SONIC-MAN: A Distributed Protocol for Dynamic Community Detection and Management

Barbara Guidi, Andrea Michienzi, Laura Ricci

SONIC-MAN: a distributed protocol for dynamic community detection and management

Barbara Guidi, Andrea Michienzi, and Laura Ricci

University of Pisa
Department of Computer Science
Largo B. Pontecorvo, 56127
Pisa, Italy
Email:{guidi, andrea.michienzi, ricci}@di.unipi.it

Abstract. The study of complex networks has acquired great importance during the last years because of the diffusion of several phenomena which can be described by these networks. Community detection is one of the most investigated problem in this area, however only a few solutions for detecting communities in a distributed and dynamic environment have been presented. In this paper we propose SONIC-MAN, a distributed protocol to detect and manage communities in a peer-to-peer dynamic environment. Our approach is particularly targeted to distributed online social networks and its main goal is to discover communities in the ego-network of the users. SONIC-MAN is based on a Temporal Trade-off approach and exploits a set of super-peers for the management of the communities. The paper presents a set of evaluations proving that SONIC-MAN is able to detect dynamic communities in a distributed setting and to return results close a centralized approach based on the same basic algorithm for community discovering .

Keywords: Peer to Peer, Community Detection, Complex Networks, Decentralized Online Social Networks

1 Introduction

Today, several important real life networks, such as those defined by social relationships, cryptocurrency transactions, biological systems, etc., are modeled by graphs. Due to the huge size of these graphs, their study requires novel methodologies, which are investigated in the research area referred as complex network analysis. Several algorithms and tools have been developed, mainly conceived in a centralized way.

However, in recent years we witnessed a dramatic shift from fully centralized applications to completely or partially distributed applications. One of the applications which have recently benefit from this shift are Online Social Networks, for whom a set of distributed proposals, i.e. Distributed Online Social Networks (DOSNs), have been recently presented. Implementing the social services in a distributed fashion is the key point of addressing well known problems, such as

the scalability of the service and, most of all, the possibility for the users to gain more privacy over their data. On the other hand, this scenario rises several challenges, such as data availability and information diffusion, mainly regarding the dynamic nature of the network.

One well known concept in the field of the social network analysis is the community detection, which is one of the most investigated problems in this area. Being able to group entities according to some rule, can be exploited to address many problems.

Many centralized algorithms [10] have been proposed for community detection, mainly based on different definitions of community. However, only a few proposal of distributed algorithms/protocols have been presented till now, even if the concept of community is useful for several problems arising in distributed scenarios, for instance to guide information diffusion in P2P networks, mobile or opportunistic networks. In particular, in the DOSN scenario, detecting communities in the ego networks of a node may support data replication strategies exploited to guarantee a high level of data availability. In these scenarios, it is necessary to define a distributed protocol to maintain and manage communities, in presence of an high level of dynamism of the network.

In this paper we propose SONIC-MAN, a distributed protocol for dynamic community detection and management targeted to DOSNs. The main characteristics of SONIC-MAN is the presence of a set of super-peer nodes whose goal is the management of the communities discovered in the ego network of a node. Several current approaches re-adapt existing community detection algorithms, such as the Label Propagation [17], in a distributed fashion. Instead, our approach relies on a set of super-peers, which apply a sequential algorithm for detecting communities and synchronize among themselves to maintain communities consistent. Our approach exploits the triangle as community model and uses a Temporal Trade-off approach [19] to manage the evolution of communities. The main novelty of our approach is that, while current approaches are conceived mainly in the field of data mining, our solution is better suited in a distributed system to discover and manage communities varying over time.

The paper is organised as follows. In Section 2 we describe the state of the art and related works. Section 3 contains the walkthrough of our approach and the results of our experiments are presented in Section 4. Finally, in Section 5, we conclude this paper and we point out some of the future works.

2 Related work

In this section we introduce the state of the art concerning the topics treated in this paper, starting from an overview on DOSNs, then presenting the main concepts of Community Detection, and finally concluding this section with a quick survey of Decentralized Community Detection.

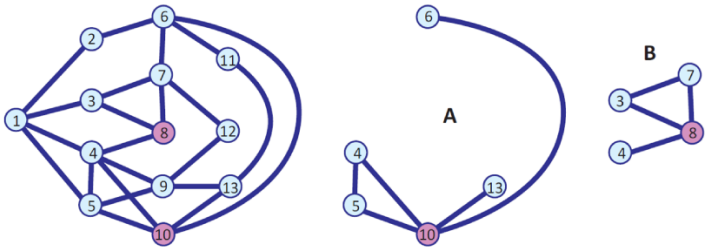


Fig. 1: On the left, an example of a network. Figure A shows the ego network of node 10 and figure B shows the ego network of node 8. It is worthwhile to notice that node 4 belongs to the ego network of both node 10 and node 8

2.1 Distributed Online Social Networks

A Distributed Online Social Network (DOSN) [5] is an online social network implemented on a distributed information management platform, such as a network of trusted servers, P2P systems or an opportunistic network. During the last years, DOSNs have been argument of several works from both academic researchers and open source communities. By decentralizing OSNs, there is no more a single service provider, but a set of peers that take over and share the tasks needed to run the system. In this contexts, social relationships are represented by a logical overlay, named social overlay [7]. In a social overlay, a connection between two users means that these users are friends. A common way to model the social overlay is through the concept of Ego Network (EN) [8]. The ego network is a Social network model that can be adapted in a P2P environment because it represents a user-centric view of the network, and it can be used to model the local knowledge of the network an ego has. An ego network is made of the user itself, which is also called *ego, its direct friends, known as* alters, and also include information about the direct connections between the alters. Formally, given a social network modeled through a graph $G = (V, E)$, where V is the set of vertices and E is the set of relations connecting the nodes, each vertex $u \in V$ can be seen as an ego and $EN(u) = (V_u, E_u)$ is the ego network of u where $V_u = \{u\} \cup \{v \in V | (u, v) \in E\}$, and $E_u = \{(a, b) \in E | a = u \vee b = u \vee \{a, b\} \subseteq N_u\}$. $N(u) = V_u \{u\}$ is the set of adjacent nodes of u. In Figure 1 an example of ego network is shown. On the left an example network is shown, on the right the ego network of two nodes is highlighted: the ego, represented by a red node, its alters, represented with blue nodes, and the relations among them.

2.2 Community Detection

Even if community detection is an important task in complex network analysis [1, 2], no common definition of community is currently available. However, each notion of community is based on the detection of a set of entities where each

entity is closer to the other entities within the community than to the entities outside it [4]. While community detection has been widely studied in static networks, the interest is quickly growing also for dynamic networks. This is because dynamic networks better model the dynamic nature of current complex networks such as social networks, economic networks and many more. In this case, it is even harder to formally define what a community is. A first very abstract definition is proposed in [19], where a community is identified with the sets of closely correlated nodes considering the fact that the relations between them may change over time, thus changing the communities.

Dynamic Community Detection algorithms can be classified into the following main classes [19]: *Instant-optimal Community Detection*, *Temporal Trade-off Community Detection*, and *Cross-Time Community Detection*. In the *Instant-optimal Community Detection* class, communities existing at time t are discovered by considering only the state of the network at time t. The network evolution is seen as a series of successive snapshots, each representing the state of the network at a particular instant of time. In the second class, *Temporal Trade-off Community Detection*, the communities identified at time t depend on the state of the network at all the instant of time less or equal t, possibly up to the initial known state. Typically communities are discovered by an iterative procedure which consist of an initial bootstrap, which yields the existing communities when the observation starts, followed by a set of successive updates to these initial communities. Finally, the *Cross-Time Community Detection* class includes all the methods that use all available information, i.e. past, current and future with respect to time t, to identify communities at instant t.

2.3 Decentralized Community Detection

Some studies about community detection or node clustering, in dynamic P2P networks have been recently presented. Many of these approaches are basically just a re-adaptation, in a distributed setting, of the Label propagation [17] approach. For instance, in [3] authors propose a revised label propagation divided in five phases, each of which has a different rule to update the labels of nodes. A simpler approach is presented in [13], where the rule to update the labels of the nodes is based on a similarity metric. In [14], authors propose a distributed approach for local dynamic community detection and three implementation variants. In this case, the distributed nature of the algorithm induces a very weak consistency among the nodes of the network. Also, the node clustering problem has been tackled with distributed approaches [18]. A downside of this approach is the fact that it is difficult to deal with node dynamics, i.e. join and leave, because, due to the presence or absence of nodes, the clusters may differ a lot. A common technique to tackle dynamism is to add new nodes to existing clusters and to periodically run from scratch the distributed clustering algorithm.

3 A novel distributed protocol for dynamic community detection

In this section we present the protocol SONIC-MAN (SOcial dyNamIc Community MANager), our distributed solution targeted to the detection of communities in the ego networks of a DOSN.

Contrary to the works present in literature, the approach defined in this paper is not a re-adaptation in a distributed fashion of another well-known approach and adopts a pure Temporal Trade-off approach. Moreover our approach guarantees a consistent view of the communities through all the nodes in the network, making them more usable in the development of distributed systems.

Our protocol is built on top of a 2-tiers architecture, described in [11]. The DHT, which implements the lowest layer, is used as look-up service to store information about super-peer nodes, which are managers of the information about the communities. The topmost layer is a logical social overlay modeled through an ego network [8]. The ego network is a well-known social network model used to represent user centric networks. Indeed, the ego network of a node is the subgraph of the whole network considering only the node itself, its neighbours and all the edges between these nodes.

3.1 An overview of SONIC-MAN

We investigated the design of an approach to support dynamic community discovery in ego network of DOSNs, and, as suggested in [12], our natural choice for the distributed community detection has been the Temporal Trade-off one. This choice was driven by the fact that the other two approaches do not naturally fit a dynamic distributed scenario. Cross-Time CD is not usable in principle due to the fact that the results of the algorithm are needed while the system lives, so future information is not available. Instant-optimal CD has the major issue connected to the need of a separate mechanism to match communities discovered at each time instant which may lead to wrong matchings. Another major downside of Instant-optimal CD is the fact that each time communities must be detected, we need the current snapshot of the network. This results in having synchronization, or strong consistency, between nodes, which is difficult to obtain in distributed asynchronous systems [9].

SONIC-MAN may be exploited to discover and manage the communities within the ego network of a node of a distributed online social network. The protocol considers the dynamism of the communities, due to the fact that nodes may autonomously change their presence status from online to offline, and vice-versa.

The algorithm is executed by the nodes of the social network themselves and exploits a set of super-peers, chosen among the nodes of the ego networks, which execute a Temporal Trade-off algorithm and maintain the discovered communities. The management of communities is essential due to the fact that dynamic communities show a life-cycle during their evolution, which is characterized by

a list of events [2, 16, 21]. In a distributed environment, these events are generated by the temporal behaviour of nodes and they affect the structure of the ego network, as explained in [12]. Within each community we identify one peer which leads and manages the community. In particular, this super-peer, called *moderator* of the community, can decide which nodes belong to the community, following a set of rules explained later in this section. Considering both the high dynamism and the possibility to have involuntary disconnections, we introduce a secondary moderator which acts as a normal peer, while the primary moderator is up and running. In all the cases a new secondary moderator is needed, it is elected by the primary moderator, choosing at random a node inside the community. The two moderators execute a ping/pong protocol to detect each other failures. When a failure is detected, the secondary moderator takes the role of the primary moderator. SONIC-MAN exploits a DHT to store information about moderators of active communities, so joining nodes may discover moderators of the communities of the ego networks.

Each moderator maintains a copy of the ego network of each ego for which it is moderating at least a community. This design choice is made to allow the moderator to iterate the evaluation of the communities when some event occurs, so avoiding to repeat the access to the community structure and saving a huge amount of communication.

Assuming that a moderator knows the ego network for which it is maintaining a community, we can also make very strong assumptions about the structure of the graph which represents the current state of the network. When a node enters or leaves the network, each moderator has the possibility to update each community that node belongs to. Indeed, when a node joins an ego network, the moderators can detect if the joining node belongs to the communities they manage by knowing only the identity of the node. This is because they know to whom the joining node is connected inside the ego network and who are the nodes inside the community.

In this work we decided to adopt a triangle-based community model. A node, to be accepted inside a community, must close a triangle with two other nodes already inside the community. This choice helps in having tight and clustered communities, and, at the same time, avoiding building communities made of chain-like structures. The same model is used in TILES [20] or in the 3-cliques percolation methods, and they are all linked to the concept of clustering coefficient.

3.2 A walkthrough of SONIC-MAN

In the following, we describe in detail the expected behaviour of a node running SONIC-MAN when it joins the network. An important point is to keep in mind that each node is, at the same time, ego for its ego network and alter for the ego networks of its neighbours in the social network.

Node join A node joining the network, after joining the DHT, must undertake two independent actions:

1. Join the active communities belonging to the ego networks of its alters;
2. Retrieve the active communities belonging to its ego network.

Algorithm 1 shows the steps needed to complete the first action. For each of its alters a (which are also ego of their respective ego network), the joining node n firstly searches in the DHT for the moderators of a, then it notifies all the moderators of a that it is now online (line 2-3). The joining node then sends to the moderator the identity e of the ego networks it is joining to (as alter), because each moderator can manage the communities of a set of ego networks. Finally, if the moderator m inserts the joining node in a community, the joining node records the identity of the moderator.

The moderators will update each community based on the nodes inside the community itself and how they are connected to the joining node. It is important to highlight that, in the Temporal Trade-off algorithm used by SONIC-MAN, to enter a community, a node must be involved in at least a triangle with two different nodes that are already inside the community. However, this behaviour is customizable, and can also be different from node to node. If the updated community contains the new node, its moderator will notify the joining node that it is part of its community for that particular ego network.

Algorithm 1: Node joining the network as alter of e

```
1 procedure NodeJoin(Ego e)
2     moderators ← DHT.getModerators(e);
3     for each moderator m ∈ moderators do
4         reply ← m.send(e,"I_AM_ONLINE");
5         if reply = true then
6             store(m);
```

An online node also periodically checks that it is still in a community for each of its neighbours. If it finds out that it is lacking some communities, it tries to build one using the Algorithm 2. The node pings all its neighbours which also belong to the ego network for which a community is missing (line 4). Then, after a fixed timeout, it tries to search for triangles (line 7) among its online neighbours. If one is found, the community is formed (line 8). Upon the birth of the community, the node itself become the primary moderator and it is in charge of updating the list of the moderators in the DHT and notify all the nodes inside the community that a new community is formed and that it is the primary moderator (line 9). If, instead, no triangle is found, the node waits for a timeout and then repeats the process, until a community is found for that particular ego network. It may be the case, if very specific conditions are met, that two nodes build the same community for the same ego network at, roughly, the same time. As stated later in this section, moderators periodically check whether communities can be merged together. This case falls inside the general case of the merging of two communities.

Algorithm 3 shows the sequence of the actions that an ego must do to retrieve the communities in its ego network. Even if this is not strictly necessary, because communities could still be managed by the moderators or even destroyed while the ego is online, it is desirable for the ego to maintain the communities by itself even if it is online. In this way, when the ego leaves the network, the communities have not to be reconstructed from scratch so avoiding an overhead and smoothing the communities over time. To retrieve its communities, the ego searches in the DHT for the moderators of such communities (line 2), then it notifies the moderators it is now online and sets itself as the only moderator for its ego network (line 4). The moderators will send to the ego the community they are managing and will stop manage it right after.

Algorithm 2: Creation of a new community in the ego network of e

```
1  procedure CommunityForm(Ego e)
2      while true do
3          for each neighbour n do
4              pong[n] ← n.ping();
5          wait(timeout);
6          if searchTriangles(pong[]) then
7              createCommunity();
8              DHT.addModerator(e, myself);
9              return
10         wait(timeout);
```

Algorithm 3: Node joining the network as an ego

```
1  procedure EgoJoin(Ego e)
2      moderators ← DHT.getModerators(e);
3      for each moderator m ∈ moderators do
4          reply ← m.send(e, "I_AM_ONLINE");
5          store(reply.community);
6      DHT.setModerator(e, myself);
```

Node leaving When a node leaves the network it has to undertake actions for each of the communities it is part of and for each of the communities it is moderator of. When the node leaving the network is just a simple node, not a moderator, it just sends a message to the moderators of the communities it belongs to, notifying them that it is going offline, which, in turn, update the structure of the community. When a node leaves the community, the moderator builds sets of adjacent triangles (sharing two nodes), called *triangle components*. A *triangle component* is defined as a set of nodes which forms triangles which

share at least one node in twos. For each of the *triangle component* a new community is created and a new primary moderator is elected for each of the new communities, while all members are notified that the community is no more. The new moderator, upon receiving the community, notifies the nodes inside that it is a moderator of a new community and adds an entry in the DHT. If the leaving node is a primary moderator, it informs the secondary moderator it is leaving and stops managing the community. The secondary moderator promotes itself as primary moderator, it elects a new secondary moderator and updates the DHT. Then it updates the community roles and tells all the community members it is the new primary moderator. Finally, if it is the secondary moderator of a community that it is leaving the network, it has to inform the primary moderator it is leaving. The primary moderator elects a new secondary moderator, updates the community roles and updates the DHT as well.

A special case of node leaving is the failure of nodes. Whenever the primary moderator discovers that a node inside its community failed, it is its duty to update the community roles as if the node voluntarily left the network. It is worth to point out that if the failed node is the secondary moderator, the primary moderator also has to promote a node to secondary moderator. The node failure may be discovered by the primary moderator by itself, by pinging the nodes inside its community, or by a clue coming from the other nodes inside the community. One last case is the one where the primary moderator itself is the failing node. In this case, the secondary moderator should discover this event because each pair of moderators of a community run a ping/pong protocol. The secondary moderator can treat this situation as a voluntarily leave of the primary moderator.

Community merging The community merge event is a non trivial event because it requires to detect the communities which have to merged. For this reason, periodically, a synchronization phase between moderators is executed with the main goal of handling merge events. During this phase, moderators with highly overlapping communities communicate each other so that, when a community is fully contained into another one, the smaller one can be absorbed (merged). When two communities have to be merged, the moderator of the smaller community destroys the community it is managing and deletes the relative entry in the DHT. No further action is needed because all the nodes inside the smaller community are also part of the larger one, so they have a reference to at least a moderator.

4 Experimental results: a Facebook case study

In this section we present an evaluation of SONIC-MAN by using a Facebook dataset. The evaluation was performed by simulations using PeerSim [15], an extremely scalable simulation environment that supports dynamic scenarios such as churn and other failure models.

4.1 The dataset

The dataset we used for our experiments contains information from real Facebook users gathered by a Facebook application, SocialCircles![1]. The application was deployed in 2014 and has gone under maintenance on the 1st of May 2015 due to the change of the Facebook APIs which were substantially reduced in size. As described in [6], SocialCircles! was able to retrieve information about the topology and profile of registered users, and the online behaviour of them and their friends. The dataset is composed by 240 users monitored and their complete ego networks (for a total of 78,129 users). For each of the registered users we were able to gather their profile and ego network, and the interactions between them and the alters. Moreover, we also obtained temporal information about the total 78,129 users for 32 consecutive days. In detail, we sampled all the registered users and their friends every 5 minutes, for 32 days (from the 9th March 2015 to the 10th April of the same year).

4.2 An analysis of temporal information

Since the time aspect is the cornerstone of our research, we preliminary analysed the temporal information contained in our dataset. This preliminary analysis is aimed to understand a general trend of online/offline behaviour of the users.

We start by recalling that, in our dataset, time is modeled in a discrete way to represent the online/offline status of the users. In particular, each day of the monitored period consists of a finite number of time slots (i.e., 288 time slots each of 5 minutes), for a total number of 9251 time slots in the whole monitored period. For the sake of our analysis, by considering that a Facebook user can have three different status values (active, idle, and offline), we do not make any distinction if the user is active or idle, because in both cases the user is online, and his device can still help the network in delivering the service.

Online users A first interesting analysis, is to show which are the time spans during which the OSN is more crowded, by simply counting the online users at each time slot. In figure 2 we put the sorted time slots on the x-axis and on the y-axis we report the number of online users for each time slot. The Figure shows that there is a clear periodic pattern which reflects the day/night cycle. The repeated pattern shows two peaks and one nadir. By taking a closer look at the timestamps of the corresponding time slots, we can notice that the first peak is always registered around 12 PM (midday), the second one is registered around 9 PM and the nadir is registered around 6 AM. This observation confirms that users tends to follow a cyclic pattern of accesses to the service which may be guided by life duties and habits. By analyzing the amount of users online for each time slot, we can see that we have at most around 18,000 online users, roughly 23% of the total amount, and at least 3000, 3.8% of the total amount of users.

[1] https://www.facebook.com/SocialCircles-244719909045196/

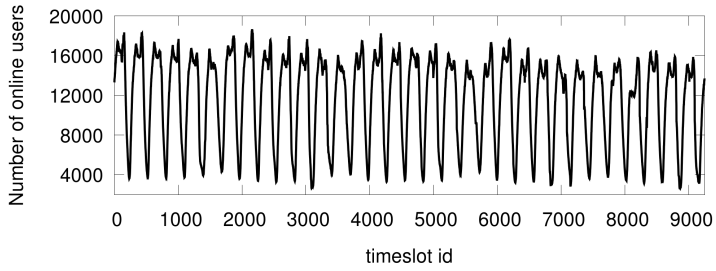


Fig. 2: Online users count during the observed period

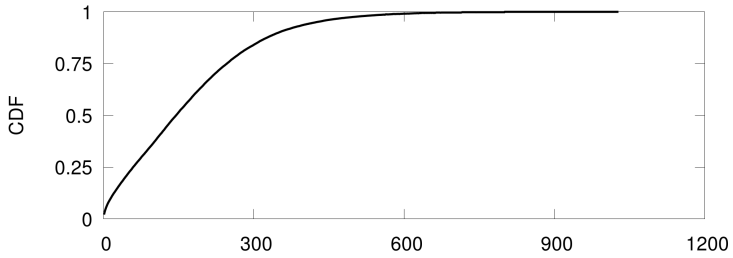


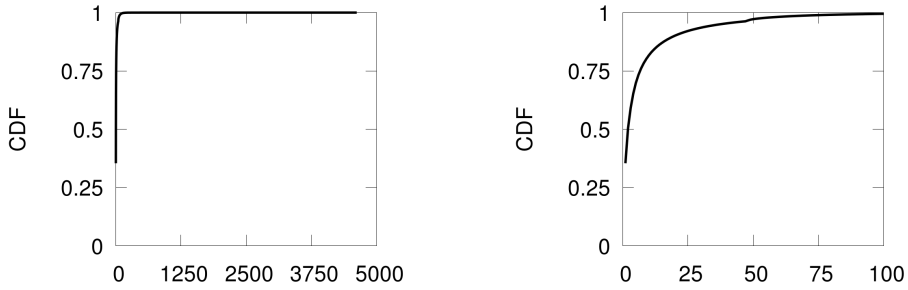
Fig. 3: CDF of the number of sessions for each user in the observed period of time

Online sessions A second analysis is to show for how long users remain online once logged in and how soon they come back online after going offline. To this aim we computed three measures: the session number, the session length, and the session inter-arrival time. A session is defined as the amount of time continuously spent on the OSN by a user.

The session number is the total amount of sessions of all the users. Figure 3 shows the Cumulative Distribution Function (CDF) of the number of sessions. As we can see most of the users has an high number of sessions during the whole observed period of time. More in detail, we can see that half of the users have more than 175 sessions and that only 20% of the users have 50 or less sessions. It is worthwhile to notice the presence of a very small fraction of inactive users, which had no sessions over the observed period of time.

The session length is the duration of a session of a user. It is important to point out that, since the time in our dataset is represented in a discrete way (time slots), each time we observe an online user in a particular time slot, we assume that the user was online during all the duration of that time slot. Figure 4a shows the CDF of the length of all the sessions. The plot shows that the majority of the sessions are very short with respect to the whole observer period. To better show the arrangement, we decided to make another plot to zoom on the leftmost part of the CDF. Figure 4b shows the CDF of the length of all the sessions, restricting to lengths ranging from 0 to 100 slots. As we can see from this figure,

more than 80% of the sessions are shorter than 10 slots (50 minutes) and half of the sessions are, at most, 3 slots long.



(a) Plot showing the CDF of all session length times

(b) Zoom, showing only sessions whose length is ≤ 100

Fig. 4: CDF of the length of all the sessions

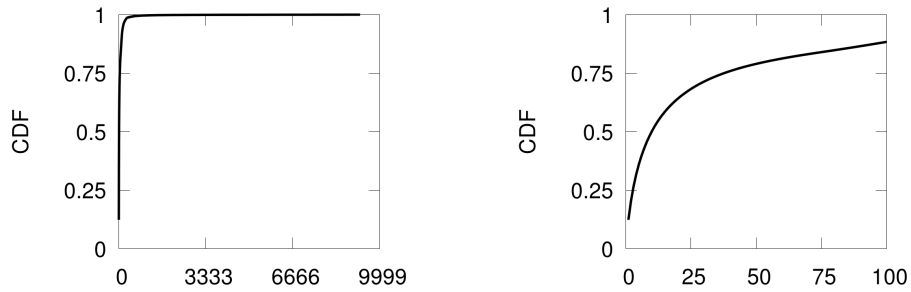
The session inter-arrival time measures the amount of time passing between two sessions of the same user, i.e. how much time a user spends offline. Figure 5a shows the CDF of the session inter-arrival time, and we can see that it is highly left skewed. Again, to better show the arrangement of the inter-arrival times, we decided to make a plot zooming on the leftmost part. Figure 5b shows the CDF of the length of all the sessions, restricting to lengths ranging from 0 to 100 slots. Session inter-arrival times tend to be slightly longer with respect to the session lengths, but are still very short.

These analyses showed that, as expected, the network is highly dynamic, and that we must take into account the fact that users tend to connect and disconnect from the service quite often.

4.3 A study of dynamic communities

We tested SONIC-MAN to assess its capabilities in finding community structures in a dynamic network and then we compared the results with a similar, but centralized, approach: TILES [20]. For the purposes of this paper, we consider TILES as a centralized oracle. The choice of the algorithm with whom we compare SONIC-MAN was driven by two reasons, the first one being the fact that TILES is also a Temporal Trade-off CD algorithm and the latter motivation is the fact that it shares with this implementation of SONIC-MAN almost the same definition of community.

First of all, we analyse how community events, listed in [12], are redefined considering the working method of SONIC-MAN. Community detection algorithms emit sets of communities that can be matched later. Indeed, the matching phase was only a guess of the exact evolution of communities. Instead, thanks to our



(a) Plot showing the CDF of all inter-arrival times

(b) Zoom, showing only inter-arrivals whose length is ≤ 100

Fig. 5: CDF of the session inter-arrival time

approach, we know exactly which is the evolution of each community. Events can be precisely detected by monitoring how the joining and leaving of nodes affect the community itself. We redefined the events as in the following:

- *Birth:* we say that a community is born whenever a node without a community succeed in closing a triangle which includes the node itself;
- *Death:* we say that a community is dead whenever, after a node leaves the community, there are no more triangles in the community;
- *Merge:* we say that two or more communities merge into a single community if each of the merging community is a subset of the same community, which will be the single one surviving the merging;
- *Split:* we say that a community splits into a set of communities if, after a node leaves the splitting community, there are two or more triangle components.

We are ignoring all the other events because they are less relevant in a distributed environment. Our evaluation concerns the output of the algorithms considering at first the communities and then the community events.

The first result is the number and size of communities discovered by SONIC-MAN. Table 1 shows the minimum, maximum, mean and standard deviation of number, aggregated by ego network, and size of the communities of both SONIC-MAN and TILES. The results of the table clearly confirms that the network is completely shattered. For SONIC-MAN, a high value of the size of communities confirms that, even tough the average number of online users is low, there is still an underlying community structure. Instead, TILES detects fewer communities in each ego network which are bigger with respect to the ones discovered by SONIC-MAN. We can explain this difference considering the fact that our algorithm is decentralized in contrast with the centralized nature of TILES.

After analysing the general structure of the network, we investigate the community events listed above. To do so, we compute some statistical measures on the events identified: birth, death, merge, and split events, aggregated by time

	SONIC-MAN				TILES			
Measure	Min	Max	Mean	Std. Dev.	Min	Max	Mean	Std. Dev.
Number	0	48	7.6	4.8	0	74	4.0	4.1
Size	3	903	13.9	23.5	3	336	7.6	9.5

Table 1: Statistical measures on number and size of dynamic communities detected by SONIC-MAN and TILES

	SONIC-MAN				TILES			
Event	Min	Max	Mean	Std. Dev.	Min	Max	Mean	Std. Dev.
Split	0	30	5.3	4.3	0	8	0.2	0.8
Merge	0	31	7.0	5.5	0	146	60	34.6
Death	0	198	47.4	28.1	0	1659	551.5	289.3
Birth	0	1198	214.7	71.8	0	2412	551.6	289.6

Table 2: Statistical measures on community events detected by SONIC-MAN and TILES

slot and ego network. The results obtained by SONIC-MAN, reported in table 2, show a strong predominance of birth events and a very low number of split and merge events. This result clearly confirms the strong dynamism of the nodes of the network: communities form and dissolve at a high rate, each time a node joins or leaves the network. Of great interest is, again, a comparison with the results obtained by TILES (Table 2). In this case we observe an overall tie between birth and death events which dominate, in number, merge and split events. So, up to this point, we can say that there is a real community structure, but it is very unstable and hard to detect.

To further investigate this situation we decide to plot the number of the events divided for each time slot. The arrangement of events is showed in Figure 6 for SONIC-MAN and in Figure 7 for TILES. Events from the first time slot have been removed to better visualize the graph, because it was in the first time slot that we registered the maximum number of birth and death events. Again, this clearly shows both the temporal pattern already observed in Figure 2, and the predominance of birth and death events with respect to merge and split events, as seen in table 2. Nonetheless, there are some differences between the two graphs. First of all, as expected, death events are a less then birth events in figure 6 with respect to figure 7. Another interesting fact is that the overall number of events detected by SONIC-MAN is lower for all the events. While this, at a first sight, seems to be a bad property of our protocol, it is instead consistent with the philosophy behind it. We recall that, rather than developing another data mining tool, as in TILES, our protocol was designed as support to the development of more sophisticated distributed application.

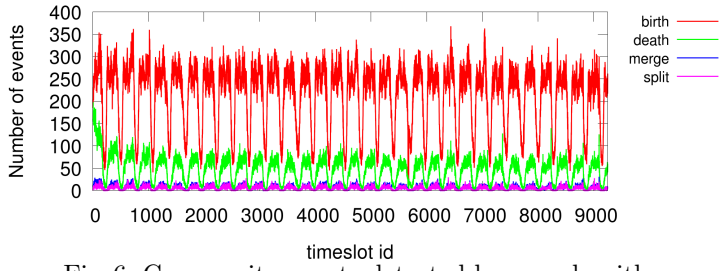


Fig. 6: Community events detected by our algorithm

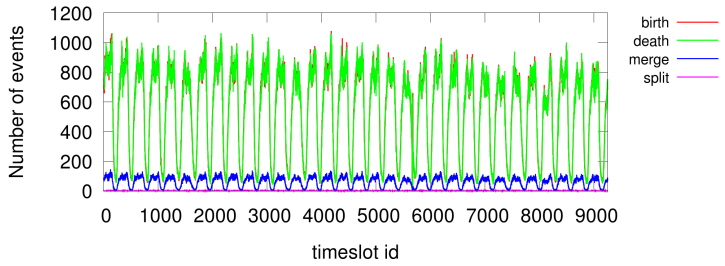


Fig. 7: Community events detected by TILES

5 Conclusion and Future works

In this paper we propose SONIC-MAN, a new distributed algorithm for dynamic community detection and management. This algorithm is designed to be used in DOSNs when a notion of community over time is needed to manage data availability and information diffusion. Our algorithm follows a *Temporal Trade-off* approach to keep overheads as low as possible while not giving up to the community quality. Moreover the communities detected are consistent through all the network thanks to the usage of super-peers. We evaluated our approach, comparing it with the results obtained with a similar, but centralized approach. The results show that SONIC-MAN provided results comparable to the centralized approach.

In the future we will evaluate new approaches for selecting the moderators and we will define a strategy to replicate social content exploiting the communities detected by SONIC-MAN. Finally, we are going to develop a completely decentralized version of the algorithm which avoids the use of super-peers.

References

1. Aynaud, T., Fleury, E., Guillaume, J.L., Wang, Q.: Communities in evolving networks: definitions, detection, and analysis techniques. In: Dynamics On and Of Complex Networks, Volume 2, pp. 159–200. Springer (2013)
2. Cazabet, R., Amblard, F.: Dynamic community detection. In: Encyclopedia of Social Network Analysis and Mining, pp. 404–414. Springer (2014)

3. Clementi, A.E.F., Ianni, M.D., Gambosi, G., Natale, E., Silvestri, R.: Distributed community detection in dynamic graphs. CoRR abs/1302.5607 (2013)
4. Coscia, M., Giannotti, F., Pedreschi, D.: A classification for community discovery methods in complex networks. Statistical Analysis and Data Mining: The ASA Data Science Journal 4(5), 512–546 (2011)
5. Datta, A., Buchegger, S., Vu, L.H., Strufe, T., Rzadca, K.: Decentralized Online Social Networks. In: Handbook of Social Network Technologies and Applications, pp. 349–378 (2010)
6. De Salve, A., Dondio, M., Guidi, B., Ricci, L.: The impact of user's availability on On-line Ego Networks. Comput. Commun. 73(PB), 211–218 (2016)
7. De Salve, A., Guidi, B., Ricci, L.: Evaluation of structural and temporal properties of ego networks for data availability in dosns. Mobile Networks and Applications 23(1), 155–166 (Feb 2018)
8. Everett, M., Borgatti, S.: Ego network betweenness. Social Networks 27, 31–38 (01 2005)
9. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM 32(2), 374–382 (1985)
10. Fortunato, S.: Community detection in graphs. CoRR abs/0906.0612 (2009)
11. Guidi, B., Amft, T., Salve, A.D., Graffi, K., Ricci, L.: Didusonet: A P2P architecture for distributed dunbar-based social networks. Peer-to-Peer Networking and Applications 9(6), 1177–1194 (2016)
12. Guidi, B., Michienzi, A., Rossetti, G.: Dynamic community analysis in decentralized online social networks. In: European Conference on Parallel Processing. pp. 517–528. Springer (2017)
13. Herbiet, G.J., Bouvry, P.: Sharc: Community-based partitioning for mobile ad hoc networks using neighborhood similarity. In: 2010 IEEE International Symposium on "A World of Wireless, Mobile and Multimedia Networks". pp. 1–9 (2010)
14. Hui, P., Yoneki, E., Chan, S.Y., Crowcroft, J.: Distributed community detection in delay tolerant networks. In: Proceedings of 2Nd ACM/IEEE International Workshop on Mobility in the Evolving Internet Architecture. pp. 1–8 (2007)
15. Montresor, A., Jelasity, M.: PeerSim: A scalable P2P simulator. In: Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09). pp. 99–100 (Sep 2009)
16. Palla, G., Barabási, A.L., Vicsek, T.: Quantifying social group evolution. Nature 446(7136), 664–667 (2007)
17. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Physical review E 76(3), 036106 (2007)
18. Ramaswamy, L., Gedik, B., Liu, L.: A distributed approach to node clustering in decentralized peer-to-peer networks. IEEE Transactions on Parallel and Distributed Systems 16(9), 814–829 (2005)
19. Rossetti, G., Cazabet, R.: Community discovery in dynamic networks: a survey. Tech. rep. (2017)
20. Rossetti, G., Pappalardo, L., Pedreschi, D., Giannotti, F.: Tiles: an online algorithm for community discovery in dynamic social networks. Machine Learning 106(8), 1213–1241 (2017)
21. Takaffoli, M., Sangi, F., Fagnan, J., Zaïane, O.R.: Modec-modeling and detecting evolutions of communities. In: 5th International Conference on Weblogs and Social Media (ICWSM). pp. 30–41. AAAI (2011)