i(C_{-1}) \geq P_1$ and it has not done so during the last period. Otherwise it decrements both its current routing period size and its $UpdateInstant$ to enforce the one second difference at most.
- UP: it take the value $True$ or $False$ depending on whether the period size is incremented or decremented respectively.

Before going any further, some deeper explanations might be necessary to better understand how does our algorithm work. Through figures (2.a), (2.b) and 3, we better explain how under our assumption, saying that we can't have a situation where a node decides to decrement its routing period size while another node decides to increment its own, we do guarantee that the size of the i^{th} routing period as well as its time origin can only differ by at most one second for any considered couple of nodes. Since the period size can only change by at most one second from one period to its successor, and knowing that all nodes firstly started using the same period size, we can assume that a difference of two seconds or more in the routing period sizes needs at least two consecutive periods to cumulate shifts in time origins. In other words, such a situation supposes that at least one node in the network detects that a shift is created. This particular node, and once it executes Algorithm 1, will go to the part pointed out by line 17 of our algorithm. This particular situation can be reached in two cases: the first one is when this node decides to keep its routing period the same as it was in the previous one, while other nodes choose to decrement it. The second one is when this node increments it routing period size while other nodes keep their own unchanged.

An example of the first situation is illustrated by figure (2.a). If the node detecting the shift once created, do not tries to recuperate this origin time lag, we can go in a situation of two or more seconds of difference in routing period sizes as it is illustrated in figure (2.a). The node treatment corresponds to line 30 It firstly decrements its routing period size (since we detected a shift in time origins even if we have kept our routing period unchanged, this means that there are some nodes that have decremented their own periods), then tries to recuperate this shift in the next period by scheduling the next updating wave one second before the end of the current routing period (i.e. at $updateInstant + newPeriod - 1$). Otherwise, and if we only decrement our routing period size, the difference of at most one second between nodes periods sizes and time origins can no longer be guaranteed (the situation illustrated by figure (2.a) for instance). Notice that if we are already using a period size equal to T_{min} seconds, the shift can only be recovered two periods later (we can't schedule the updating 1 second before the end of the new computed period which has already ended).

An example of the second situation is presented by figure (2.b). As already mentioned, a node gets into this situation when it decides to increment its routing period while other nodes just kept their own ones unchanged. At the start of the next period, this particular node detects the shift formation. In our algorithm, we distinguish two different cases: if this node, and after measuring the validity of its collected cartography, finds that it is $< P_2$ (in usual situations, it means, whether keep the same period or decrement it), then it just decrements its period size and recuperate the shift. Such a decision can be argued by the fact that the decision of incrementing its previous period was neither confirmed at this period nor followed by the other nodes in the network (we still have a shift). The second situation is when the measured validity is $\geq P_1$ which represents a confirmation of the previous made decision at the previous period. The very first detected shift at the previous period means that some nodes in network didn't yet noticed the betterment of the validity measure, and will probably follow in this period. That's why the node just freezes its routing period size (keep it the same), avoiding by the way a hysteresis phenomena of the routing period size; but still needs to recuperate the shift to avoid situations where more than one second difference occurs (the exact situation presented by figure (2.b): even if at period $i + 1$, node x decided to keep its routing period size unchanged, a $2sec$ shift was created). The consequences of such a shift could be worse as the illustrative case shown in figure . Figure 3 illustrates how the node succeeds in keeping the shift equal to at most $1sec$ by behaving the way indicated by Algorithm 1.

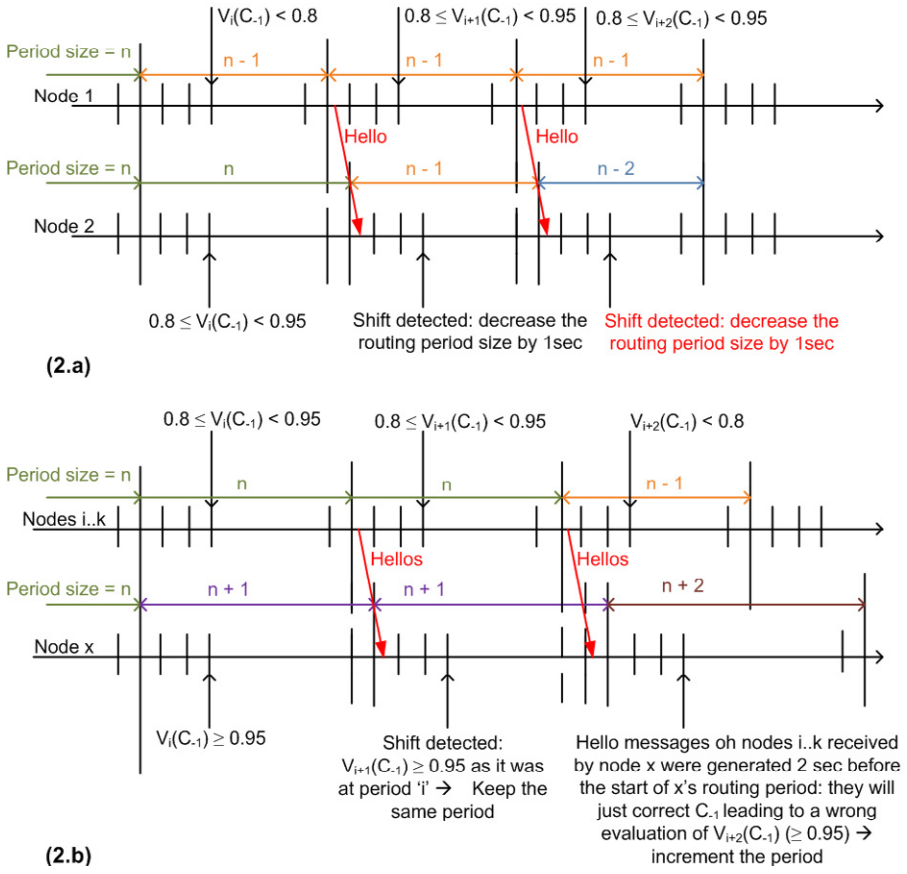


Fig. 2: Possible dysfunction results of a non recovered detected Shift: (2.a): first illustrative case, (2.b): second illustrative case

5. performance evaluation: SARP vs OLSR

To bring out the effectiveness of our proposal, we run a simulation for 600 seconds where we modify the nodes speed each 120 seconds as follows: The first $120sec$ start with a stationary network for $25sec$ and then switches to $3m/sec$ for $95sec$. This first period is considered as a transient regime and consequently only performances emanating from the last 480 seconds are taken into account. For the second $120sec$ period, nodes continue moving at the same speed of $3m/sec$; then at $6m/sec$ during the next $120sec$, then $2m/sec$ and finally $0m/sec$. We considered a simulation area of $1000m \times 1000m$ where we deployed 100 mobile nodes using the Random Way Point mobility model [11]. We used a transmission range of $250m$, a tolerance d of 25 meters, a network capacity of $54Mbps$, a maximum MAC retransmission count equals to 3 and a $MaxDurationPeriod$, representing the life time of an entry in the routing table, set to 30 seconds. All required modifications are ported on the OMNET++ network simulator where we used a priority IP [12] at the network layer to enforce that Hellos are treated before any waiting data packet. We immobilized 10 source nodes at the left edge of the simulation surface (randomly distributed over the three sub-areas at the left edge) and 10 destination nodes at the right edge. we also consider ten traffic flows using these already fixed source-destination pairs (pairs are chosen randomly). As such, we assured multi-hop routes (4 to 5 hops). The data packet size used is 200 bytes.

For our comparison, we will limit our attention solely to the network throughput which is considered as one of the major performance metrics. We will compare the performances assured by our proposal to that of the very known OLSR [1]. We deployed it in the same prioritization IP model. The TC period is set to 8

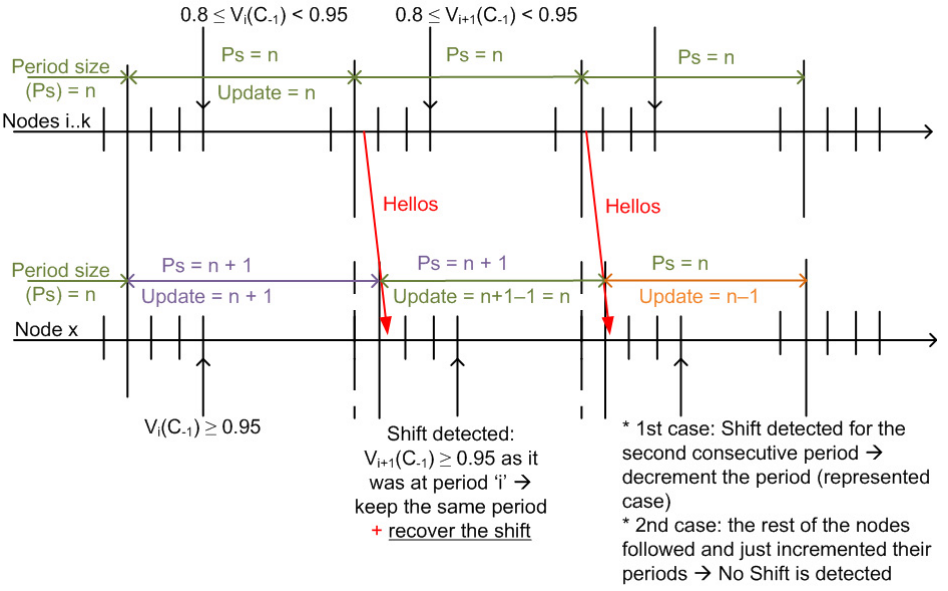


Fig. 3: How does the Shift recovery resolve the second illustrative case dysfunction

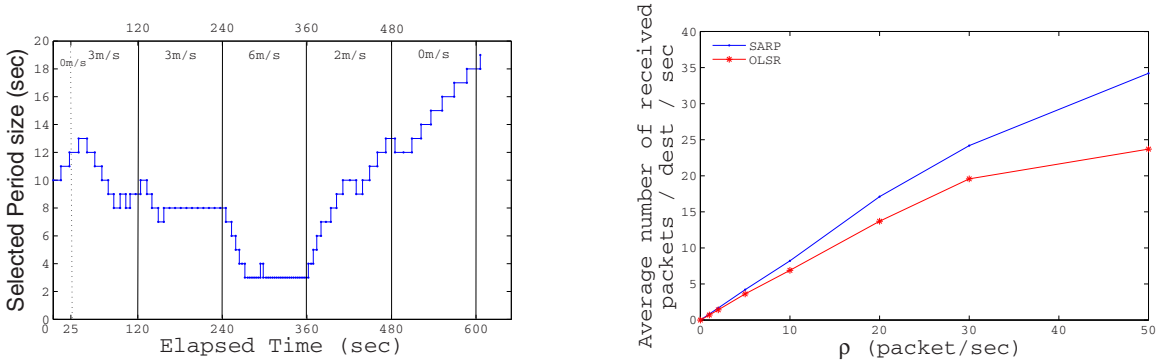
seconds and the Hello one to 2 seconds. For our proposal, we fixed the maximum updating period size T_{max} to $20sec$, and the minimum, T_{min}, to 3 seconds.

Fig. 4a clearly portrays that the proposed approach senses correctly the change in nodes' speeds and dynamically adjusts the routing period size accordingly. For the first 120 seconds sub-period, the period size started increasing because nodes were kept stationary during the first $25sec$ of our simulation. Since the period size is firstly initiated to $Tmax/2$ seconds; that is $10sec$; the node starts incrementally increasing its routing period size until the nodes start moving. At a speed of $3m/sec$ an adequate routing period size, equal to $8s$ or 9 is constantly used. At the start of the third sub-period, the speed is increased to $6m/sec$. Here, we observe how our algorithm started decreasing its period size until reaching the correct routing period size of $3seconds$ (Notice here that we forced all the nodes to move using the exact indicated speed; as such, at a speed of $6m/s$ for instance, the tolerated value of $25m$ will probably be exceeded after 3 seconds). As indicated by Fig. 4a, this size is automatically reached after 4 to 5 routing periods (after around 40 seconds). At the start of the fourth sub-period, we decreased the speed to $2m/sec$. The routing period gets back slowly to the appropriate value of 12 seconds. For the last sub-period, we changed the speed from $2m/sec$ to $0m/sec$ and we note that our self regulating approach thrived appropriately to increase the routing period size to its ultimate value (namely $T_{max} = 20s$), though slowly since we only add one second at each step.

As we already noticed, we will limit our attention to the network throughput only. Figure 4b portrays the network throughput, defined here as the average number of correctly received packets per flow per second, as a function of the traffic data load per flow and for both OLSR and the adaptive protocol SARP. We clearly observe the superiority of our proposal and this for the entire range of traffic loads. At moderate to high workloads, OLSR protocol is unable to cope with the loss of routing validity as it uses a TC equal to 8 seconds fixed, in time, and independent from the nodes mobility.

6. Conclusions

We proposed a distributed algorithm that collects the network cartography; then, through extensive simulations, we evaluated its validity as a function of the time since the start of a routing period, the network load and the network mobility. This cartography is then used to appropriately and dynamically adjust the current



(a) ***Routing period dynamic self regulation*** **(b)** ***Throughput: SARP vs OLSR***

Fig. 4: Performance evaluation of SARP

routing period size in a way to reduce the control traffic and maximize the routing pertinence. Simulations showed that our scheme is indeed capable of properly tracking the network dynamics and accordingly adjusting the current routing period size. We ended the paper by a brief comparison of our proposal to the well known OLSR, showed that using SARP yields to a better performances of our network. Further refinements of our proposed schemes are being investigated to speed up the dynamic local self adjustment process of the current routing period size.

References

[1] T. C. A. L. A. Q. Philippe Jacquet, Paul Muhlethaler, L. Viennot, Optimized link state routing protocol for ad hoc networks, Proceedings of the IEEE International Multitopic Conference (INMIC 2001) (2001) 1374 – 1378.
[2] P. Samar, Z. Haas, Strategies for broadcasting updates by proactive routing protocols in mobile ad hoc networks, IEEE MILCOM 2002, Anaheim, CA, USA.
[3] M. G. G. Pei, T. Chen, Fisheye state routing in mobile ad hoc networks, in: ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications.
[4] S. B. Y. Huang, S. Sorensen, Analysing the impact of topology update strategies on the performance of a proactive manet routing protocol, 27th International Conference on Distributed Computing Systems Workshops, ICDCSW'07, Toronto, Ontario, Canada.
[5] I. Jemili, A. Belghith, M. Mosbah, A synchronous tiered based clustering algorithm for large-scale ad hoc networks, in: The 10th IFIP International Conference on Mobile and Wireless Communications Networks (MWCN08), 2008.
[6] A. Belghith, I. Jemili, M. Mosbah, A distributed clustering algorithm without an explicit neighborhood knowledge, International Journal of Computing and Information Sciences (IJCIS) 5 (1) (207) 24–34.
[7] S. Giordano, I. Stojmenovic, Position-based ad hoc routes in ad hoc networks, 2003, pp. 287–300.
[8] A. C.Saad, Jean-Claude, At-dist: A distributed method for localization with high accuracy in sensor networks, International journal Studia Informatica Universalis, Special Issue on Wireless Ad Hoc and Sensor Networks 6, 1 (2008).
[9] E. N. A. Boukerche, H.A.B. de Oliveira, A. Loureiro, Towards an integrated solution for node localization and data routing in sensor networks, in: Proceedings 12th IEEE Symp. Computers and Comm. (ISCC '07), July 2007.
[10] W. A. Abdelfettah Belghith, J. M. Bonnin, Traffic aware power saving protocol in multi-hop mobile ad-hoc networks, The Journal of Networks 2 (4), academy Publisher.
[11] T. Camp, J. Boleng, V. Davies, A survey of mobility models for ad hoc network research, Wireless Communications & Mobile Computing (WCMC): Special Issue on Mobile Ad Hoc Networking: Research, Trends and Applications 2 (2002) 483–502.
[12] M. A. Abid, A. Belghith, Modeling efficiency of mobile ad hoc networks in omnet++The 4th Joint IFIP/IEEE Wireless and Mobile Networking Conference, Toulouse, France.
[13] M. A. Abid, A. Belghith, Stability routing with constrained path length for improved routability in dynamic manets, The International Journal of Personnal and Ubiquitous Computing, IJPUC Springer 15 (8).
[14] M. A. Abid, A. Belghith, Period size self tuning to enhance routing in manets, International Journal of Business Data Communications and Networking (IJBDCN) 6 (4) (2010) 21–37.
[15] I. Lassoued, J. Bonnin, A. Belghith, Towards an architecture for mobility management and ressource contro, in: The IEEE Wireless Communications and Networking Conference WCNC 2008, IEEE Computer Society, Las Vegas, Nevada, USA, 2008.
[16] A. Belghith, M. A. Abid, Dynamically self adjustable proactive routing protocols for mobile ad hoc networks, in: IEEE 34th Conference on Local Computer Networks, 2009. LCN 2009., 2009, pp. 506 – 513. doi:10.1109/LCN.2009.5355176.