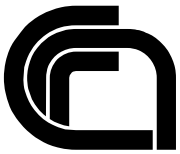


Consiglio Nazionale delle Ricerche

Detecting users' communities in mobile social networks

E. Borgia, M. Conti, A. Passarella

IIT TR-19/2011

Technical report

settembre 2011

Istituto di Informatica e Telematica

Detecting users' communities in mobile social networks

Eleonora Borgia, Marco Conti, Andrea Passarella
Institute of Informatics and Telematics (IIT)- CNR
via G. Moruzzi,1 - 56124 Pisa, Italy
email: {*e.borgia, m.conti, a.passarella*}*@iit.cnr.it*

Abstract

In this paper we focus on approaches which aim at discovering communities of people in Opportunistic Networks. We first study the behaviour of three community detection distributed algorithms proposed in literature [1], in a scenario where people move according to a mobility model which well reproduces the nature of human contacts, namely HCMM [2]. By a simulation analysis, we show that these distributed approaches can satisfactory detect the communities formed by people only when they do not significantly change over time. Otherwise, as they maintain memory of all encountered nodes forever, these algorithms fail to capture dynamic evolutions of the social communities users are part of. To this aim we propose AD-SIMPLE, a new solution which captures the dynamic evolution of social communities. By an extensive simulation analysis, we demonstrate that it accurately detects communities and social changes while keeping computation and storage requirements low.

I. Introduction

Opportunistic Networks [3] are self-organising wireless networks formed by mobile devices carried by people. Due to people mobility, the network is disconnected most of the time and the existence of a complete path between pairs of senders/destinations cannot be assumed. However, communications among nodes are still possible since local interactions are exploited to deliver messages in a hop-by-hop fashion. This can be achieved by means of *i)* dissemination-based routing protocols, which essentially implement form of controlled flooding [4], or *ii)* context-based routing protocols, which aim at maximising the probability of message delivery searching for the most appropriate relay node. The latter category exploits information about the context which are used to construct the so-called *utility* of a node, i.e., the usefulness of a node to be the best next-hop for a message. To this end, several schemes have been proposed taking into account the history of encounters [5] [6], or the mobility [7] or a combination of attributes [8].

Among the various classes of context-based routing protocols, those exploiting information about social relationship between the users are considered very promising. Humans are social individuals that have social ties and form communities to better answer to their needs. People belonging to the same community meet with high probability and regularly. On the contrary, people from different communities meet less frequently. Therefore, understanding the human structure and the rules which regulate social interactions and aggregations can be a great advantage. An accurate knowledge of local community can be exploited to increase the performance of forwarding schemes for Opportunistic Networks, as shown for example

in HiBOp [8] and BubbleRAP [9]. As users belonging to the same community share common interests, social information can be efficiently used to design smart strategies to choose the most suitable next hop.

Complex network analysis has been recently proposed as a efficient way to detect the structure of a network, and indices such as betweenness, similarity and centrality are defined as a compact representation of the properties of the nodes. The approaches presented in [10] [11] [12] show to reliably extract the community structure in real world networks by deriving statistics of communities from traces. They rely on centralised schemes which require the complete knowledge of the entire structure of the graph. Conversely, the approach proposed in [13] infers the community structure, named *local modularity*, without relying on some centralised points by exploring each vertex at a time. By exploiting such results, in [1] authors proposed the distributed version of three centralised community detection algorithms (i.e., SIMPLE, k-CLIQUE and MODULARITY) and show their great potential and their accuracy in identifying social community wrt the centralised versions.

In this paper, we focus on approaches aimed at discovering dynamic communities of people in an opportunistic scenario by starting from the three distributed algorithms proposed in [1]. We can summarise our major contributions as follows:

i) We compare and contrast them in different scenarios in terms of the achieved *similarity* metric, i.e., how similar the detected communities are with respect to those detected by the centralised algorithms. We differentiate from [1] in the type of traces which are used for running the analysis. We use synthetic traces generated from a mobility model which well reproduces human movement patterns observed in real traces, namely HCMM [2]. This allows us also to create different scenarios wrt those in [1] and thus to make a deeper analysis. Obtained results show that SIMPLE performs better with respect to k-CLIQUE and MODULARITY.

ii) We also demonstrate that their accuracy in identifying communities decreases with the increase of the complexity of the scenario. They are not able to correctly represent the dynamic users' social behaviour in scenarios with nodes moving across different communities.

iii) We finally propose a new community detection algorithm, namely Adaptive Detection SIMPLE (AD-SIMPLE), which is able to capture the evolution of social communities in dynamic scenarios, while keeping computation and storage requirements low. To the best of our knowledge this is the first attempt proposed in literature that explicitly includes *adaptive* community detection mechanisms.

The paper is organised as follows. Section II states the problem to be faced, introducing the community detection algorithms defined in [1] and the mobility model assumptions. Section III presents the simulation analysis to compare the distributed algorithms proposed in [1], and discusses the obtained results. Section IV introduces the proposed Adaptive Detection SIMPLE algorithm, while in Section V we provide an extensive evaluation of the proposed solution by means of simulation. Specifically, first, we show the sensitiveness of AD-SIMPLE on specific parameters, then we present results of a comparison of AD-SIMPLE against that of the algorithms in [1] in case of dynamic social communities. Finally, Section VI concludes the paper.

TABLE I
NOTATION

$F_0(F_i)$:	Familiar Set of node $v_0(v_i)$
$C_0(C_i)$:	Local Community of node $v_0(v_i)$
$\tilde{F}_j$:	local approximation of the Familiar Set of node $v_j \in C_0$
$FSoLC_0$:	local approximation of the Familiar Sets of all vertices in C_0 ($FSoLC_0 = \{\tilde{F}_j \mid v_j \in C_0\}$)

II. Distributed Communication Detection algorithms

In this section we provide the background information necessary to understand the three distributed algorithms which are used in the simulation analysis to detect social communities. We first provide some basic definitions and then we report the main characteristics of each algorithm. Interested readers can refer to [1] for additional details.

A. Definitions

The following definitions are common to all the three algorithms:

- **Familiar Set (F):** a familiar set of a node v_0 is composed by all those encountered nodes for which the cumulative contact duration[1] exceeds a predefined threshold t_{th}.
- **Local Community (C):** a local community of a node v_0 is composed by all the nodes in its familiar set and the nodes for which criteria, specific to each algorithm, are valid.

Table I summarises the notation used hereafter.

The idea at the basis of all the three community detection algorithms is that each node determines the composition of its community based on the contacts it has with other nodes. This is achieved by storing specific information. Specifically, when two nodes meet each other, they first exchange local information and then take independent decisions about including or not the encountered node into the Familiar Set (i.e., *threshold criteria*) [2], or only into the Local Community (i.e., *admission criteria*). In addition, each node evaluates if its Local Community can be merged (partially or completely) with the Local Community of the encountered node (i.e., *merging criteria*).

The threshold criteria are common to all the the three algorithms.

Threshold Criteria: when two nodes meet, each node evaluates the cumulative contact duration t_{cum}. If $t_{cum} \geq t_{th}$, then the encountered node is added to the Familiar Set (and consequently to the Local Community).

On the contrary, the admission criteria and merging criteria are specific for each algorithm and they are carefully described in the next subsections.

[1]The cumulative contact duration for a pair of nodes (t_{cum}) is the sum of the duration of all the different contacts.

[2]When a node is added to the Familiar Set it is automatically added to the Local Community

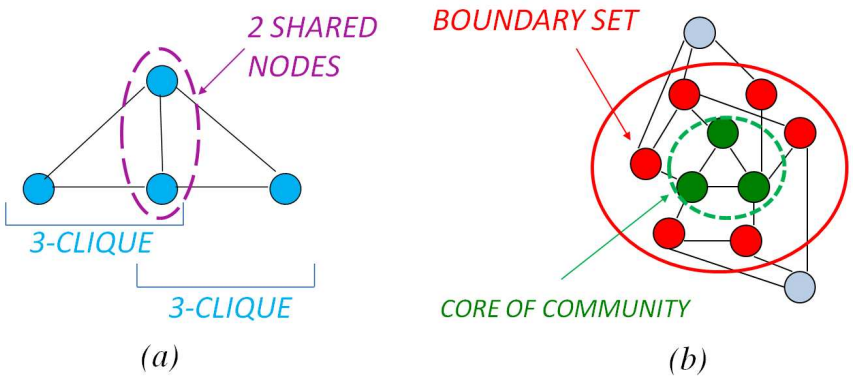


Fig. 1. (a) Example of k-CLIQUE with k=3 and (b) example of Boundary Set for the Modularity algorithm.

B. SIMPLE community detection algorithm

As suggested by the name, in SIMPLE nodes exchange very few information. Suppose that two nodes, e.g., v_0 and v_i, make contact. During the contact duration, each node sends to the other node its Familiar Set and its Local Community. At the end of the contact, each node evaluates the following two criteria to update the relative data structures. Focusing on v_0:

B1. **Admission Criteria:** in case of failure of the threshold criteria, if the number of nodes shared among C_0 and F_i is higher than λ times the number of nodes in F_i, then the encountered node v_i is added to the Local Community of v_0 (i.e., $if|C_0 \cap F_i| > \lambda \cdot |F_i| \Longrightarrow C_0 = C_0 \cup \{v_i\}$).

B2. **Merging Criteria:** in case v_i is added to the Local Community of v_0 as a consequence of the success of threshold or the admission criteria, if the number of nodes shared between C_0 and C_i is higher than γ times the number of nodes in the set union of C_0 and C_i, then the two Local Communities are merged (i.e., $if|C_0 \cap C_i| > \gamma \cdot |C_0 \cup C_i| \Longrightarrow C_0 = C_0 \cup C_i$).

C. k-CLIQUE community detection algorithm

k-CLIQUE is the distributed version of the centralised method originally proposed in [10]. Authors define a community as the union of all *k*-cliques which can be reached from each other through a series of adjacent *k*-cliques, where a *k*-clique is a complete subgraph of size *k* while two *k*-cliques are adjacent if they share $k-1$ nodes. An example of a 3-clique community is provided in Figure 1a.

In addition to the Familiar Set and Local Community, nodes exchange also their local approximation of the Familiar Sets of all the nodes within their Local Community ($FSoLC$)[3]. Focusing on node v_0, the following two criteria are evaluated at the end of the contact:

C1. **Admission Criteria:** in case of failure of the threshold criteria, if the Familiar Set of v_i contains at least $k-1$ nodes of the Local Community of v_0, then the encountered node v_i is added to the Local Community of v_0 (i.e., $if|C_0 \cap F_i| \geq k-1 \Longrightarrow C_0 = C_0 \cup \{v_i\}$).

C2. **Merging Criteria:** in case v_i is added to the Local Community of v_0 as a consequence of the success of threshold or the admission criteria, if the Familiar Set of each node inside the Local Community

[3]This is an approximation because it contains information about the Familiar Set of the encountered nodes which are collected during contacts and may be incomplete and not updated.

of v_i (e.g., v_j) contains at least $k-1$ nodes of the Local Community of v_0, then v_j is added to the Local Community of v_0 (i.e., $if|C_0 \cap \tilde{F}_j| \geq k-1 \Longrightarrow C_0 = C_0 \cup \{v_j\}$, where $v_j \in C_i$ and $\tilde{F}_j \in FSoLC_i$).

In addition, if the merging criteria are satisfied then the local approximation of the Familiar Sets of all nodes in C_0 needs to be updated, i.e., $FSoLC_0 = FSoLC_0 \cup \tilde{F}_j$.

D. MODULARITY community detection algorithm

MODULARITY is a variation of the method for discovering local community structures which has been presented in [13]. The original method is based on the concept of *Local Modularity (R)* and on the computation of the variation rate of the Local Modularity (ΔR) when adding a new vertex to an existing local community.

To define the Local Modularity the concept of *Boundary Set* should also be introduced. Specifically, the Boundary Set of a node (B) is defined to be the subset of vertices in a local community whose members have edges connecting to one or more vertices located outside the local community (see Figure 1b).

Taking into account the above definition, the Local Modularity measures the sharpness of local community boundary of each node and can be expressed by the quantity:

$$R = \frac{I}{|T|} \tag{1}$$

where I is the number of edges with no endpoints outside the Local Community and T is the set of edges with one endpoints in the Boundary Set. If the Boundary Set coincides with the Local Community, R is equal to 1 by definition.

When a new vertex v_i is added to an existing community C_0 of vertex v_0 with Boundary Set B_0, the variation of the Local Modularity can be computed by the following equation:

$$\Delta R_0 = \frac{x - R_0 \cdot y - z(1 - R_0)}{|T| - z + y} \tag{2}$$

where x is the number of edges in T that terminate in v_i, y is the number of edges that will be added to T due to the inclusion of v_i and z is the number of edges that will be removed from T due to the inclusion of v_i.

The variation to the Modularity algorithm proposed by [1] works as follows. When two nodes meet they exchange the following information: *i)* F, *ii)* C and *iii)* $FSoLC$. Once received, each node uses them to update its local structures. Specifically, v_0 first updates its local approximation of $\tilde{F}_i$ by merging it with F_i directly received by v_i. Then, it updates each local approximation of all the Familiar Sets in $FSoLC_0$ with the corresponding version in $FSoLC_i$. The same is for v_i as well. Afterwards each node evaluates the following criteria. Referring to node v_0:

D1. **Admission Criteria:** in case of failure of the threshold criteria, if the difference between the Local Modularity measured before and after including v_i into the Local Community of v_0 exceeds 0, then the encountered node v_i is added to the Local Community of v_0 (i.e., $if \Delta R_0 \geq 0 \Longrightarrow C_0 = C_0 \cup \{v_i\}$).

D2. **Merging Criteria:** in case v_i is added to the Local Community of v_0 as a consequence of the success of threshold or the admission criteria, the algorithm considers adding to C_0 the nodes inside the set K:

$K = \{v_k \mid \exists j s.t. v_j \in C_0 \cap C_i \wedge v_k \in \tilde{F}_j \wedge v_k \in C_i \setminus C_0\}$.

Specifically, the set K consists of the subset of all the nodes of C_i that are adjacent to those nodes shared between C_0 and C_i.

For each node v_k in K, if the Familiar Set of v_k is a subset of the Local Community of v_0, then v_k is added to the Local Community of v_0 (i.e., $if \tilde{F}_k \subseteq C_0 \Longrightarrow C_0 = C_0 \cup \{v_k\}$).

In addition, the variation of Local Modularity (ΔR_0) is computed for all the remaining nodes in K. Nodes with $\Delta R_0 > 0$ are directly added to the Local Community of v_0. For those nodes with $\Delta R_0 \leq 0$, the value of ΔR_0 is re-computed and re-checked after each inclusion. This procedure may be repeated several times and it ends when *i)* K is empty, or *ii)* after having added v_k to C_0, $\Delta R_0 \leq 0$ for all the remaining nodes in K.

Furthermore, after each inclusion, $\tilde{F}_k$ is merged with the corresponding version in $FSoLC_0$.

Note that the thresholds t_{th}, λ, γ and k used in the three algorithms are design parameters, thus they need to be chosen appropriately in order to calibrate the system. For this reason they are objects of investigation of the paper (see Section III-C).

As come out from the above description, the three community detection algorithms have different memory and computational requirements. On one hand, the SIMPLE algorithm makes use of low memory usage and low power computation. On the other hand, the MODULARITY algorithm is the most complex, both in terms of memory usage as it has to maintain a copy of Familiar Set of all the nodes in the Local Community and in terms of computation as it has to compute ΔR at each iteration for all the nodes belonging to set K. The k-CLIQUE algorithm represents an intermediate solution since it needs the same memory usage of MODULARITY but has lower computation complexity.

III. Performance Analysis

In this section we evaluate the communities detected by the distributed algorithms against the centralised algorithms, where a centralised algorithm has an a-priori knowledge of the composition of each community. We developed the described community detection algorithms as part of the OMNeT++ discrete-event network simulator[4]. The default scenario is composed of 54 nodes divided into 2 communities. Nodes move with an average speed of 1.5 m/s (representing a walking person) in a square of 1000mx1000m. Nodes move according to the HCMM mobility model [2] which is briefly described in the following subsection. The default parameters are set as in Table II. Despite being a particular and simple scenario, it is sufficient to highlight the properties of the community detection algorithms that we want to check.

A. Mobility Model: HCMM

HCMM [2] is a mobility model which well reproduces the statistical figures of real human movement patterns. Each node is initially associated with a specific community (its *home community*), and has

[4]http://www.omnetpp.org/

TABLE II
DEFAULT PARAMETERS

Parameter	Value	Parameter	Value
Area	1000mx1000m	N	54
Transmission range	20m	Community number	2
Average speed	1-1.86m/s	Traveller number	1

social ties with all the other members of its home community. Certain nodes (*travellers*) have also social links with communities other than the home (*foreign community*). For each social tie, the specific foreign community and the specific node inside the foreign community are selected according to a uniform distribution. The mobility pattern of a node is driven by its social links, i.e., a node located into its home community moves towards a given community (i.e., home and/or foreign) with a probability proportional to the number of ties with nodes of that community. In addition, when the node reaches a community which is not its home, it remains in the foreign community for the next movement with a given probability (p_e) and goes back home with probability $1 - p_e$. Hence, HCMM models a realistic scenario in which users are generally attracted to those people within the same home community, but they are also attracted to foreign people with whom they spend some time before coming back home.

HCMM permits to periodically re-select the foreign communities for the traveller.

B. Performance metrics

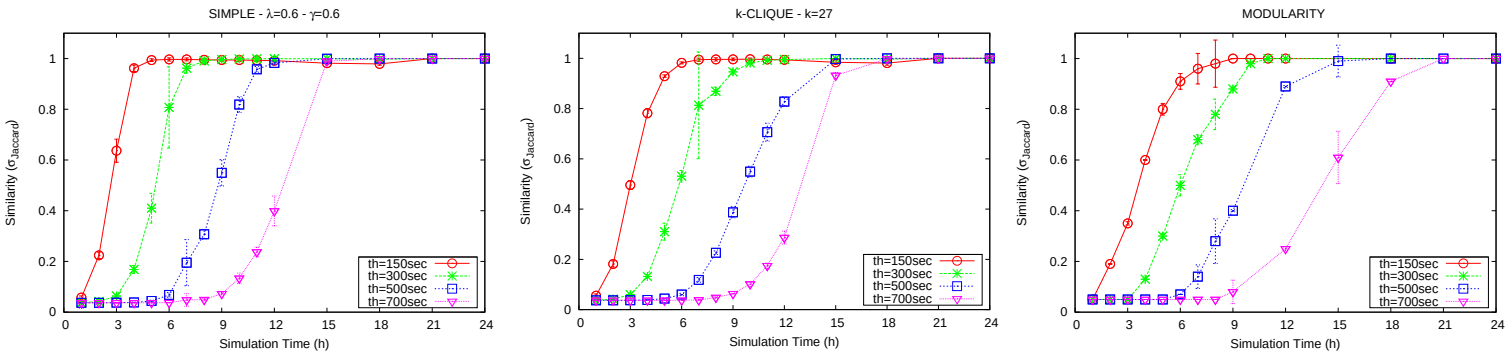
The objective of the simulation analysis is to measure how the distributed approaches proposed in [1] are able to identify communities formed by people. To this aim, we use the classic Jaccard index [14] as the metric to evaluate the similarity among the communities detected by the distributed algorithms and those detected by the centralised algorithms. The Jaccard index is defined by the following quantity:

$$\sigma_{Jaccard} = \frac{\left| G_i^{(c)} \bigcap G_i^{(d)} \right|}{\left| G_i^{(c)} \bigcup G_i^{(d)} \right|} \tag{3}$$

where G_i is the set of nodes which belongs to community *i* and $|G_i|$ is the cardinality of the G_i, while (c) and (d) indicate that the set is detected by the centralised algorithm and by the distributed algorithm, respectively. In the following we give an estimation of the local community similarity by averaging the Jaccard index for all the nodes. Specifically, each simulation is replicated 5 times and the results are averaged over all the replicas with 95% confidence intervals.

C. Results

The analysis presented in this section refers to the default scenario in Table II which is not subjected to any reconfiguration. Figure 2 shows the similarity metric as a function of simulation time and threshold value, respectively, for the three algorithms. Each curve represents, for a fixed threshold value, the corresponding $\sigma_{Jaccard}$ obtained at different points in time in the simulation. As depicted by the figure, the similarity value increases with the increase of simulation time for all the three algorithms. This is



(a) (b) (c)

Fig. 2. Impact of the Familiar Set threshold as function of simulation time for SIMPLE (a), k-CLIQUE (b) and MODULARITY (c).

quite obvious since increasing the simulation time corresponds to gather more precise information for the distributed algorithms.

The similarity also increases when the threshold value becomes lower. By reducing t_{th} the probability that the cumulative contact duration for pairs of nodes satisfies the corresponding criterion increases. The threshold should be also set taking into account the nature and the dynamic of the groups. If nodes have strong social interactions, the corresponding cumulative contact duration will be higher. On the other hand, if nodes have weak social interactions it is high probable that their cumulative contact duration remains low. A suitable tuning of the threshold value may guarantee better performance in terms of similarity. For example, we find out that, for a 6h simulation run, the average cumulative contact times are about 550 sec. In this case, appropriate values are 150 sec for k-CLIQUE and MODULARITY or even 300 sec for SIMPLE. By setting a threshold of 150 sec, all the three distributed algorithms can reach at least 95% of the performance of the corresponding centralised algorithm. In addition, SIMPLE is able to have good performance (around 80%) with a higher threshold (300 sec).

Figure 3 shows the similarity metric for SIMPLE when varying the admission and merging thresholds. The trend of curves is similar to figure 2(a). The similarity increases with the simulation duration as the system has more time to correctly detect the communities. However, a variation of λ and γ has less impact in the overall performance. There is a slight difference between curves, especially for values of λ and γ greater than 0.6. Note that similar trends are obtained for different values of t_{th}, but here they are omitted due to space reasons.

Concerning the k parameter of k-CLIQUE, note that it represents the largest community which we would like to detect. As a consequence, in this scenario it should be set to 27. Changing the scenario requires setting it to a different value.

A comparison among the three algorithms in terms of similarity is illustrated in Figure 4. It refers to a simulation running for six hours with the choice of parameters that optimises the behaviour of the three algorithms. We can see that k-CLIQUE and MODULARITY show almost similar performance while SIMPLE has better performance. This is more apparent when the threshold is set to 300 sec. In this case SIMPLE reaches 80% similarity while both k-CLIQUE and MODULARITY do not exceed 55%. However, the difference between the curves decreases for higher threshold values, where all the algorithms

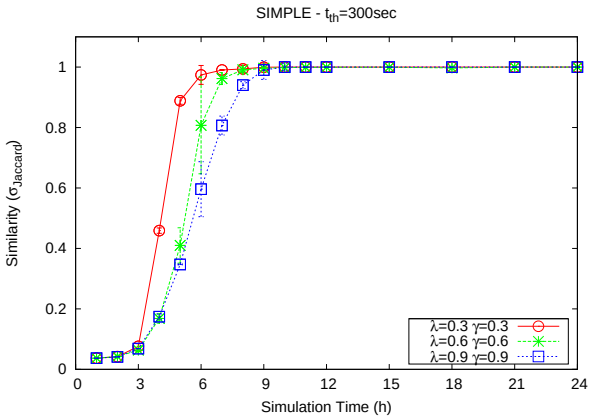


Fig. 3. Impact of λ and γ as function of simulation time for SIMPLE

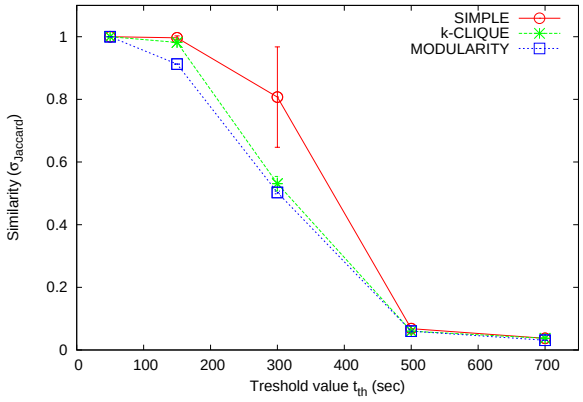


Fig. 4. Similarity as function of threshold value for SIMPLE(λ=0.6, γ=0.6), k-CLIQUE and MODULARITY

result in a very low performance. Note that the same line of reasoning can be applied for all the other investigated scenarios.

Summarising, from this analysis, SIMPLE results the more appropriate community detection solution as it is computationally lightweight and it generally performs better than the other two approaches.

IV. Towards an Adaptive Simple algorithm

Although the three presented algorithms are generally able to detect social communities, they suffer from a common limitation: during its whole life, each node (the traveller in particular) has not always a consistent view of its real familiar set and/or local community since it maintains memory of all encountered nodes.

For example, let's consider a scenario where a user A spends its time between two social communities: the work community during the day and the friend community after the working day. Let's also suppose that B is a colleague of A. Since A and B meet every day, it is correct to include B into the local community of A (and viceversa) for the time they meet at work. However, the contact with B becomes less relevant when A has finished to work and meets his friends for playing football, for example. This is even more apparent if B changes job or moves to another city. In above situations, no actions are made by the three algorithms to reflect these changes in the social community structures. On the contrary, B (A) will always remain in the A's community (B's community) even though they will not meet again.

From the above discussion, it turns out that the presented community detection algorithms need to be improved to well represent the dynamic users'social behaviours. This can be achieved by implementing some policies which take into account:

i) mechanisms for aging contacts among users;
ii) rules for deleting nodes from communities.

To this aim, in the following we present a novel social community detection algorithm, named **Adaptive Detection SIMPLE** (**AD-SIMPLE**), which is able to dynamically change social structures.

AD-SIMPLE takes inspiration from the original distributed SIMPLE. We focus on it since SIMPLE represents a good compromise among complexity and performance.

The main idea behind this algorithm is to maintain the basic features of the original SIMPLE for what concerns the inclusion of nodes while to improve its behaviour with additional policies in order to identify and remove from communities the nodes which have not be seen for a long period of time. Therefore, during a contact time, nodes exchange the Familiar Set and the Local Community and update their local view of social communities by applying the same criteria of SIMPLE (see section II-B). The additional policies to remove nodes from communities are carefully described in the following subsections.

A. Familiar Set pruning policy

The idea is to keep a running average of the percentage of the contact duration (over the total simulation time) for each node in the Familiar Set and to decide whether to remove or not a node if this percentage falls below a given threshold. To this aim, time is divided in slots (i.e., T) and the node computes the percentage of the contact duration in each slot for all the nodes in its Familiar Set as a $Sample\Delta T$. At the end of the slot, the node computes an $Estimated\Delta T$ as a weighted average value between the previous estimate and this new sample, as defined by equation 4:

$$Estimated\Delta T = \alpha \cdot Estimated\Delta T + (1 - \alpha) \cdot Sample\Delta T \tag{4}$$

where α is a parameter to smooth the $Estimated\Delta T$. Note that a small α tracks more quickly changes in the ΔT, while a large α guarantees a more stable value but it is not able to quickly adapt to real changes.

Finally, a node v_i with $Estimated\Delta T_i$ is deleted from the Familiar Set if the following equation is true:

$$Estimated\Delta T_i < FSout_{th} \tag{5}$$

where $FSout_{th}$ is the minimum threshold to be kept in the community.

B. Local Community pruning policy

The idea is to keep a timer for each node in the Local Community (i.e., $LCout_{timer}$). The timer is set when a node is inserted into the Local Community (e.g., v_i is inserted in C_0) and is refreshed when one of the following conditions occurs:

i) v_0 encounters directly v_i;

ii) v_0 encounters another node (e.g., v_j) and v_i is stored in the Local Community of v_j.

A node is removed from the Local Community when the corresponding timer expires.

If this happens and the node is in the Familiar Set, it is also removed from it. This guarantees to maintain consistency among the two sets during the pruning process.

In practise, *i)* and *ii)* are analogous to the criteria used for including nodes into Local Community. Note also that criterion *ii)* guarantees that a node is deleted only if no node of the community has met him for a long time. In fact, if there exists at least one node in the community with which it has still some social interactions, the node must still be kept in the Local Community of all the community members.

TABLE III
PARAMETERS USED FOR THE EVALUATION

Parameter	**Value**	**Parameter**	**Value**
Simulation time	172800 sec	Traveller number	1
Reconfiguration time	21600 sec	t_{th}	150 sec
N	54	γ	0.6
Community number	3	λ	0.6

V. RESULTS

In this section we evaluate the performance of the proposed protocol first in isolation and then by a comparison with the SIMPLE algorithm. We show that AD-SIMPLE results in higher values of the similarity metric because it identifies and correctly adapts to social changes.

A. Sensitiveness of AD-SIMPLE on α, $FSout_{th}$ and $LCout_{timer}$

As first analysis, we evaluated the sensitiveness of AD-SIMPLE on its parameters.

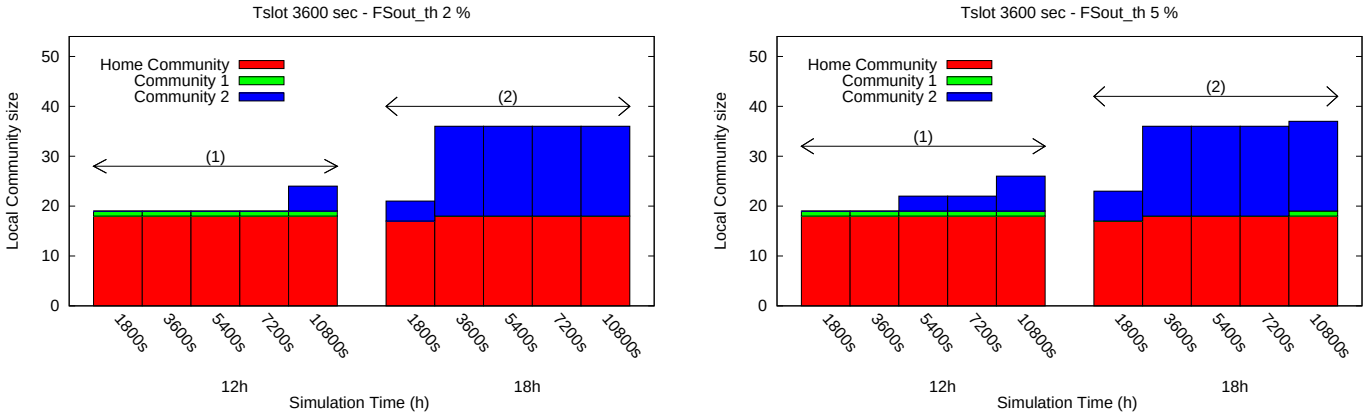
In scenarios such as those described in the previous section, we find out that the behaviour of AD-SIMPLE when varying t_{th}, λ and γ is similar to the original SIMPLE algorithm. This is quite intuitive since AD-SIMPLE makes use of the same criteria for what concern the inclusion of nodes in social communities. Specifically, we observe high variability by changing t_{th} (i.e., the similarity increase when the threshold becomes lower), while we find almost the same behaviour when varying λ and γ.

Concerning to parameters introduced for the Familiar Set policy, we observe that, for a fixed value of time slot T, $FSout_{th}$ should be chosen taking into account also the dynamic of social interactions. In our case study, where nodes have short social interactions (i.e., the cumulative contact duration is low), we note that $FSout_{th}$ should not exceed 5%, otherwise the Familiar Sets would remain empty for most of the time. Regarding the α parameter, we note that high values (i.e, more than 0.6) imply that quick changes at the social level are not reproduced so quickly at the Familiar Set level. Hence, it is important to maintain $\alpha <= 0.3$.

To better investigate how a proper tuning of the $LCout_{timer}$ parameter affects the performance of the LC pruning policy, we refer to a dynamic scenario where nodes may be connected to two communities simultaneously, i.e., home and foreign, and move between them for an entire reconfiguration interval. At each reconfiguration, such nodes select randomly one foreign community to start visiting. The simulation parameters are specified in Table III.

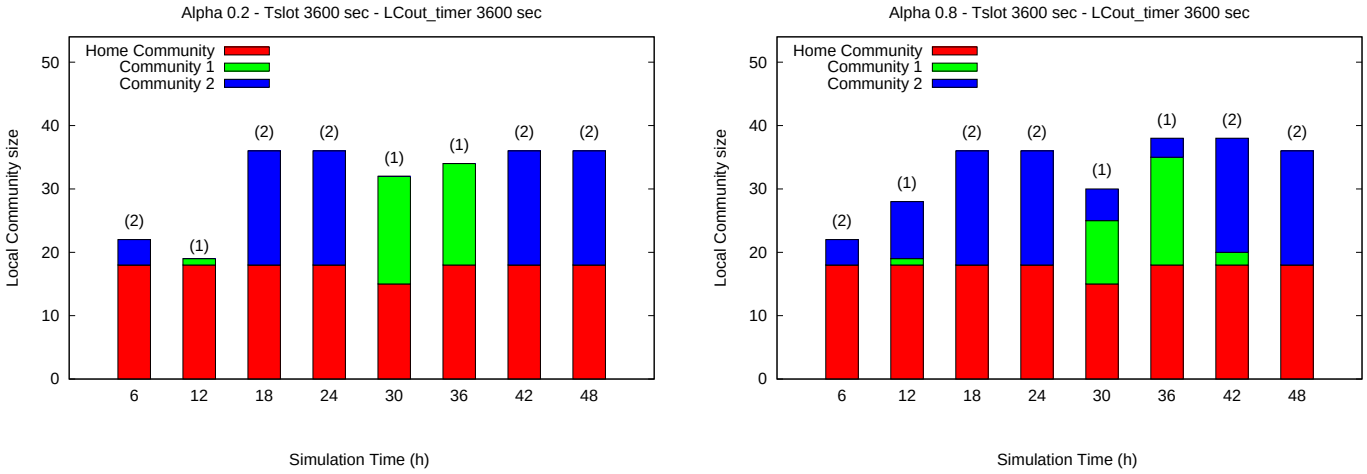
Figure 5 provides relevant examples of different behavior that can be obtained if such parameter is not correctly tuned. Each bar represents a snapshot of the Local Community composition for a traveller at the end of two reconfiguration intervals when varying $LCout_{timer}$ and the $FSout_{th}$. Note that the number in brackets above the bars indicates the foreign community visited by the traveller during that period of time.

By fixing T (3600sec in this example), $LCout_{timer}$ (on the x-axis) must be of the same order. In fact, if $LCout_{timer}$ is set too high (e.g., 10800 sec), nodes of a community previously visited may still stored in the Local Community. This is apparent for example in Figure 5(a) at 12h, where some nodes of community



(a) (b)

Fig. 5. Representative cases of the composition of the Local Community for AD-SIMPLE as a function of $LCout_{timer}$ and $FSout_{th}$.



(a) (b)

Fig. 6. Representative cases of the composition of the Local Community for AD-SIMPLE for different values of α.

2 are present in the Local Community or in Figure 5(b) at 18h, where nodes of community 1 are still present. Furthermore, $LCout_{timer}$ must not set less than T, as this could lead to a too rapid emptying of the Local Community (i.e., timer expires too rapidly). For example, in Figure 5(a) at 18h, only a small percentage of nodes of community 2 are present in the Local Community in case of $LCout_{timer}$ equal to 1800 sec. This means that the system has not a sufficient time to detect all the nodes which belong to the communities. From this analysis, we points out that it is important to properly tune $LCout_{timer}$ in order to maintain a consistent view of the social community at each point in time and a reasonable value is obtained by setting $LCout_{timer}$ = T.

Finally, note that $LCout_{timer}$ is influenced in part by the value of $FSout_{th}$ and in part by the value of α. For example, some nodes of community 2 still remain in LC when $FSout_{th}$ is set to 5% (see,

TABLE IV
AD-SIMPLE PARAMETERS

Parameter	Value	Parameter	Value
α	0.2	$FSout_{th}$	2
T	3600 sec	$LCout_{timer}$	3600 sec

for example, results of 3rd and 4th bars at 12h in Figure 5(b)). As far as the α parameter, if it set too high, not only the Familiar Set is not able to report changes, but Local Community too: a variation of α impacts also on the social community. This is highlighted by Figure 6 that shows results for two different value of α (i.e., 0.2 and 0.8). As it is apparent, it is important to maintain α low and specifically $<= 0.3$, which is also in line with that found earlier for the Familiar Set policy, otherwise this might cause wrong decisions when deleting nodes from the Local Community.

B. SIMPLE vs AD-SIMPLE: a comparison analysis

As far as the comparison with SIMPLE, we consider the same scenario as before (see Table III), which is composed of 54 nodes divided into 3 social communities. The simulation lasts for 48 hours with reconfigurations every 6 hours. This means that, after each reconfiguration, the traveller selects randomly one of the two foreign communities to start visiting. The rest of simulation parameters for AD-SIMPLE are summarised in Table IV.

In this scenario we aim at verifying the ability of AD-SIMPLE to correctly detect social communities, reflecting also any dynamic changes. To this aim, we mainly investigate the number of nodes discovered by the traveller during each reconfiguration interval.

Figure 7 and 8 show the simulation results of two selected simulation runs (SR), breaking down the Local Community composition for the traveller. The two simulation runs differ for the exact mobility actions taken by nodes, as well as for the sequence of foreign communities visited by the traveller during the simulation.

The limitation of SIMPLE is clearly illustrated by the figures. In both runs, the traveller's Local Community increases with the simulation time. After each reconfiguration interval, the traveller adds continuously new nodes to its community, while maintains memory of all the past encountered nodes. As a consequence, the community size reaches 54 after some time (i.e., 24h for both scenarios) and maintains the maximum size for the rest of the simulation. It is worth pointing out a particular behaviour of SIMPLE between 6h and 12h. Referring for example to SR 1, the number of nodes of community 2 increases even though the traveller is in contact with community 1. This is due to a transient phase which follows the beginning of each reconfiguration and it is strictly connected with the traveller location. As explained in Section III-A, if the traveller is visiting nodes of a foreign community, it goes back home with probability $1 - p_e$ while it remains in the foreign community for the next movement with probability (p_e). Thus, if the latter event occurs it will continue meeting nodes of the previous foreign community. This is the reason behind the highlighted increase at 12h.

As highlighted by Figures 7(b) and 8(b), AD-SIMPLE is able to keep always updated the community view of the traveller. Thanks to the implemented pruning policies, the traveller maintains in its Local

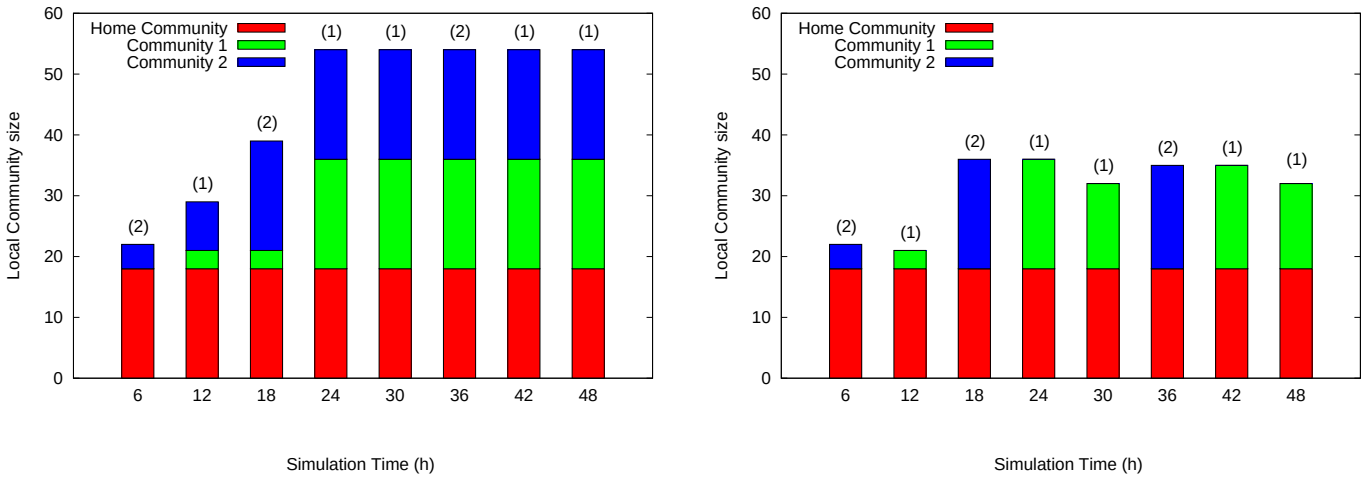


Fig. 7. SR 1: Local Community composition of the traveller for SIMPLE (a) and AD-SIMPLE (b).

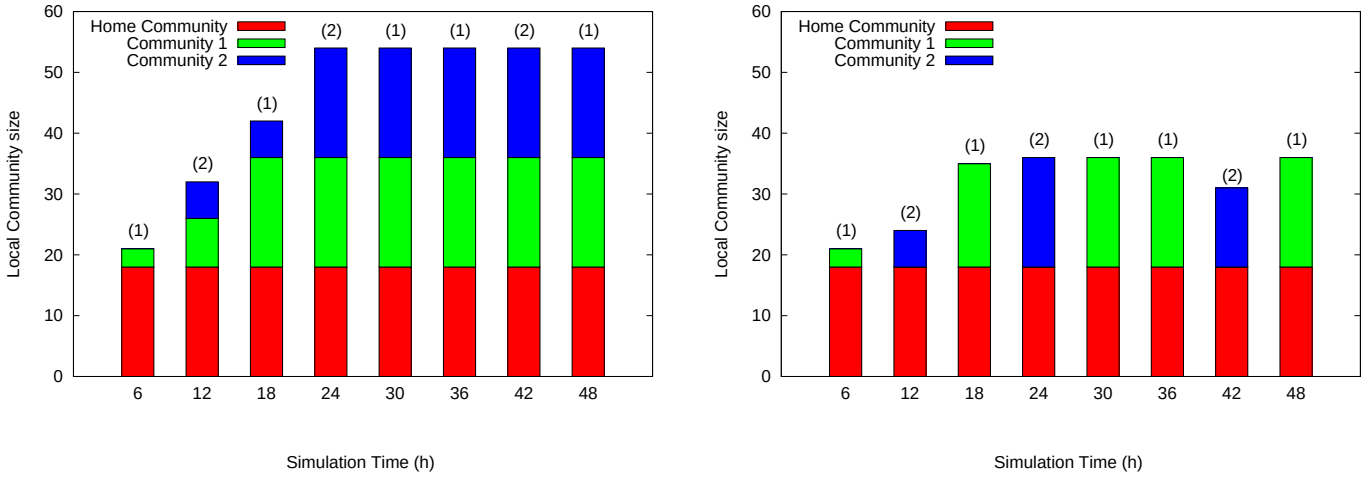


Fig. 8. SR 2: Local Community composition of the traveller for SIMPLE (a) and AD-SIMPLE (b).

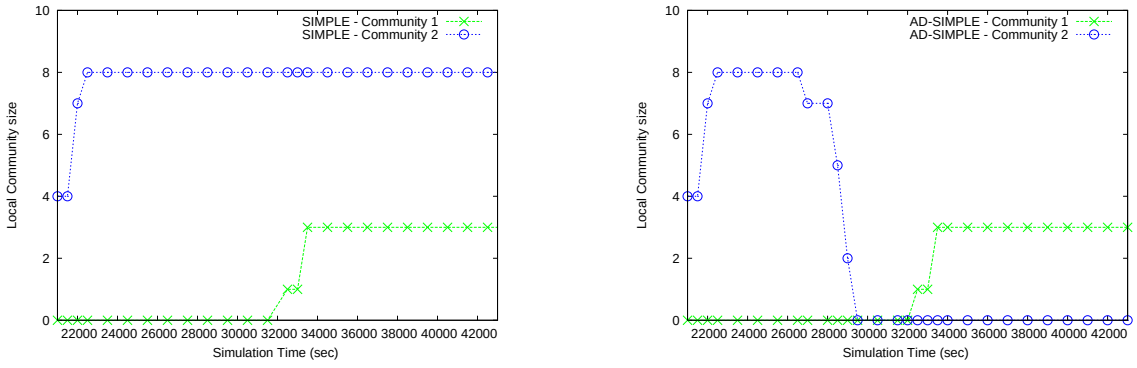
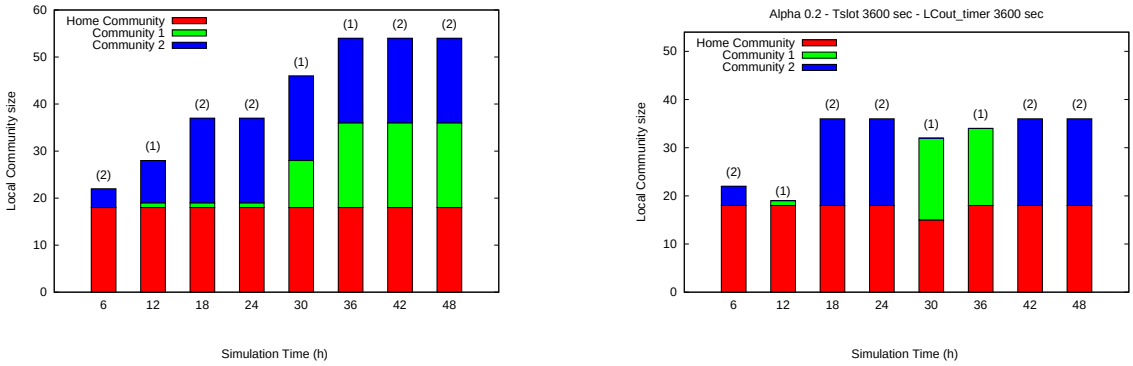
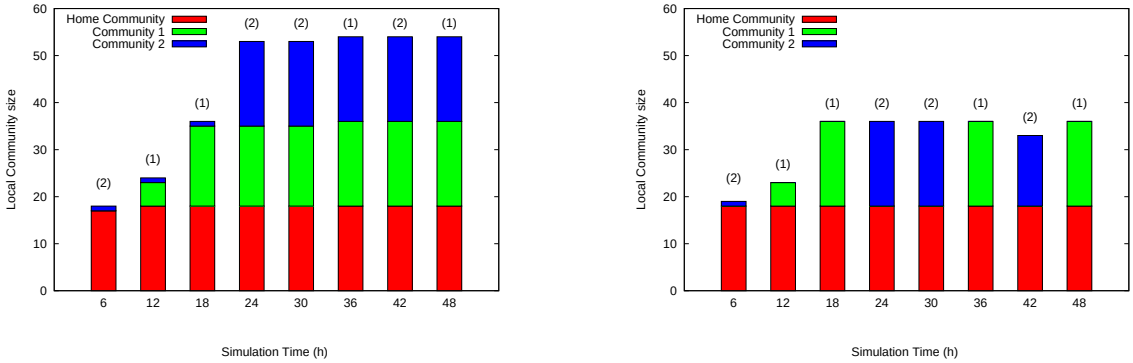


Fig. 9. SR1: Local Community evolution of the traveller for SIMPLE (a) and AD-SIMPLE (b).



(a) (b)

Fig. 10. SR3: Local Community evolution of the traveller for SIMPLE (a) and AD-SIMPLE (b).



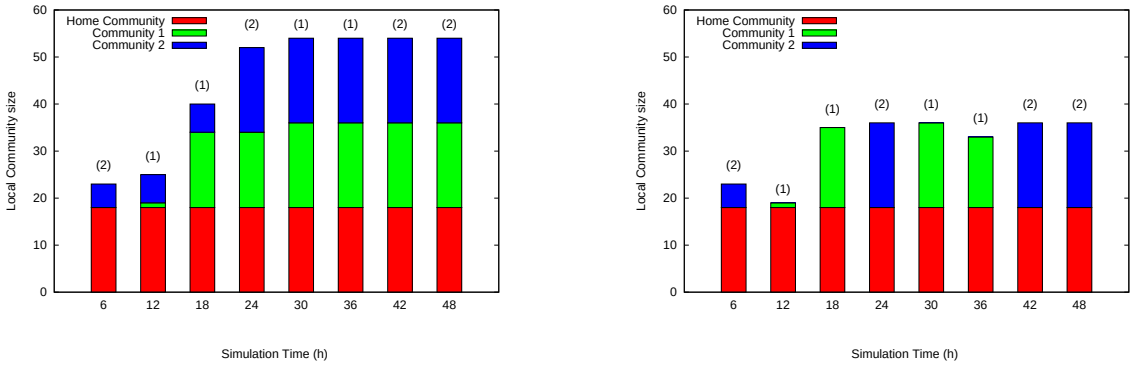
(a) (b)

Fig. 11. SR4: Local Community evolution of the traveller for SIMPLE (a) and AD-SIMPLE (b).

Community only nodes belonging to its home community and to the foreign community that it is visiting, while it removes the nodes that are not part of its community anymore. These entries either belong to the previous visited community (e.g., passing from community 1 to 2, it removes all the entries related to community 1 and viceversa) or are nodes of the currently visited community that it has not met for sufficient time. For example, this happens in SR 1 at 30h, where the Local Community size decreases even though the foreign community is not changed after the reconfiguration.

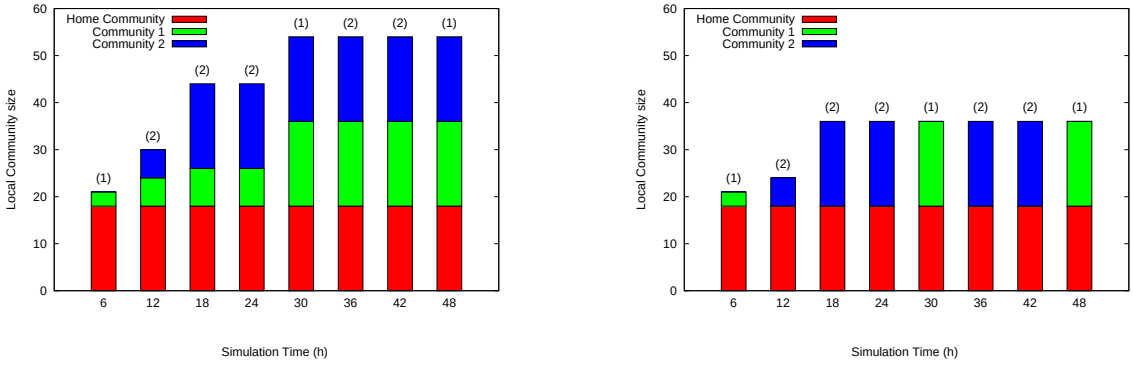
Note also the high increase of the Local Community from 12h to 18h. In this period, the traveller visits one of the community already visited. Thus, it is high probable to add more nodes as the correspondent cumulative durations are close to the threshold t_{th}. Figure 9 shows the evolution of the traveller's Local Community during the reconfiguration interval [6h,12h] (note that nodes belonging to the home community are not included in the graph) for both community detection algorithms. Focusing on the SIMPLE behaviour (see Figure 9 (a)), at the beginning (21600sec), the Local Community is composed of only 4 nodes of community 2. Between 21600sec-24000sec (transient phase), the curve of Community 2 increases since the traveller has still some social integrations with community 2. When these interactions become sporadic and eventually completely disappear, the curve remains constant. At the same time, since interactions with community 1 become predominant, the curve related to community 1 increases too.

Figure 9 shows the evolution of the traveller's Local Community during the reconfiguration interval



(a) (b)

Fig. 12. SR5: Local Community evolution of the traveller for SIMPLE (a) and AD-SIMPLE (b).



(a) (b)

Fig. 13. SR6: Local Community evolution of the traveller for SIMPLE (a) and AD-SIMPLE (b).

[6h,12h] (note that nodes belonging to the home community are not included in the graph) for both community detection algorithms. Focusing on the SIMPLE behaviour (see Figure 9 (a)), at the beginning (21600sec), the Local Community is composed of only 4 nodes of community 2. Between 21600sec-24000sec (transient phase), the curve of Community 2 increases since the traveller has still some social integrations with community 2. When these interactions become sporadic and eventually completely disappear, the curve remains constant. At the same time, since interactions with community 1 become predominant, the curve related to community 1 increases too.

Figure 9(b) shows the same simulation period for AD-SIMPLE. The behaviour of the curve of Community 2 is similar to the previous graph until 26500sec, then the curve decreases reaching 0 at 30000sec due to the expiration of the timeouts associated with Local Community entries. On the contrary, from 32000sec the curve of Community 1 increases since contacts with community 1 become frequent. At the end of the interval, the traveller's Local Community is composed only of nodes belonging to community 1 (and home, of course).

Figures from 10 to 13 provide results for additional simulation runs, here referred to as SR 3 - SR 6. It is worth noting that they show the same behaviour of SR 1 and SR 2. For what concern SIMPLE, when the scenario becomes more complex and dynamic (e.g., people belong to more than one communities simultaneously and spend much time in them), nodes no longer have a consistent vision of the true

TABLE V
SIMILARITY FOR SIMPLE AND AD-SIMPLE IN SR 1

Sim_time	SIMPLE	AD-SIMPLE	Sim_time	SIMPLE	AD-SIMPLE
6h	0.611	0.611	30h	0.66	0.88
12h	0.477	0.58	36h	0.66	0.972
18h	0.923	1	42h	0.66	0.972
24h	0.66	1	48h	0.66	0.88

composition of the community, mainly because they keep memory of all the nodes encountered so far. On the contrary, AD-SIMPLE overcomes such limitation and the traveller maintains only nodes belonging to the two communities which are currently visiting, while it removes those nodes which are not part of its community anymore.

We conclude this section with Table V which summarises the similarity results for SR 1. AD-SIMPLE reaches higher performance while SIMPLE results in very low performance due to its inefficiency of adapting to social changes. Similarity results of SR 2-6 are aligned with those of SR 1 and are omitted mainly for sake of space.

To summarise, AD-SIMPLE results a good solution as it performs better that SIMPLE when detecting social communities and keeps low the computational burden. In addition, its ability to capture the evolution of social interactions adapting to the changes makes it very attractive.

VI. CONCLUSIONS

In this paper we have evaluated the performance of three distributed community detection algorithms for Opportunistic Networks. We have compared them by a simulation analysis in scenarios where nodes move according to a mobility model which realistically reproduces the human mobility patterns. Results from this analysis have shown a good correspondence degree between the communities detected by the three distributed algorithms and the corresponding centralised algorithms. We have also highlighted a common limitation: nodes moving in different social communities may not have a consistent view of their local community since they maintain memory of all encountered nodes. To this aim, we have also investigated a new computationally light-weight solution, namely AD-SIMPLE, which overcomes the above problem. We have shown by simulation that AD-SIMPLE is able to adapt to the dynamic evolutions of social communities maintaining a consistent view of each user.

ACKNOWLEDGEMENT

This work was funded by the European Commission under the SCAMPI (258414) FIRE project.

REFERENCES

[1] P. Hui, E. Yoneki, S.-Y. Chan, and J. Crowcroft, “Distributed community detection in delay tolerant networks,” in *Proceedings of MobiArch’07*, 2007.

[2] C. Boldrini and A. Passarella, “Hcmm: Modelling spatial and temporal properties of human mobility driven by users,” *Computer Communication*, vol. 33, no. 9, pp. 1056 – 1074, 2010.

[3] L. Pelusi, A. Passarella, and M. Conti, “Opportunistic networking: data forwarding in disconnected mobile ad hoc networks,” *IEEE Commnication Magazine*, vol. 44, no. 11, pp. 134 – 141, 2006.

[4] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, “Efficient routing in intermittently connected mobile networks: The multiple-copy case,” *IEEE/ACM Transactions on Networking*, vol. 16, no. 1, pp. 77–90, 2008.

[5] X. Chen and A. Murphy, “Enabling disconnected transitive communication in mobile ad hoc networks,” in *Proc. of the Workshop on Principles of Mobile and Computing, in conj. with PODC’01*, Newport, RI, August 2001.

[6] B. Lidgren, A. Doria, and O. Shelén, “Probabilistic routing in intermittently connected networks,” *Mobile Computing and Communications Review Networks*, vol. 7 (3), July 2003.

[7] J. Leguay, T. Friedman, and V. Conan, “Dtn routing in a mobility pattern space,” in *Proc. of the ACM SIGCOMM 2005 Workshop on delay tolerant networks*, Philadelphia, PA, August 2005.

[8] C. Boldrini, M. Conti, and A. Passarella, “Exploiting users’ social relations to forward data in opportunistic networks: The HiBOp solution,” *Pervasive and Mobile Computing*, vol. 4, no. 5, pp. 633–657, 2008.

[9] P. Hui, J. Crowcroft, and E. Yoneki, “BUBBLE Rap: Social-based forwarding in delay tolerant networks,” *IEEE Transactions on Mobile Computing (to appear)*, 2011.

[10] G. Palla, I. Dereny, I. Farkas, and T. Vicsek, “Uncovering the overlapping community structure of complex networks in nature and society,” *Nature*, vol. 435, no. 9, pp. 814 – 818, 2005.

[11] L. Danon, J. Duch, A. Diaz-Guilera, and A. Arenas, “Comparing community structures identifications,” 2005.

[12] M. Newman, “Detecting community structures in networks,” *The European Physical Journal B - Condensed Matter and Complex Systems*, vol. 38, no. 2, pp. 321–330, 2004.

[13] A. Clauset, “Finding local community structures in networks,” *Physical Review E*, vol. 72, no. 2, pp. 026 132+, 2005.

[14] P. Jaccard, *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 37, no. 547, 1901.