A Peer-to-Peer Notification System for Distributed Online Social Networks

Michele Amoretti
University of Parma, Italy
michele.amoretti@unipr.it

Lorenzo Gandolfi
University of Parma, Italy
lorenzo.gandolfi@studenti.unipr.it

Michele Tomaiuolo
University of Parma, Italy
michele.tomaiuolo@unipr.it

***Abstract*—Current social networking systems are almost always centralized systems. This architecture poses issues about privacy, censorship and control of personal data. On the other hand, peer-to-peer systems can overcome these issues, in exchange with additional architectural complexity. This paper describes a peer-to-peer system provided with a spanning tree for distributing online notifications inside a group of interested peers. These notifications may regard discussion messages for a chat system, or any kind of update messages for spreading social activities performed by users of a Distributed Online Social Network. In particular, we describe and compare different mechanisms for the creation and management of the spanning tree.**

***Index Terms*—Distributed Systems, Online Social Network, Peer-to-Peer, Notification Systems.**

I. Introduction

Social media attracts millions of people, who in fact spend most of their online time social networking, for a variety of everyday actions. Accordingly, these social platforms are assuming different forms and aims, including distribution of news, sharing of photos and videos, direct messaging, group discussions, etc. [1]. Together with their mass spreading and also in consequence of big scandals, social platforms are also raising concern and criticism. In particular, many users are wary of privacy threats coming from other users, external entities, and also directly from the service providers.

In fact, even if the social networking systems are greatly dissimilar in their user base and functionality [2], they are almost always centralized systems, which often allow service providers to: (*i*) mine user provided data for advertisements and other purposes, (*ii*) guide their users into "walled gardens", without full control over their own information [3], [4], (*iii*) perform a-priori or a-posteriori censorship [5], (*iv*) disclose all the information they have to other entities, either motivated by selfish interests or forced under legal terms and other forms of pressure.

Conversely, in exchange with additional architectural complexity, peer-to-peer (P2P) systems essentially achieve automatic resource scalability, in the sense that the availability of resources is proportional to the number of users [6]. Moreover, without a central entity, nobody is in the position of censoring data systematically. Privacy can also be achieved, by means of key systems and cryptography. In a P2P system, if global trusted third parties are avoided, no entity has the ability to disclose a user's private key or other sensible credentials.

However, such P2P systems have to confront with the most convenient features provided by current social platforms, both in terms of functionalities and responsiveness. While some trade-offs are certainly to be considered, it is necessary to allow users of distributed social platforms to develop fluent online discussions, with notifications of new activities received with short and acceptable delays.

In this paper, we describe specifically a P2P system provided with a spanning tree for distributing online notifications inside a group of interested users. These notifications can regard discussion messages for a chat system, or any kind of update messages for spreading social activities performed by users of a Distributed Online Social Network (DOSN) [7]–[10]. In particular, we describe and compare some mechanisms for the creation and management of the spanning tree. With respect to similar approaches, our spanning tree does not rely on a specific P2P architecture, and does not require that all nodes of the underlying P2P network are fully involved in the DOSN. Any structured P2P network (such as Chord, Kademlia, or Pastry) could be used as a substrate for several partially overlapping spanning trees, each one corresponding to a specific group of users.

The manuscript is organized as follows. Section II analyzes the state of the art for P2P publish-subscribe systems. Section IV illustrates and proposes some algorithms for the creation and management of a tree structure among peers. Section V shows the results obtained by comparing the proposed different mechanisms and policies. Finally, some concluding remarks are provided in Section VI.

II. Related works

Castro *et al.* proposed Scribe [11], an application level multicast infrastructure on top of the Pastry DHT, which is used in a number of projects for peer-to-peer collaboration and dissemination of information. Scribe creates and manages multicast groups on top of Pastry. Any Scribe node can create a group, providing a group ID and some credentials to be used for access control. Other nodes can then join the group or send multicast messages, which are delivered to all members. Multicast messages are delivered by some forwarder nodes, which form a multicast tree. Forwarder nodes themselves are not required to be part of the group, instead they automatically become forwarders if they are on the Pastry route of some new member of the group, when it sends a join request.

FeedTree [12] is an RSS (Real Simple Syndication) feed distribution service based on P2P subscription mechanisms. FeedTree proposes a transition toward pushing RSS items over a P2P network, distributing the load over the nodes of a group multicast tree. For this purpose, FeedTree exploits Pastry and Scribe.

Xu *et al.* introduced Cuckoo [13] as a decentralized and socio-aware online micro-blogging service. It follows a hybrid approach consisting of: (*i*) a structured overlay network, Pastry, and a gossip protocol for disseminating micro-news among users with the same interests; and (*ii*) support for centralized dedicated services, like Twitter, which in fact still store user profiles and other data. Friend nodes help each other to balance load, thus creating a sort of virtual node. Notifications are dealt with direct push, in the case of normal users, or with gossip propagation, in the case of celebrities and broadcasters.

Perfitt & Englert proposed Megaphone [14] as a micro-blogging system, based on an optimized, trustworthy peer-to-peer network. In fact, nodes are enabled to sign and encrypt each piece of content they publish, making it verifiable and confidential for subscribers. The basic distribution mechanism is based on Scribe multicast trees. Thus, a subscriber node has to know in advance the node ID of the posters to follow, or at least it has to be able to generate it. The poster's node ID corresponds exactly to a Scribe multicast group ID. In Megaphone, the node ID is a hash of its public key, and the couple of public/private keys is generated autonomously by each node.

Messina *et al.* introduced HySoN [15], based on an overlay network of software agents, which exploits a gossip protocol. HySoN allows users to locally maintain sensitive user's data, satisfying the privacy requirements preserving sensitive data. Indeed, the properties involved in the HySoN user aggregation are inferred by local data not published in the social network.

Though some research works exist, for building a notification system exploiting the Pastry DHT, very few works try to exploit the Kademlia DHT, which is used in BitTorrent and other content sharing systems, including the Blogracy platform [8]. Matl *et al.* [16] deal with group communications in overlay networks based on the Kademlia distributed hash tables (DHT), considering three cases:

- *Anycast*, to deliver a message to any member of the group;
- *Multicast*, to deliver to all members;
- *Manycast*, to deliver to a particular subset of the group.

The article describes some abstract solutions, based on tree structures built on top of the Kademlia layer. The advantages of these structures are better exploited if the branches are balanced. Additional maintainance tasks are required to guarantee robustness and reliability, also in the case of frequent disconnections of nodes. These tasks require periodical monitoring of links and recovery mechanisms, for reconnecting the whole tree and avoiding loosing messages.

With respect to the aforementioned approaches, our spanning tree does not rely on a specific P2P architecture. Moreover, it does not require that all nodes of the underlying P2P network are fully involved in the DOSN. Structured P2P networks like Chord, Kademlia, or Pastry, could serve as a substrate for partially overlapping spanning trees, each one corresponding to a specific group of users. That is, one peer may belong to different groups at the same time.

III. Distributed social architectures

The diffusion of online social networks is opening new scenarios for envisaging novel kinds of applications, either to support new social networking activities, or to exploit established relationships among users and use them to offer higher-level services. Software agents are a natural fit for mediating access to local software- or hardware-based services, including access to data, sensors, monitors, printers and various kinds of actuators. Given their ability to negotiate and plan in a dynamic social context, software agents are also good for composing locally available services and resources, following existing trust relationships with other persons and agents located in the users proximity area. New trust relationships can also be created, on the basis of reputation and mutual acknowledgement, through the incremental and controlled exchange of profile data.

A. Autonomous agents for DOSNs

Especially in the case of completely distributed or federated social networking platforms, multi-agent systems can play an important role. Indeed, one of the very specific features of multi-agent systems is the sociality of agents, i.e., their ability to communicate in a semantic way and to develop trust relationships among them. Moreover, agents can (*i*) express their communication acts by means of acknowledged standards for interoperability among diverse systems, like FIPA; (*ii*) and exchange messages directly, in a peer-to-peer way. Therefore, it is not surprising that these two technologies are often applied together for developing advanced social platforms. In particular, multi-agent systems have been used as (*i*) an underlying layer, or middleware, for developing social networking platforms; and (*ii*) a technology to increase the autonomous and intelligent behavior of existing systems.

For the first type of solutions, many of the distinguishing features of multi-agent systems can be fully exploited. Multi-agent systems provide semantic communication among agents, which is handy for expressing all the different actions that users can perform in a social platform. The different types of messages can be understood according to their meaning and applied according to existing trust relations among the users and their respective agents. In addition, complex negotiation protocols help creating acknowledgements and trust among users, in an automatic or assisted way, without exposing sensitive data. Mobility can also be useful for moving the computation closer to data, if massive analysis is needed, but it can also be handy for adding functionality to a node of a distributed social platform or to a users client application.

In the second case, agents are mainly used because of their proactive and reactive behaviors that can provide recommendations of both users and content, and that can enable the

personalization of results. Reactive abilities are particularly important in a social networking environment where events happen continuously and users can be easily distracted by the huge information flow, which is associated with highly connected social networks [17]. Sensing the environment and executing automatic tasks can reduce this overload significantly. Goal-oriented behaviors, on the other hand, can support users in prosecuting their long term objectives about friend and content discovery, i.e., to discover known persons registered in the network, to make new acquaintances with users with common interests, to find interesting content hidden in less relevant data or from new sources.

B. Blogracy

Blogracy [8] is a distributed social networking system which uses many of the services and techniques described above, with the aim to provide adaptive and composite services on top of its core features. At the lower level, Blogracy uses widespread and stable peer-to-peer technologies, such as distributed hash tables and the BitTorrent protocol, for coping with the intrinsic defects of centralized architectures and to become the basis of solid distributed social networking platforms. At the higher level, it takes advantage of multi-agent systems for simplifying the implementation of social network services in a decentralized setting.

The architecture of the application is modular and composed of two basic components: (*i*) an underlying module for basic file sharing and DHT operations, built as an extension of existing implementations, and (*ii*) an OpenSocial container, i.e., a module providing the services of the social platform to the local user through a Web interface. Additionally, the system supports autonomous agents for providing (*i*) recommendations of both users and content, (*ii*) personalization of results, and (*iii*) trust negotiation mechanisms.

The Blogracy system itself relies only on users nodes for its operation and users need to perform background tasks on their own, in a distributed way. A layer of autonomous agents takes charge of assisting the user in finding new interesting content and connections, and in pushing the local users activities to followers.

IV. Designed algorithms

As demonstrated by the systems described is Section II, a practical approach for implementing distributed publish-subscribe systems is to organize a group of peers in a tree-like structure. In this way, each node has the duty to forward messages to a limited number of intermediate destination nodes, which are directly linked to it. Section V will discuss some guidelines for the organization and functioning of these trees, obtained through simulations of various algorithms and configurations, which are introduced in this section.

Since P2P systems have to scale to a very large number of nodes, as a first feature to configure it is necessary to choose the degree of nodes, *i.e.*, the maximum allowed number of children nodes, for each parent node. The aim is to obtain the best performance, without creating excessive burden for each involved node. Higher values of this parameter lead to less deep trees, but increase the number of messages to forward at each step.

A. Group Join

A node that intends to join a group, and thus its associated logical tree, has to find a node of the tree and send a join request to that node. There are various possibilities for performing both steps of the process. A connection point is a node to which a join request can be sent. We suppose that each node participating to a group, registers under the group identifier into the P2P network. As described by Matl *et al.* [16] with reference to the Kademlia network, it is possible for a new node to contact some other node, already in the group, without finding and contacting the root node. In particular, we have defined the following two connection policies: (*i*) the join request is only sent to the root node (*root* strategy); (*ii*) the join request is sent directly to the first node found, which already participates to the tree (*first* strategy). In the first case, the join request is sent only after finding the root node. Sending the request to the root node can lead to a more balanced tree, under some conditions. In the second case, the join request is sent more easily as soon as any node of the tree is found. In this way, the workload on the root node is reduced, consequently removing a possible bottleneck.

B. Connection

After a node n has been chosen as an entry point to the group, a join request is sent to it, for being accepted as a new child node. Node n checks if it can accept one more child node, according to the node degree k of the tree (*i.e.*, the maximum number of children each node may have). If the answer is positive, the connection is successful. Else, if n does not have room for one more child node, it is necessary to find another possible entry point. In fact, in its refusal answer, n also inserts a reference to an alternative connection point, which is chosen among its own children. Simple possible policies for this choice include: (*i*) the minimun XOR distance between the chosen node and the new one, similarly to the other protocols of a Kademlia network; or (*ii*) a random selection, which may be a simplistic approach, but could nevertheless provide surprisingly good results in some cases. However, also the new connection point can be unavailable, thus the process goes on iteratively, till finding a suitable connection point.

Algorithms 1 and 2 illustrate the procedures for sending and handling join requests, respectively.

Algorithm 1 Join request, followed by acceptance or refusal.

Require: $a\ reference\ to\ a\ node\ of\ the\ tree$
  $response = reference.ConnectionRequest()$
  while $not\ response.connectionAccepted()$ **do**
    $reference = response.getAlternativeNode()$
    $response = reference.ConnectionRequest()$
  end while

Algorithm 2 Management of a join request. *numChildren* represents the children count, *maxChildren* is the maximum degree set for the tree.

Require: $request\ from\ node\ n$
Ensure: $request\ response$
  if $numChildren < maxChildren$ **then**
    $buildConnection()$
    $numChildren = numChildren + 1$
    $return\ ConnectionAccepted$
  else
    $response = ConnectionRefused$
    $response.setAlternativeNode()$
    $return\ response$
  end if

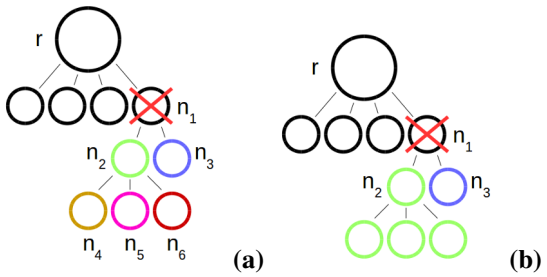

Fig. 1. Subtree breakout (a) and subtree preservation (b) algorithms. In subtree breakout, nodes n_2,n_3,n_4,n_5 and n_6 try to reconnect autonomously to the tree, after n_1 fails. In subtree preservation, the structures of subtrees below n_2 and n_3 are unaltered after n_1 fails.

C. Tree Reconstruction

Once the tree structure has been established, it is necessary to ensure its maintainance. In fact, in a P2P environment, each element can suddenly disconnect or disappear. If a node of the tree fails, some mechanisms need to be in place to assure the reachability of all remaining nodes, including the children and descentants of the failed node.

First of all, some mechanisms need to be adopted to periodically check connections. This can be enforced in practice by sending periodical *ping* requests from each node to its own direct neighbours. If a *ping* request is not answered before a timeout, the node has to be supposed to be missing.

A disconnection can lead to two fundamental problems. If a leaf node disconnects, the problem is limited and it is sufficient to remove its link with the parent node. Instead, if a node with children diconnects, an additional problem is represented by the reachability of the children and descendants; in fact, it is necessary to reconnect all those nodes to the main tree, *i.e.*, to reconstruct the tree. To solve the problem of the reconstruction of a tree, we have devised various algorithms.

The *subtree breakout* algorithm is the simplest procedure, from the logical point of view. It simply consists in assigning to each node in the broken branch the duty to reconnect to the tree, individually. A node that finds its own parent to be disconnected, tells its children to find a new entry point. It then removes all its own links and autonomously tries to reconnect to the tree. Each child and descendant acts in the same way, till all nodes are reconnected.

The *subtree preservation* algorithm is more conservative with respect to the broken branches, after their parent node disconnects. In fact, it is based on a mechanism of reconnection, in which the topmost node is assigned the responsibility to reconnect to the main tree, possibly without affecting its descendants.

Figure 1 shows the two cases. Case (a) represents the subtree breakout algorithm. Node n_1 fails; the different colors of the descendants of n_1 indicate a fragmentation of the substructure, after which each node reconnects autonomously. Case (b) is related to the subtree preservation algorithm. After node n_1 fails, the structure of its former branches (colored in green and blue) remains unaltered till the new connection. Only nodes n_2 and n_3 are involved in finding a new connection point to the main tree.

The last recovery algorithm we propose is denoted as *recursive election*. The previous algorithms do not guarantee to solve the problem of the disconnection of the root node. This particular case can be solved with an election of a substitute node among the former children of the disconnected root. This election can be performed efficiently with the *bully* algorithm [18]. For the election, we have considered two different policies:

- *distance*, the elected node is the one with the closest identifier to the group identifier, according to the XOR distance;
- *lifetime*, the elected node is the one which has been connected for the longest time, thus coherently assigning a more important role to the more reliable and continuous nodes.

To complete the election process, it is necessary that each child maintains a reference to all its own siblings. Such references have to be kept fresh and constantly updated. Once a new root node is elected, it takes charge of all its former siblings. Instead, its own previous children may have to reconnect to the tree, if their parent node is no more able to keep them, according to the maximum allowed degree for the tree.

For their reconnection, these nodes can break or maintain the structure of their own branch, according to one of the policies described above. However, an alternative solution is to apply the same mechanism described to replace the root node, with an election among siblings according to the bully algorithm. And similarly, this approach can be applied recursively at each level of the disconnected branch. In this case, each node, at each level of the structure, has to keep references to all its own siblings. Moreover, since each node could have to substitute its own parent, in case of being elected, it has to keep a reference to its parent and to the node immediately above it, in the tree.

Figure 2 represents some executions of the recursive election algorithm after a node disconnects. To ease the representation, the node degree k is supposed to be 3.

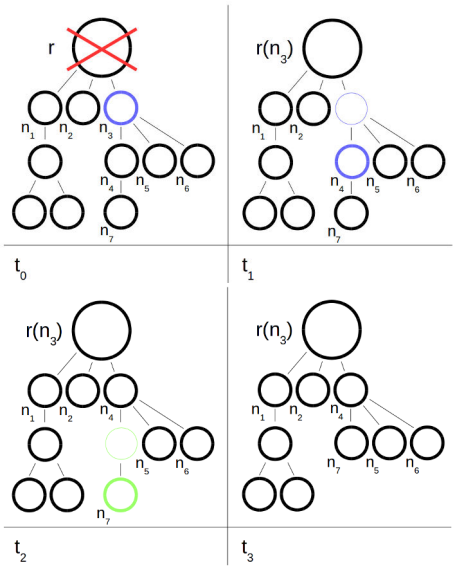

Fig. 2. Executions of the *recursive election* algorithm. At time t_0 the root node *r* fails. Node n_3 is elected as a substitute. At time t_1, node n_3 is no more available for its previous role and must be substitued. Node n_4 is elected as substitute. At time t_2, node n_4 is no more available and node n_7 substitutes it. No election is required in this case. At time t_3, the structure is completely reorganized, with all nodes connected.

V. Simulations

To evaluate the proposed algorithms, we used DEUS, general-purpose discrete event simulation environment [19], which is available as open source [20]. DEUS enables the simulation of large and highly dynamic networks, with the desired detail level. DEUS is particularly suitable to study P2P architectures, focusing on overlay protocols [21]–[23].

Tree construction algorithms are compared in terms of

- workload ditribution on network nodes,
- quickness,
- communication robustness.

To this purpose, the following performance indicators are taken into account.

1) Number of control messages ν.: Tree construction and maintenance require that nodes exchange control messages. In our simulations, each node has its own counter ν, which is incremented by 1 every time a control message is delivered to the node. In this way, it is possible to characterize the amount of network traffic both locally and globally. A large total number of control messages implies high consumption of network bandwidth, and poor user experience due to delayed tree construction and maintenance.

2) Tree depth δ.: Tree depth, defined as the maximum distance between the root node and any leaf node, is a very important metric. Given two trees with N nodes each, but different depths $\delta_1 < \delta_2$, the one with depth δ_1 is more balanced than the one with depth δ_2. Higher balancing is preferable, as it means reduced total delays and better parallelism.

3) Propagation delay π.: According to a widely used approach [16], the communication delay between any two nodes A and B is defined as $\tau = \tau_A + \tau_B$. Each node's contribution is a continuous random variable with uniform distribution in the interval $[\tau_{min}, \tau_{max}]$. In our simulations, we adopted the following values: $\tau_{min} = 10$ [ms], $\tau_{max} = 20$ [ms]. We define the propagation delay π as the time that is necessary for the message to reach all nodes in the tree.

A. Group Join

To analyze the group join strategies denoted as *root* and *first*, illustrated in Section IV, we have compared the performance indicators defined above, by taking into account different values for the node degree k.

In a first group of simulations, whose results are illustrated in Figure 3, a tree with 4000 nodes has been constructed using the *root* strategy and $k \in \{2, 4, 8, 16, 32, 64\}$. Then, the same experiment has been performed using the *first* strategy. For each value of parameter k, the simulation has been repeated 10 times, with different pseudorandom number generation seeds.

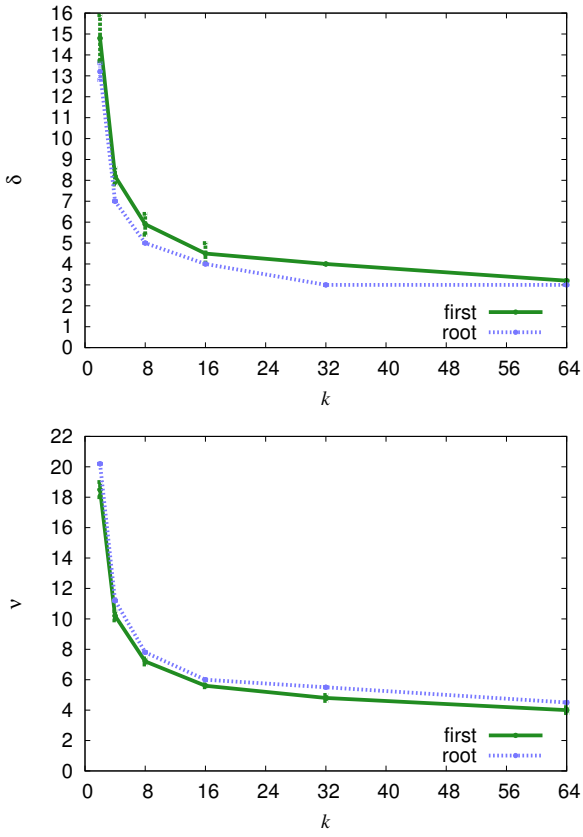

Fig. 3. Tree depth δ (left) and number of control messages ν per node (right), for increasing values of k, comparing the *root* and *first* strategies for group join.

The variation of number of requests and tree depth with respect to k is evident. The higher k, the higher the balancing of the tree and the efficience in terms of message traffic. It is also worth noting than this behavior is more accentuated with the *first* strategy. The *root* strategy produces slightly lower depth values, with respect to the *first* strategy, for any k value.

Taking into account all these aspects, the *first* strategy is better than the *root* one. Thus, in the following, all presented results are those regarding the *first* strategy.

In Figure 4, the propagation delay as a function of k is reported. Also for this performance indicator, the higher variation is achieved for low k values, up to $k = 8$. Thus, considering all the performance indicators, the best tradeoff between performance and complexity is $k = 8$.

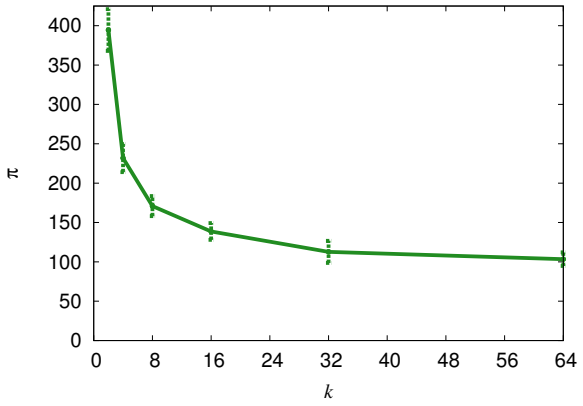

Fig. 4. Propagation delay π for increasing values of the node degree k, considering the *first* strategy for group join.

B. Connection

When a node receives a connection request, there may be two different scenarios. In the first one, the node is able to accept a new child and acknowledges the connecting node. In the second scenario, the node has already k children, thus cannot accept a new one but can suggest another parent to the connecting node, according to either the *distance* or *random* strategy (described in Section IV).

To evaluate the performance of the aforementioned strategies, we performed 50 simulations for each one, considering the construction of a tree with 4000 nodes, with node degree $k = 8$ and *first* strategy or group join. The results reported in Table I show that the *random* connection strategy is slightly better that the *distance* one.

TABLE I
PERFORMANCE EVALUATION OF THE CONNECTION STRATEGIES.

strategy	δ	ν
distance	6.93	7.68
random	6.00	7.20

C. Tree Reconstruction

The strategies for tree reconstruction, illustrated in Section IV, are *subtree breakout*, *subtree preservation* and *recursive election*. Their performance has been evaluated with respect to the disconnection of different groups of nodes, which are 1%, 5%, 10%, 20% and 50% of the total number of nodes in the tree, respectively. We considered a tree with 4000 nodes, $k = 8$, *first* group join strategy and *random* connection strategy. We recall that for the strategy that imply node election after a root node failure, the XOR distance has been used as the winner selection metric. More specifically, the node with lower distance is the one that gets selected.

TABLE II
EFFECTS OF THE *subtree breakout* STRATEGY, FOR INCREASING VALUES OF THE NODE DEGREE k. THE FIRST ROW SHOWS THE δ AND ν VALUES BEFORE THE NODES FAIL.

% of failed nodes	δ	variation	ν	variation
-	6.0	-	7.1	-
1%	6.0	0%	7.5	4.3%
5%	5.9	-1.5%	9.4	31.1%
10%	5.8	-2.8%	11.5	60.9%
20%	5.8	-2.8%	15.6	117.3%
50%	5.4	-9.0%	37.9	427.9%

TABLE III
EFFECTS OF THE *subtree preservation* STRATEGY, FOR INCREASING VALUES OF THE NODE DEGREE k. THE FIRST ROW SHOWS THE δ AND ν VALUES BEFORE THE NODES FAIL.

% of failed nodes	δ	variation	ν	variation
-	6.0	-	7.1	-
1%	6.4	8.1%	7.3	1.8%
5%	7.5	26.2%	7.6	6.1%
10%	7.0	18.0%	8.1	13.7%
20%	8.1	36.0%	9.9	38.6%
50%	8.6	44.2%	18.9	163.7%

TABLE IV
EFFECTS OF THE *recursive election* STRATEGY, FOR INCREASING VALUES OF THE NODE DEGREE k. THE FIRST ROW SHOWS THE δ AND ν VALUES BEFORE THE NODES FAIL.

% of failed nodes	δ	variation	ν	variation
-	6.0	-	7.1	-
1%	6.0	0%	7.2	1.3%
5%	6.0	0%	7.6	5.8%
10%	6.0	0%	8.0	12.3%
20%	6.0	0%	9.3	29.5%
50%	5.9	-1.6%	14.7	105.1%

The results related to the *subtree breakout* strategy are illustrated in Table II. We can observe that tree depth remains small even for the largest group of failed nodes. However, the number of requests increases too much (427% when 50% nodes fail).

The results related to the *subtree preservation* strategy are presented in Table III. With respect to *subtree breakout*, the tree depth increases considerably (by 44.26% in the worst cade). The reason is that not breaking the tree may cause a branch to be reconnected to a node that is already very deep in the tree. Fortunately, the increase of network traffic is lower (163.7% in the worst case).

The *recursive election* strategy is a compromise between the previous ones. This is confirmed by the results reported in Table IV. Facing a node group failure, tree depth slightly changes and the number of requests has a very limited increase. For these reasons, the *recursive election* strategy has to be preferred.

VI. Conclusion

In this paper, we described and compared a number of mechanisms for creating and managing a spanning tree, hosted by a generic structured P2P network. The adopted decentraized approach is motivated by the need to solve issues about privacy, censorship and control of personal data in DOSNs.

According to our simulations, tree robustness is guaranteed by the following mix of strategies: *first* for group join, *random* for connection, and *recursive election* for tree reconstruction in case of node failures.

Regarding future work, we plan to implement the proposed algorithms in the Blogracy platform [8] and test them over the PlanetLab facility. Furthermore, we plan to improve the agorithms by means of adaptive strategies, *e.g.*, for online tuning of the node degree k.

References

[1] G. Angiani, P. Fornacciari, M. Mordonini, M. Tomaiuolo, and E. Iotti, "Models of participation in social networks," in *Social Media Performance Evaluation and Success Measurements*. IGI Global, 2016, pp. 196–224.

[2] E. Franchi, A. Poggi, and M. Tomaiuolo, "Social media for online collaboration in firms and organizations," *International Journal of Information System Modeling and Design*, vol. 7, no. 1, pp. 18–31, 2016.

[3] S. Shankland, "Facebook blocks contact exporting tool," *Retrieved January 26, 2014*, 2010. [Online]. Available: http://news.cnet.com/8301-30685_3-20076774-264/facebook-blocks-contact-exporting-tool/

[4] T. Berners-Lee, "Long live the web," *Scientific American*, vol. 303, no. 6, pp. 80–85, 2010. [Online]. Available: http://www.scientificamerican.com/article.cfm?id=long-live-the-web

[5] A. D. Salve, P. Mori, and L. Ricci, "A survey on privacy in decentralized online social networks," *Computer Science Review*, vol. 27, pp. 154–176, 2018.

[6] A. Poggi and M. Tomaiuolo, "A dht-based multi-agent system for semantic information sharing," *Studies in Computational Intelligence*, vol. 439, pp. 197–213, 2013.

[7] B. Guidi, T. Amft, A. De Salve, K. Graffi, and L. Ricci, "DiDuSoNet: A P2P architecture for distributed Dunbar-based social networks," *Peer-to-Peer Networking and Applications*, pp. 1–18, 2015.

[8] E. Franchi, A. Poggi, and M. Tomaiuolo, "Blogracy: A peer-to-peer social network," *International Journal of Distributed Systems and Technologies (IJDST)*, vol. 7, no. 2, pp. 37–56, 2016.

[15] F. Messina, G. Pappalardo, D. Rosaci, C. Santoro, and G. Sarné, "Hyson: A distributed agent-based protocol for group formation in online social networks," in *Multiagent System Technologies (MATES 2013), Lecture Notes in Computer Science*, 2013, pp. 320–330.

[9] B. Guidi, A. Michienzi, and G. Rossetti, "Dynamic community analysis in decentralized online social networks," in *Euro-Par 2017: Parallel Processing Workshops*, 2017, pp. 517–528.

[10] M. Amoretti, A. Ferrari, P. Fornacciari, M. Mordonini, F. Rosi, and M. Tomaiuolo, "Local-first algorithms for community detection," in *CEUR Workshop Proceedings 1748, KDWeb2016*, 2016.

[11] M. Castro, P. Druschel, A. Kermarrec, and A. I. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in communications*, vol. 20, no. 8, pp. 1489–1499, 2002.

[12] D. Sandler, A. Mislove, A. Post, and P. Druschel, "Feedtree: Sharing web micronews with peer-to-peer event notification," in *P2P Systems IV, ser. Lecture Notes in Computer Science*, vol. 340. Springer, 2005, pp. 141–151.

[13] T. Xu, Y. Chen, X. Fu, and P. Hui, "Twittering by cuckoo: decentralized and socio-aware online microblogging services," in *ACM SIGCOMM Computer Communication Review*. ACM, 2010, pp. 473–474.

[14] T. Perfitt and B. Englert, "Megaphone: fault tolerant, scalable, and trustworthy p2p microblogging," in *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*. IEEE, 2010, pp. 469–477.

[16] L. Matl, T. Cerny, and M. J. Donahoo, "Effective Manycast Messaging for Kademlia Network," in *Proc. of the 30th Annual ACM Symposium on Applied Computing (SAC '15)*, 2015, pp. 646–652.

[17] G. Lombardo, P. Fornacciari, M. Mordonini, M. Tomaiuolo, and A. Poggi, "A multi-agent architecture for data analysis," *Future Internet*, vol. 11, no. 2, 2019.

[18] H. Garcia-Molina, "Elections in a distributed computer system," *IEEE Trans. on Computers*, vol. C-31, no. 2, pp. 48–59, 1982.

[19] M. Amoretti, M. Picone, F. Zanichelli, and G. Ferrari, "Simulating mobile and distributed systems with DEUS and ns-3," in *Proc. of the 2013 International Conference on High Performance Computing and Simulation (HPCS 2013)*, 2013, pp. 107–114.

[20] M. Amoretti, "DEUS on GitHub," cited June 2019. [Online]. Available: https://github.com/dsg-unipr/deus

[21] M. Martalò, M. Picone, M. Amoretti, G. Ferrari, and R. Raheli, "Randomized network coding in distributed storage systems with layered overlay," in *Proc. of the 2011 Information Theory and Applications Workshop (ITA 2011)*, 2011, pp. 324–330.

[22] M. Picone, M. Amoretti, and F. Zanichelli, "Evaluating the robustness of the DGT approach for smartphone-based vehicular networks," in *Proc. of the 36th Annual IEEE Conference on Local Computer Networks (LCN 2011)*, 2011, pp. 820–826.

[23] M. Amoretti, A. L. Lafuente, and S. Sebastio, "A cooperative approach for distributed task execution in autonomic clouds," in *Proc. of the 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2013)*, 2013, pp. 274–281.