Efficient community-based data distribution over multicast trees

Stavros Souravlas, *Member, IEEE,* and Angelo Sifaleras

***Abstract*—Communities are important attributes of today's networking, since people who join networks tend to join communities. Thus, community detection is an important issue in designing algorithms for Delay Tolerant Networks (DTN). Multicasting is an appropriate method to share information within a community or between communities, because it delivers messages from a source to a group of targets using limited resources. This work addresses the problem of detecting communities in weighted networks with irregular topologies. Based on communities, we propose an efficient data distribution algorithm for DTNs. The distribution strategy is based on the construction of a number of multicast trees, where each tree can be used to select the best relay node for each target to improve multicast efficiency. The proposed strategy provides better delivery ratio compared to other strategies and it reduces latencies when multiple nodes from a community need to multicast to other community members.**

***Index Terms*—Social Intelligence, Social Systems Simulation, Community Detection, Multicast Trees**

I. Introduction

Several systems of high interest to the scientific community can be represented as networks. Network examples are the Internet, the World-Wide Web, and the social networks [1]. Perhaps the latest are the type of networks in which researchers are most interested nowadays. Examples of social networks are Facebook (1.6 billion users), Instagram (400 million users), or Twitter (320 million users) [2]. Typically, the social networks are organized into groups of users [3]. These users join a network, create their own profile, publish information and find other users with the same interests. In this way, smaller or larger groups of users are formed within networks. Such groups are referred to as *communities*. Although there is no universally acceptable definition of a community, one can define it as a set of nodes and links in a network, such that its internal connections are more than its external connections [4]. The nodes of a community are considered similar to each other, dissimilar to the other nodes of the network [5] and represent its users. The edges represent the connection between the users of one community or between users of different communities.

In DTNs, communication between the mobile devices is performed in an opportunistic way. Due to their mobility, the network nodes can be used as relays that forward data in an opportunistic manner towards the targets. When two nodes are found within each other's range, they communicate directly. When their contact is lost, the data to be delivered needs to be stored and forwarded by intermediate relays. Therefore, the main problem in distributing data across a DTN is how to select the appropriate relays to make the distribution of data as efficient as possible. Many researchers have been working on constructing efficient algorithms for DTNs [6], [7], [8], [9]. The focus of these papers is mainly to minimize the total time and to make their strategies energy-efficient. Since mobile nodes are often powered by short-duration batteries that (in most cases) can't be replaced immediately when they run out of energy, the energy issue is rather important. A good performance measure is the routing path length, that is, the number of hops from source to destination. A short routing path clearly results in shortest delays.

Since social relations among mobile users are more likely to be long-term characteristics compared to mobility, forwarding schemes based on social concepts tend to become the dominating approach [10]. Some algorithms that study the social factors [8], [11] and exploit the fact that people with shared social features tend to form communities, involve multicasting. In multicasting, a message holder forwards a message to multiple destinations. To reduce the overhead and forwarding cost, the destinations will share the routing path until a point that they have to be separated. This usually defines a tree structure [12]. Tree structures generally cover a variety of applications in networking and grid computing, (e.g., broadcasting, multicasting, and data replication [13]). In multicasting, tree structures have been proposed and widely used in a wide range of disciplines, including medicine, distance learning, and intelligent systems [9].

Several works have addressed the problem of creating multicast trees for DTNs. Most of them have used geographic information [9], [14], [15], [16] to design algorithms that construct minimum length multicast trees for DTNs. Some of their goals are queue stability [17], low complexity and energy efficiency and bandwidth guarantee issues [18].

In this article, we first consider the problem of community detection over a weighted network. Afterwards, we develop a scheme to create multiple multicast trees, which are then used to distribute the data in an efficient way.

The remaining of this work is organized as follows: Section II summarizes the related work. Section III presents our community detection strategy and its complexity analysis. In Section IV, we show how communities are used to construct multicast trees to distribute the data packets over the network. In Section V we show how we distribute data using the multicast trees generated by our scheme. Simulation results and discussion are presented in Section VI. Section VII concludes the paper and offers aspects for future work.

Stavros Souravlas and Angelo Sifaleras are with the Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece.
Email: sourstav, sifalera@uom.gr

II. Related Work

Recently, a lot of researchers have focused their attention on the problem of community detection in social and data networks. Communities give a more realistic approach of the networks and they can be used to relay messages within a network [10], [19], [20], [21]. According to the majority of the papers found in the literature, common experience shows that communities really exist in social and data networks [3], [22]. The study of community structures is closely related to the problem of network partitioning, which is typically defined as the division of a network into a set of groups of approximately equal sizes with minimum number of edges [5], [23], [24]. This problem is NP-hard and thus heuristics and approximation algorithms have to be employed [25]. In this section, we present the related work divided into two categories: (a) Community detection algorithms, which will be further divided into 4 classes based on the way communities are identified, and (b) Multicasting algorithms.

A. Community Detection Algorithms

Most of the papers found in the literature agree that the members of a community are more strongly connected between them than they are connected to members of other communities. These papers introduced a number of algorithms to identify communities and a number of measures for testing and comparisons [23], [26], [27]. Members in the same community share the same interests, viewpoints, preferences, hobbies, professions, newsgroups, and, as far as social networks are concerned, they are more likely to comment or "like" topics of the same interest. In most of the papers, this whole activity is modeled via analyzing the link weights [5], [10], [18], [28], [29], [30] of the network. In order for a node to be considered as a member of a community, its internal/external links to this community should have some well-defined properties.

A category of algorithms also detects *overlapping* communities [18], [31], [28], [32], [33], i.e., communities with members in common. Generally, these strategies start from some formed community structures and try to spot nodes that may reside in multiple communities, based on pre-imposed criteria. If such nodes exist then, the communities to which they belong are said to be overlapped. In the remaining of this subsection, we present the most important of the existing overlapping community detection schemes, grouped into three classes, based on the detection method (see Xie et al [31]): Clique percolation, local expansion and optimization, dynamic algorithms.

Clique Percolation

The Clique Percolation Method (CPM) assumes that a community consists of fully connected subgraphs. Sets of such subgraphs may overlap. The community detection is based on searching and identifying neighboring cliques. Initially, it finds all cliques in the network, which are then represented in the graph by a vertex. If two cliques share a predefined number of members, then their corresponding vetrices are connected. Thus, connected vertices on the graph represent network communities.

Another interesting clique percolation technique was introduced by Farkas et al. [34]. In this algorithm, a predefined intensity threshold was introduced, and only the cliques with intensity higher than the threshold get a community membership.

In [35], the community detection process is divided into phases: In the first phase, cliques of $k-1$ members are detected by checking all cliques of $k-2$ members, residing in the neighborhood of two vertices. Then, the connected components of the $k-1$ cliques are checked, in order to form a community.

Line Graph and Link Partitioning

This group of algorithms partitions links instead of nodes to detect a community. When the links connected to a node are found in more than one communities, this node is assumed to be overlapped. Chen et. al [18], calculate the node strength for each node at first, and then detect an initial community from the node with the largest node strength (the sum of weights of all the edges connected to the node). Then, the "strongest" node is selected and its belonging degree (a measure based on the coefficients of links) is measured against a threshold. The node belongs to a community if its belonging degree is less than the threshold. Apparently, the node's links may be connected to a number of communities, to which the node itself may belong to, according to its belonging degree and its threshold value.

A similar approach is found in [33], where the authors present a strategy that can detect both overlapping and non-overlapping communities and adds one node to a community in every *expanding step*. Specifically, each expanding step computes the *belonging degree* of the nodes to a community C and then, they select the one with the highest belonging degree. This node is temporarily attached to C, forming $C^{'}$. Then, if the *conductance* of $C^{'}$ is lower compared to the conductance of C, the selected node is finally attached to C.

Ahn et. al [36] use the metric of Jaccard index to compute the similarity for any given pair of links connected to a node. Based on similarities to build a link dendrogram, which is then cut at some threshold to produce the communities.

Local Expansion and Optimization

This class of algorithms is based on expanding a partial community. Normally, they use a function that characterizes the quality of a interconnected node group. In [32], the authors assume that communities are essentially local structures, involving the nodes belonging to the modules themselves plus at most an extended neighborhood of them. The community is a subgraph identified by the maximization of the nodes fitness. This measure is based on the total internal and external degrees of the nodes of a group (or module). The internal degree of a module is the double of the number of internal links of the module. The external degree is the number of links connecting each member of the module to the rest of the graph. The aim is to find a subgraph starting from a specified node, such that the inclusion of a new node, or the elimination of one node from the subgraph would lower the fitness value (optimization problem).

The Overlapping Community Algorithm (OCA) algorithm [37] is based on the idea of mapping each node to a multi-dimensional vector. Each node subset is then defined as the sum of individual vectors in this set. The fitness function is defined as the directed Laplacian on function O, where O is the squared Euclidean length of a subset vector. The algorithm tries to remove or add a node that results in maximizing the value of the tness function.

The intrinsic Longitudinal Community Detection (iLCD) algorithm [38] operates as follows: given a set of edges created at some time step, it updates communities by adding a new node if its number of neighbors is greater than predefined expected values. A merging procedure is used to improve the detection quality if the similarity is high. The similarity between two communities is the ratio of their common nodes.

Dynamic Algorithms In dynamic algorithms, the observed dynamic behaviors are taken into account and no predefined objective functions are used. An interesting approach was presented by Raghavan et al., [5]. This strategy uses a simple label propagation algorithm. Unlike most of the strategies found in the literature this strategy does not use a predefined objective function (like the conductance mentioned in the previous paragraph) to identify the communities. Every node is initialized with a unique label and at every step each node adopts the label that most of its neighbors currently have.

Gregory [28] proposed an extension of the label propagation technique of Raghavan et al., [5] named Community Overlap PRopagation Algorithm (COPRA), in order to be able to detect overlapping communities. Specifically, the label and propagation steps are extended to include information about more than one community, thus each vertex can now belong to more communities. The network vertices that represent cliques have labels that propagate between neighboring vertices so that members of a community are aware of their community membership.

B. *Multicast Algorithms*

The problem of multicast tree construction is also well-studied. Generally, most of the algorithms developed that have been designed for general DTNs do not take into account the social factors, with a few exceptions that will be discussed at the end of this paragraph. The main concern of the majority of the proposed strategies is to have minimum data paths, achieve energy efficiency and queue stability. In [9], the authors designed the Toward Source Tree (TST) algorithm, to build multicast trees in WSNs. The algorithm includes 3 phases: (1) Receiver's identification: A message containing all receivers is sent from the source so that all nodes know if they are to receive data, (2) Receiver's connection: A "temporary tree" is built, consisting of the multicast group members and then the shortest path between all pairs of members that are directly connected in the "temporary tree" is defined, and (3) Cycle elimination: When nodes are added to the temporary tree as relays, cycles can be formed. This phase eliminates these cycles. The running time of TST is $O(\sqrt{n \log n})$, where n is the number of nodes in total, and it is proven to be energy efficient.

In [16], the proposed algorithm takes into account the tree cost, the communication delay, and Demand Response (DR) capability (demand-supply balancing capability through manipulation of the demand side) in each destination. The scheme ensures that the total cost of multicast tree construction lies within an acceptable range and improves the effect of DR capability giving priorities to destinations with larger DR capability.

In [7], the authors develop a a multicast routing protocol that constructs multiple multicast trees and uses network coding. Each multicast tree can satisfy a predefined percentage of the bandwidth requirement. The proposed protocol can reduce the total bandwidth consumption while it provides bandwidth guarantees for any work flow. Due to network coding, no redundant packet is generated, and the need for a scheduling algorithm for packet distribution among the trees is eliminated.

Not many papers have focused on multicasting with social criteria: In [12], the authors rely on the dynamic social features to capture nodes' dynamic contact behavior, exploit the social relationships among nodes, and use communities to select the best relay node for each destination in each hop to improve multicast efficiency. Two multicast algorithms are proposed: The Community and Social feature-based multicast involving Destination Only (multi-CSDO), which involves only the destination nodes in community detection, and the Community and Social feature-based multicast involving Destination nodes and the Relay candidates (multi-CSDR), which involves both the destination nodes and the relay candidates in community detection. The algorithm is proven efficient in terms of delivery rate, latency and number of forwardings. Deng et al. [8] proposed a social profile-based multicast routing scheme (SPM). Nodes with a small average affiliation distance or large common language ratio compared to a multicast group are selected as the relay nodes. These two metrics are easily obtainable by all network nodes. Xu et al. [11] extended this work by introducing dynamic social features for capturing the nodes' behavior (Multi-Sosim algorithm).

In this work, we initially propose a scheme which detects communities based on multiple similarity paths whose weights are as large as possible. Then, these paths are used to construct multicast trees. The multicast trees are then using these paths for data distribution from a node to a set of other nodes. Thus, the data distribution is *performed with social criteria*. The main contributions of this work are the following:

- It introduces a novel method for community detection using a depth-first based search over the network.
- Based on community detection, it constructs multiple multicast trees based on social features.
- It uses the multicast trees to perform efficient data distribution across the network.
- Data distribution is based on social criteria.

III. Community Detection

Consider a small network of irregular topology, such as the one shown in Fig.1, where there are already three communities

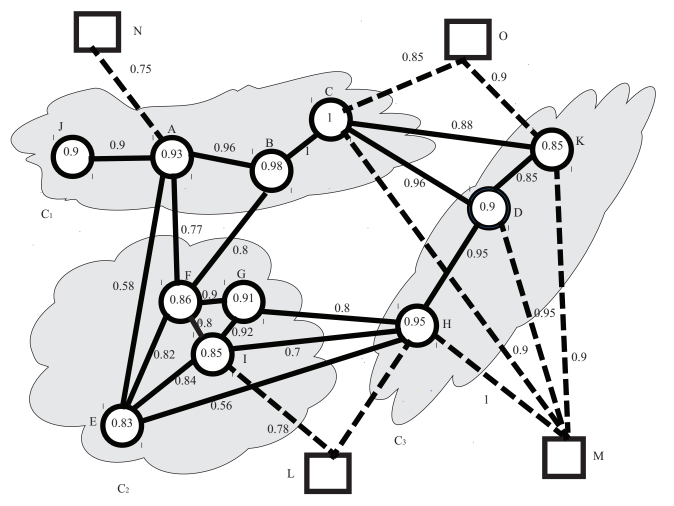

Fig. 1. An irregular network with 3 communities and 4 new-coming nodes

C_1, C_2, and C_3 and four newly-inserted nodes L,M,N,O (depicted as squares), that are out of C_1, C_2, and C_3. Fig.1 reveals some aspects than need to be considered:

1) User B is directly connected to user F that belongs to C_2. Thus, there is a chance that B also belongs to C_2.
2) User C is connected to C_2 via B. Thus, there is a chance that C belongs to C_2 as well. In this case, the relay B may also belong to C_2.
3) Continuing Topic 2, if there is a high percentage of overlapping between two communities, chances are that they are actually one community.
4) Not all users of a network are necessarily related (e.g., two Facebook users although they are not related, they may have common friends). In Fig. 1, C is not directly connected to F, but it is connected to B. In its turn, B is connected to F. To continue with the Facebook analogy, this is a case that a user (for example C) is unaware of a community C_2 until he/she visits the page of another user that has some relationship to members of this community (like B). Thus, a way to quantify the *similarities* between unrelated users, like C and F is required.
5) Newly connected users like L,M,N,O should be observed and, based on their behavior, classified as members of one or more communities.

Let $G = (V, E)$ be a weighted, undirected graph, where V and E are the sets of nodes and edges, respectively. Nodes represent users and edges represent the connection between two users (e.g., friendship in Facebook). The *similarity* $w_{i,j}$ between users i and j is the weight of the edge that connects i and j. This value lies in the interval $[0 \ldots 1]$. For example, a value of 0.78 shows that the likes or opinions of i and j can be considered quite converging, at a percentage of 78%. Bu et al. [3] present a table of user phrases that indicate supportive or opposing attitude towards a comment or opinion, etc. These phrases are accompanied by a value that shows the degree of support.

The network nodes are also weighted: the weight of a node i indicates its *Network Connectivity Degree* (NCD), i.e., how well the preferences, likes, views of a user are fitted to a community. The network connectivity degree is also between $[0 \ldots 1]$. The letters are the node names, the edge values indicate view similarities and the node values indicate similarity degrees. The network connectivity degree of a user i, NCD_i, is computed as follows:

$$NCD_i = \frac{\sum w_{i,j}}{n} \tag{1}$$

where $w_{i,j}$ is the weight of any edge that connects user i with any user j that lies in the same community and n is the number of such edges. For example, consider community C_3. User D has two internal links, namely with users H and K and two external links, with users C and M. To compute $NCD_{\mathtt{D}}$, we only consider the internal links and we have $NCD_{\mathtt{D}} = \frac{(0.95+0.85)}{2} = 0.9$. This value is stored in node D.

Our community detection strategy includes two phases: (A) Computing the similarities between unrelated users, (B) Determining overlaps based on the values found in Phase A. Having determined the overlaps, we use an update method that handles newly-incoming nodes. In the following subsections, we discuss these two phases of community detection and the updating procedure.

A. Computing the Similarities Between Unrelated Users

According to the aspects discussed in the beginning of this section, it is clear that a user can be connected to a community, either *directly*, through a relationship with one or more users of the community, or *indirectly*, through a relationship with a user that is related to a member of the community. The similarities are obtained by observing the users behavior and quantifying them, based on the strategy provided by Bu et al. [3]. Since we know the similarities between related users, it is important to quantify the similarities between unrelated users, (i.e., not directly connected users of different communities). These similarities are necessary to identify if users that appear to have no relationship between them can have a membership in the same community. Our approach suggests that, the similarity of two unconnected users is higher when the product of weights on the links that connect them (i.e., the *path similarity*) is as large as possible. We use the product of weights to reflect the case that, the similarity between two users decreases when some of their opinions differ (the similarity values lie in the interval [0..1]). Equally, one could use another approach, the averaged similarity, that is, the sum of similarities divided by the number of paths. Our goal is to find as many such products as possible. The reason we produce many products instead of one is twofold: (1) The decision regarding a node's membership is more fair when it is based on multiple values, and (2) the paths created will be used for multicasting, as explained in the next section.

To find the path products, we use two procedures: a *greedy* and a *stepwise* procedure. The greedy procedure seeks for paths from a node U to the members of a community C (and therefore to the community itself), where each link has the maximum similarity value. These paths can be as long as possible, provided that, at each step, the link with the largest weight is chosen. The largest weights of the paths found guarantee optimality, in terms of social similarity. The stepwise procedure is implemented after the greedy one, and searches for paths not found by the greedy strategy with the minimum number of links. The stepwise strategy can be proven optimal in cases when the greedy strategy can't find satisfactory paths (in terms of similarity), thus shortest paths can be used as an alternative. In the remaining of this subsection, the stepwise and the greedy strategies are presented.

The Greedy Procedure

The greedy, is a depth-first based procedure that works as follows: For a network G, we first identify a node (or user) U as the new root node $\mathtt{R}_{new}$ and the members of a community C whose similarities to U are going to be computed. The direct connections of $\mathtt{R}_{new}$ are identified and placed in a list L with decreasing order of link weights w. Then, we check if any of the connections in L involve a member of C. If not, we start with the first element of L, L_1, where $w_{\mathtt{R}_{new},L_1}$ is the maximum. L_1 becomes the new root $\mathtt{R}_{new}$ and the link U$\rightarrow$ $\mathtt{R}_{new}$ becomes the *working link*. Then, L is updated with the direct connections of $\mathtt{R}_{new}$. To keep track of the root updates, we use a pointer $prev$ as follows:

$$L_1.prev = \mathtt{R}_{new} \quad (2)$$

$$\mathtt{R}_{new} = L_1 \quad (3)$$

We subsequently update the new root $\mathtt{R}_{new}$ and L in a similar way, until we reach a point where L_1 is a node in C. In every updating step, we also update the similarity a of the path formed by the selected roots:

$$a = a \times w_{\mathtt{R}_{new}\rightarrow L_1} \quad (4)$$

where the starting value of a is the similarity between U and the first L_1 on the path. After reaching a node in C, we return to the current root and successively examine the next elements of L, $L_2, L_3, \dots L_n$ (L_n is the last element of L) using the procedure described. After processing all these elements, we return to the current root and similarly work backwards by setting as new root its $.prev$ value:

$$\mathtt{R}_{new} = \mathtt{R}_{new}.prev \quad (5)$$

The pseudo code of the procedure used to compute the similarities between unrelated users is given in Algorithms 1-3. It consists of a main algorithm, Algorithm 1, and two called methods, one to update the root and one to work backwards in order to find new maximum similarities, Algorithms 2 and 3, respectively. We now use the example of Fig. 1 to illustrate how this procedure works. Suppose that we seek to find the similarity between I and the members of community C_1. Initially, we set U=I and the corresponding list $L_{\mathtt{I}}$ of node I will be $\{\mathtt{G,F,E,H}\}$. Since the link I$\rightarrow$G has the maximum weight, 0.92, we set L_1 = G and the *root_update* method is called with parameters ($\mathtt{R}_{new}, L_1$,a)=(I,G,1). Now, the algorithm repeatedly follows the link with the maximum weight until reaching a node in C_1. Since L_1=G is not in C_1 and the list of G's neighbors is not exhausted (meaning that

Algorithm 1: Computing Similarities Between Unrelated Users – Greedy Procedure (Main Routine)

```
input : All similarities between connected users, a
        community $C$, and a user U
output: The highest existing similarity between U and
        members of $C$
1  begin
2  Set U = $\mathtt{R}_{new}$; // new root node
3  Set $a = 1$; // keep the similarity of the path formed by
     the selected roots and its maximum
4  $L_{R_{new}} = \{L_1, L_2, \dots L_n\}$; // Links to the root in
     descending order of link weights
5  Start from link $\mathtt{R}_{new} \rightarrow L_1$;
6  Mark link $\mathtt{R}_{new} \rightarrow L_1$ for path as processed;
7  root_update ($\mathtt{R}_{new}, L_1, a$);
8
9  if ($L_{L_1}$ is exhausted) then
10     Select next $L_1$ from $\mathtt{R}_{new}$;
11     root_update ($\mathtt{R}_{new}, L_1, a$);
12 end
13 Update path similarity using (4);
14 Save $a$ for path; // path similarity
15 $a = a / w_{\mathtt{R}_{new}\rightarrow L_1}$; // path weight before entering C
16 backward_examination($\mathtt{R}_{new}, a$);
17 end
```

Algorithm 2: Root Updating (Auxiliary Routine)

```
input : $\mathtt{R}_{new}, L_1, a$
output: Another new root $\mathtt{R}_{new}$
1  method root_update ($\mathtt{R}_{new}, L_1$,a);
2  while ($L_1$ is not in C) and ($L_{L_1}$ not exhausted) do
3      Update path similarity using (4);
4      Update root information using (2) and (3);
5      From $L_{R_{new}}$, Select $L_1$ such that $R_{new} \rightarrow L_1$ is
         unprocessed, $L_1 \neq$ U and $w_{\mathtt{R}_{new}\rightarrow L_1}$ is maximum;
6      Mark link $\mathtt{R}_{new} \rightarrow L_1$ for path as processed;
7  end
8  end method
```

unprocessed links exist between G and nodes other than U or its previous node in the path, in this case, U again, see line 2, Algorithm 2), the root information is updated according to (2) and (3) and the path similarity is updated according to (4), see also lines 2-3 of Algorithm 2:

$$a = 1 \times w_{\mathtt{I}\rightarrow\mathtt{G}} = 1 \times 0.92 = 0.92$$
$$\mathtt{G}.prev = \mathtt{I}$$
$$\mathtt{R}_{new} = \mathtt{G}$$

The list for $L_{\mathtt{G}}$ is: $L_{\mathtt{G}} = \{\mathtt{I,F,H}\}$ (line 5, of root updating algorithm) and we pick F as the next node L_1, because it is not the previous node of G, nor U and the link G$\rightarrow$F remains unprocessed. Since F is not in C_1 and $L_{\mathtt{F}}$ has unprocessed links, we update the path similarity and the root information

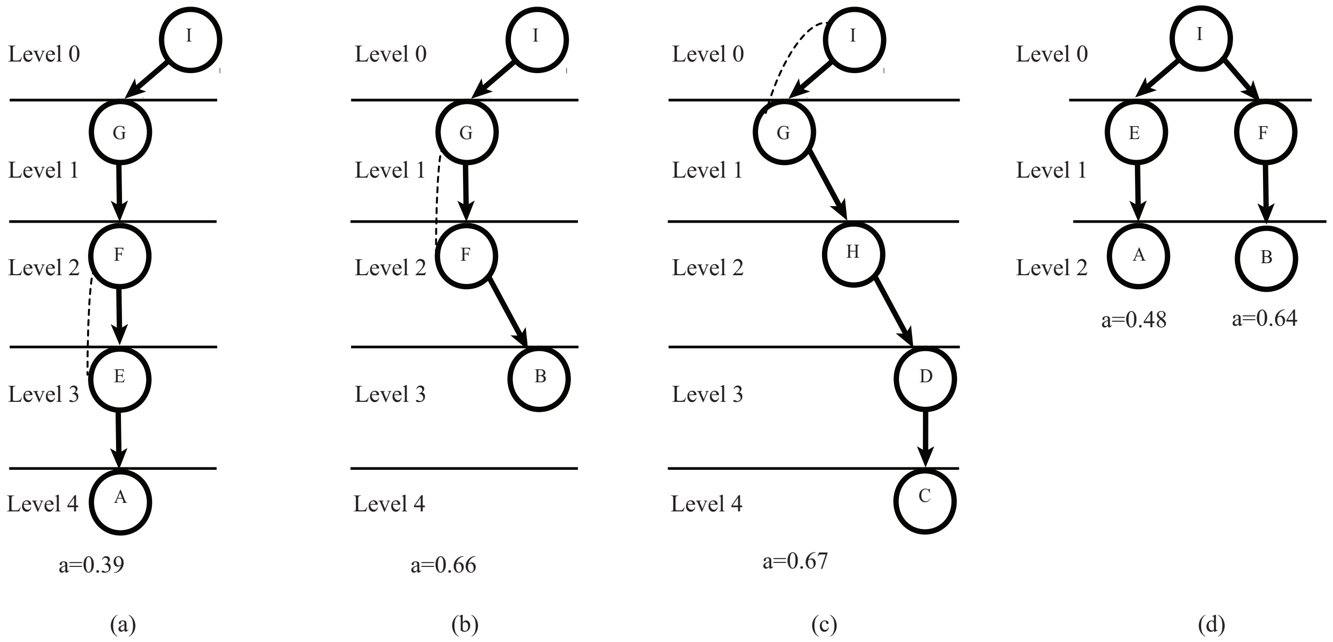

Fig. 2. Similarity paths formed for U=I, $C = C_1$: (a)-(c) Greedy strategy, (d), Stepwise strategy

Algorithm 3: Backward Examination (Auxiliary Routine)

```
input : $R_{new}, a$
output: A new maximum similarity $a$
1  method backward_examination($R_{new}, a$);
2  $a = a/w_{R_{new} \to R_{new}.prev}$ // weight of path till $R_{new}.prev$
3  Use (5) to set as root the $.prev$ value of current root;
4  if $R_{new}$=$NULL$ then
5      terminate;
6  else
7      From $L_{R_{new}}$
8      if ($L_{R_{new}}$ not exhausted) then
9          Select $L_1$ such that $R_{new} \to L_1$ is unprocessed,
             $L_1 \neq$ U and $w_{R_{new} \to L_1}$ is maximum;
10         root_update ($R_{new}, L_1, a$);
11     else
12         backward_examination($R_{new}, a$);
13     end
14 end
15 end method
```

as follows:
$a = 0.92 \times w_{\mathrm{G} \to \mathrm{F}} = 0.92 \times 0.9 = 0.828$
$\mathtt{F}.prev = \mathtt{G}$
$\mathtt{R}_{new} = \mathtt{F}$

Since $L_{\mathtt{F}} = \{\mathtt{G,E,I,B,A}\}$, node E is selected as L_1 and $L_{\mathtt{E}} = \{I, F, A, H\}$. Since $L_{\mathtt{E}}$ has unprocessed links, it follows that another update is performed:
$a = 0.828 \times w_{\mathrm{F} \to \mathrm{E}} = 1 \times 0.92 = 0.678$
$\mathtt{E}.prev = \mathtt{F}$
$\mathtt{R}_{new} = \mathtt{E}$

Since $L_{\mathtt{E}} = \{\mathtt{I,F,A,H}\}$, node A is selected as L_1 and $L_{\mathtt{A}} = \{B, J, E\}$. However, A resides in the desired community, C_1. Thus, the body of `root_update` is not executed and the control passes to line 13 of the main algorithm. The value of a is updated (line 13) to $0.678 \times 0.58 = 0.39$ and this value is the weight of the examined path $I \to G \to F \to E \to A$. Now, we have computed the maximum similarity from path I→G→F→E→ C_1 and the algorithm will check if there exist higher similarity values from the path I→G→F to C_1. Thus, we have to work backwards to root F: First, a has to be divided by the weight of the last link leading to C_1, $w_{\mathrm{E} \to \mathrm{A}}$ (line 15 of the main algorithm). Thus a becomes 0.678. Then, method *backward_examination* is called with parameters ($\mathtt{R}_{new}, a$)=(E, 0.678,). Line 2 of this method is used to find the similarity of the path till the previous root (in our example, till F). So, a becomes $a/w_{R_{new} \to R_{new}.prev} = 0.678/w_{EF} = 0.82$. Then, we set as new root the previous of the current (in our example, to the previous of E, which is F) and we search $L_{\mathtt{F}}$ to find the next unprocessed link to F, which can't be U, nor F's previous node G, nor last processed node E. Since $L_{\mathtt{F}} = \{G, E, I, B, A\}$, node B is selected and *root_update*$(F, B, 0.828)$ is called. Similarly, B is selected as L_1 (B lies in C_1), etc. Fig. 2 shows the paths detected as well as their similarities, from node I to community C_1. The back-edges (in dashed lines) show when the algorithm returns from current to previous root (line 3 of Algorithm 1). Proposition 1 gives the time required to find the path similarities.

Proposition 1: *Let m be the maximum length of the paths found by Algorithm 1. The time required to find the similarity paths is $O(m^2)$.*

Proof: Let $\mathcal{P} = \mathtt{P}_1, \mathtt{P}_2, \ldots \mathtt{P}_m$ be the path with the maximum length, m nodes, among all the paths found by Algorithm 1, from a node U to a community C. Assume that, this path starts from node P_1. Then, the path lengths originating from the other nodes of $\mathcal{P}$ will be $(m-1)/2+(m-2)/2+\ldots(m-(m-1))/2+(m-m)/2 \approx \frac{m-1}{2} \times \frac{m}{2} = \frac{m^2-m}{4}$. The time required to move from one node to its next neighbor with the largest weight w is $O(1)$. Moreover, each of the $\frac{m^2+m}{4}$ links is visited at most two times; one during the execution of the *root_update* method (on the way to reach community C) and one during the execution of the *backward_examination* method (on the way back from C to a previous node in order to find another similarity path). Thus, the time required to find the the similarity paths from a node U to a community C is at most $O(m^2)$. □

The Stepwise Procedure

The stepwise approach can be described as follows:

```
Step 1: Set $m'$ as the average length among all the paths
        found by Algorithm 1 and mark all these paths as
        processed.
Step 2: Search for unprocessed paths with length $< m'$.
        If such paths exist and their products are larger
        than at least one of the products of the paths
        found in the greedy procedure, these paths are
        also added and they are considered to determine
        the membership of U in $C$.
```

In our example, $m' = 4$, thus we search for unprocessed paths with length less than 4. The 3 paths found are I→E→A, I→F→A, and I→F→B and since there is at least one path found by the greedy strategy with lower similarity, namely I→G→F→E→A with similarity weight 0.39, the three paths will also be examined to determine if I is eligible to be a member of C_1. Fig. 2(d) shows the paths found by the stepwise strategy. As will be described in Section IV-C, unlike the greedy strategy, the stepwise strategy has no mechanism to eliminate cycles [it can be easily seen that, the 3 paths I→E→A, I→F→A, and I→F→B form a cycle. In Fig Fig. 2(d), the cycle has been eliminated by removing the path I→F→A, as will be explained in Section IV-C]. To use the paths formed by the stepwise strategy, a mechanism to eliminate cycles is needed. This mechanism will be described in Section IV-C. Proposition 2 shows that the stepwise procedure takes at most the same amount of time as the greedy.

Proposition 2: *Let m' be the minimum length among all the paths found by Algorithm 1. The time required to find the similarity paths with the stepwise strategy is at most $O(m^2)$.*

Proof: The proof is straightforward since m' is clearly less than m. Also, the paths detected by the stepwise strategy include links that have been marked as processed in the greedy strategy because, otherwise, the paths found by the stepwise strategy would have already been detected by the greedy strategy (since we search for paths with larger products compared to the paths found by the greedy strategy). So, the number of these links is also restricted by m^2 and this completes the proof. □

B. Determining Overlapping Communities

The determination of possible overlaps, requires the definition of a measure of how strongly the members of a community are connected. This measure is the *Average Community Connectivity* (ACC) for a community C and is defined as the average of the NCDs [39] of all the nodes in the C, that is:

$$ACC_C = \frac{\sum_{i=1}^{n} NCD_i}{n} \tag{6}$$

Then, a threshold value $\mathcal{T} = 0.5$ (see [10]) is set, for the percentage of paths that need to have highest similarity product than ACC_C, in order for a node U to be eligible to be a member of C. Specifically, the following procedure is used:

```
Step 1: Use the greedy and the stepwise procedures to
        find the similarity paths from a node U to a
        community $C$.
Step 2: Group these paths into two classes: Class 1, that
        includes the paths from the community of U to $C$
        and Class 2, that includes the paths from U to $C$
        through other communities.
Step 3: If 50% of Class 1 or Class 2 paths are $> ACC_C$
        then
            Include the node to the community.
            An overlap has been found.
        else check another node U.
```

In our example:

$$ACC_{C_1} = \frac{\sum\limits_{i=1}^{n} NCD_i}{n} = \frac{0.9 + 0.93 + 0.98 + 1}{4} = 0.95$$

.

All the paths found by the greedy and the stepwise strategy are Class 1, except the one in Fig. 2(c). Regardless of the value of $\mathcal{T}$, there is no path found with similarity higher than 0.95. Thus, node I can by no means be considered as member of C_1.

C. Updating Communities

When a new node enters the network, the communities have to be updated. This is done firstly by observing their behavior so that, their similarity with existing nodes can be examined. Then, the following procedure adds an incoming node to a community.

```
Step 1: Select a new-coming node U.
Step 2: Find the similarity path to a community $C$, that
        has the maximum weight among all the paths
        between U and the communities to which U is
        related.
Step 3: Compare the weight value found in Step 2 to
        the $ACC_C$ of $C$
Step 4: If the weight is $> ACC_C$ then
            Include the node to the community.
```

Examine possible overlaps between `U` and other communities, using the method described in the previous subsections.

`else` find the highest similarity path between `U` and another community C and repeat `Step 3` and `4`.

`Step 5`: **If** `U` is not determined as a member of a community, **then** it is viewed as an individual community.

For example, consider node `M`. Its maximum similarity weight is 1, to node `H`. Thus, `M` is added to C_3, the community where `H` belongs. Then, using the proposed community detection scheme, a check for possible overlaps with communities C_1 and C_2 is made. For example, in order to check if `M` is eligible to be considered as member of C_1, the proposed algorithm finds the following paths:

To C_1 (using the greedy procedure):
`M`→`H`→`G`→`I`→`E` →`F`→`B` with similarity weight 0.40
`M`→`H`→`G`→`I`→`E` →`A` with similarity weight 0.35
`M`→`H`→`G`→`I`→`F`, →`B` with similarity weight 0.47
`M`→`H`→`G`→`F`, →`A` with similarity weight 0.55
`M`→`H`→`D`→`C`, with similarity weight 0.912
`M`→`C`, with similarity weight 0.9
`M`→`D`→`K`→`C`, with similarity weight 0.71

To C_1 (stepwise procedure, $m' = 5$):
`M`→`D`→`C`, with similarity weight 0.912
`M`→`K`→`C`, with similarity weight 0.79

and since $ACC_{C_1} = 0.95$, it follows that `M` can't be a member of C_1

The communities detected can have one or more nodes in common (overlaps). However, if the percentage of overlaps between two communities is higher than 50%, then these two communities can be merged [10]. Also, the proposed method is dynamic in the sense that, it can be applied in regular intervals to determine more possible overlaps and newly formed communities (as the network relationships dynamically develop). Also, this scheme can be applied without starting from communities already formed. In this case, we start with n separate nodes and we form communities of two members, using the links with the highest similarities for each pair of nodes. Then, the greedy and stepwise procedures are used to keep on merging communities. In this case, the complexity of the proposed scheme is at most $O(n^2)$, since each node may be connected with at most $n-1$ other nodes.

IV. Constructing the Multicast Trees

Generally, the multicast trees constructed tend to have an important property: their lengths (in terms of the number of links) are usually kept to the minimum. However, in the process of desinging data multicasting algorithms with social criteria, we have to consider cases where there may be links on the shortest path between nodes whose similarity is low. In such cases, data have to be forwarded between more relays, apparently with higher similarities between them (see also Subsection VI-B). In this sense, we focus our attention on trying to construct multicast trees such that there exist both short and longer paths. Also, the presence of cycles is another issue that needs to be considered and will be analyzed. Apparently, we refer to tree structures, so we have to find a way to eliminate possible cycles.

To define cycles and to assure that the distribution of data is indeed performed via social criteria, we introduce a spatial metric of node similarity, the *social distance*. The term was initially given by Shang et al., [12] but here we will use it in a slightly different manner, to spot topologically the members of a community. Our aim is to use this metric in order to identify the possible cycles as will be described in paragraph IV.C. The spatial metric introduced is a modified version of the well-known *taxicab distance*. The taxicab distance has been widely used (see [40] for a good example) for connectivity graphs representing dynamic networks with nodes with changing behavior.

We consider a network with n nodes and M is the number of nodes that will participate in a multicast. Intuitively, since a multicast involves part of the entire network, $M \leq n$. We model the *social distance* r between two nodes n_1, n_2 as their taxicab distance d_{n_1,n_2}, which is defined as follows:

$$r = d_{n_1,n_2} = |x_1 - x_2| + |y_1 - y_2| \tag{7}$$

where $n_1 = (x_1, y_1)$ and $n_2 = (x_2, y_2)$ are the two nodes and their coordinates on the Cartesian $\mathcal{R}^2$ plane. Apparently, the largest the value of r, the lowest the similarity between n_1 and n_2. The relationship between the social distance and similarity is given by

$$w_{n_1,n_2} = \begin{cases} \frac{10-d_{n_1,n_2}}{10}, & \text{if } \mathrm{d_{n_1,n_2}} \leq 10 \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Eq. 8 models the situation where, similar in terms of social behavior, nodes have a taxicab distance up to 10 (thus, similarity of maximum 1). If the taxicab distance is larger than 10, there is no similarity between the corresponding users. Thus, if the taxicab distance of n_1 and n_2 is 2, their similarity is 0.8 and if their taxicab distance is 0.2, their similarity is 0.98.

For three nodes n_1, n_2, and n_3, we can define the notion of *betweeness*: n_3 is considered to be *between* n_1 and n_2 if

$$d_{n_1,n_2} = d_{n_1,n_3} + d_{n_3,n_2} \tag{9}$$

where the notion of “betweeness” does not imply collinearity between n_1, n_2, and n_3.

Generally, a multicast tree construction algorithm includes three phases [9], [14]: (i) Identify the receivers, (ii) Choose the proper paths through the neighborhood (relay nodes), and (iii) Remove the cycles. In the following subsections, we will describe these steps and we will show that the paths formed by our community detection algorithm can actually be used to form the broadcast trees.

A. Search for Receivers

When a node decides to multicast, it has to identify the receivers. This is done by sending a message to the entire network, so that all the nodes would know if they are receivers (or targets). Then, the source will have to construct the muticast tree.

Since multicasting will be based on social features such as similarities it follows that, the communities where the receivers belong to, have to be identified. By doing so, the source node will use the proper set relays, that will include the nodes that are highly related to these communities. This information is known from the paths constructed by the community detection algorithm.

B. Choice of Proper Paths

In the second phase, every multicast member chooses the proper neighboring nodes to connect to. In the proposed scheme, this is accomplished via the following criteria: the node tries to connect with the neighbor that has the highest similarity among all its neighbors. This list is available from the community detection scheme. If the taxicab distance between the two nodes is less than a predefined range r then, the two nodes can connect directly. Otherwise, the two nodes have to connect through relays. Two conditions must hold:
(i) The similarity of the path formed between the two nodes must be as large as possible
(ii) If relays are used, the social distance covered through the relays should be at least as large as the distance covered if the two nodes were to connect directly.

In Proposition 3, we show that, the fist condition indeed holds once the community detection algorithm (as described in the previous section) is applied. Proposition 4 states that when two nodes communicate through a number of relays, the social distance between them is indeed covered, so the distribution is indeed performed under correct social criteria.

Proposition 3: *The similarity of the path formed between the two nodes is the maximum possible.*

Proof: Assume that n_i is the source node and we order its neighbors in a descending order based on their similarity to the source. Thus, as described in Algorithm 1, the neighboring nodes form an ordered list $L = \{L_1,\ L_2, L_3, \dots L_n\}$, where L_1 and L_n have the largest and the smallest similarity to n_i, respectively. In every iteration of *root_update*, a new root is selected and the maximum similarity path from this root to the desired community is detected. For example, if n_j is the new root, then the path $p' = n_j, n_{j+1}, \dots n_k$, where n_k lies in the desired community C, has the maximum similarity. Suppose that there exists another path $p'' = n_j, \dots n_k$ with the same number of hops as p', that starts from n_j and ends in n_k. Then, the two paths should have at least two different hops. To see this, assume that the two paths differ by just one hop:

$$p_j' = n_j, n_{j+1}, n_{j+2}, \dots n_k$$
$$p_j'' = n_j, n_{j'+1}, n_{j+2}, \dots n_k$$

When n_j becomes the current root, it will examine its L list, where node n_{j+1} will be located before $n_{j'+1}$, indicating that $w_{n_j \to n_{j+1}} > w_{n_j \to n_{j'+1}}$. Even if, later, $n_{j'+1}$ becomes the current root, it will find the links $n_{j+2}, \dots n_k$ marked as processed, thus the path p_j'' will never be used. Thus, p' has the maximum similarity, if it differs to p'' by one hop. If p' and p'' differ by two or more hops, that is:

$$p_j' = n_j, n_{j+1}, n_{j+2}, \dots n_k$$
$$p_j'' = n_j, n_{j'+1}, n_{j'+2}, \dots n_k$$

then, if $n_{j'+1}$ becomes the current root, the path $n_{j'+2}, \dots n_k$ is unprocessed, thus it will be examined by the algorithm, to see if it is a path with maximum similarity starting from $n_{j'+2}$. □

Proposition 4: *The social distance covered through the relays should be at least as large as the distance covered if the two nodes were to connect directly.*

The proof is based on the well-known inequality of the taxicab distance:

$$d(n_1, n_k) \leq d(n_1, n_2) + d(n_2, n_3) + \dots + d(n_{k-1}, n_k) \quad (10)$$

for any path $p = n_1, n_2, n_3 \dots\dots n_{k-1}, n_k$; it is trivial and thus it is omitted.

Phase 2 of the multicast tree construction determines the proper paths between a source node and the desired targets, based on the community detection scheme presented in Section III. Propositions 1 and 2 prove that, the community detection scheme can be used to construct such paths. However, in order to use these paths for multicasting, we need to assure that they indeed constitute a tree. Thus, we have to show how cycles are eliminated, because, in the presence of cycles, a large number of unnecessary transmissions might exist, thus bandwidth is wasted.

C. Cycle Removal

Generally, during the construction of multicast trees, there may be paths that connect different pairs <*source, target*> that use a number of similar relays. Under this scenario, cycles are formed. In this section, we describe how Algorithm 1 prevents the formulation of such cycles.

The Greedy Strategy Suppose that a node n_i is selected as root R_{new} during the implementation of Algorithm 1. This means that it has the highest similarity to the previous root $R_{new}.prev$, compared to all neighbors of $R_{new}.prev$. Then assume that, starting from R_{new}, a number of node pairs $R_{new} \to n_{j1}, n_{j1} \to n_{j2} n_{j2} \to n_{j3} \dots n_{jk} \to n_k$ will form a path p leading to a node n_k that resides in the desired community C. Now, assume that line 3 of the backward examination algorithm is executed and one of the aforementioned nodes becomes the root, say n_{j1}. Then, the next root will be another node $n_{j'2}$, other than n_{j2}, since $n_{j1} \to n_{j2}$ has been marked as processed in the path. Then the neighbor list of n_{j2} will be used to determine the next root. If the next root is a node in p, then, in order for the greedy strategy to continue examination, the remaining links (hops) have to be completely different than the ones found in p, because these links have been marked as processed (containing the largest similarities), so they can't be reused.

Thus, all paths from a current root to a community have at most one link in common and this suffices to eliminate cycles. In the example of Fig. 2 note that the paths formed starting from `G` have only the link `G`$\rightarrow$`F` in common.

The Stepwise Strategy The stepwise strategy seeks for all shortest paths (in term of communication links) that the greedy strategy has not detected and have similarity products larger than at least one of the paths found by the greedy strategy. Since there is no cycle elimination mechanism, we can remove the cycles by the following steps:

`Step 1`: Spot the cycle.
`Step 2`: Eliminate the link with the lowest similarity.

We can topologically define a cycle as follows: For any three nodes n_1, n_2, and n_3, where n_3 is *between* n_1 and n_2 in the way described by Eq. 9, there exists a cycle involving n_1 and n_2, if there is a node $n_4 \neq n_3$ such that:

$$d_{n_1,n_2} = d_{n_1,n_4} + d_{n_4,n_2} \tag{11}$$

When a cycle is spotted, we compare the similarities of the two paths, $n_1 \rightarrow n_3 \rightarrow n_2$ and $n_1 \rightarrow n_4 \rightarrow n_2$, and we remove the one with the lowest similarity.

D. Proof of Tree Topology

In the above subsections, we have described how the similarity paths generated to detect communities can be used as paths for multicasting over the network. However, we need to prove that the paths generated by the greedy and the stepwise strategy in fact form a tree topology. To prove this fact, we need to show that the graph formed by the aforementioned strategies is connected and acyclic as described in Proposition 5:

Proposition 5: *The paths generated by the greedy and the stepwise strategy of the community detection scheme form an acyclic and connected graph.*

Proof: When the greedy strategy is executed, every root node tries to connect to another node to which it is more similar based on social criteria, in its effort to connect to the targeted community. Thus, the topology generated is connected. Also, as described in subsection IV-C, the greedy strategy has a mechanism to prevent the appearance of cycles. Periodically, as new nodes arrive and connect in the network, the procedure has to execute again to identify new community members as described in our update procedure. Naturally, when new nodes enter the network, the new paths found by the greedy strategy are also acyclic.

The stepwise procedure has no embedded mechanism to prevent cycles, but the cycle removal strategy we suggested in the previous subsection guarantees of no cycles. Apparently, since the strategy finds the minimum, in terms of number of links, paths to a desired community, the generated topology is also connected. Thus, both the greedy and the stepwise strategy generate connected and acyclic topologies or trees. □

V. Data Distribution

In this section, we focus our attention on the multicast pattern followed to distribute the data. We will consider two cases: (1) A node multicasts some data to a number of nodes residing in a community, and (2) Multiple nodes need to multicast different data to a community, using the same data paths.

A. Case 1: Multicasting From a Node to a Community

The basic idea behind the multicast pattern is to forward data between the communication links based on social criteria [10]. For this reason, when a source node wishes to multicast, it checks if itself overlaps with the targeted community. In this case, it can be used as a relay for other nodes that wish to multicast towards this community. If not, it has to search for other nodes in its community, which were found to overlap with the target community and use them as relays. If there are no overlapping nodes in its community, the source seeks for the maximum similarity paths through other communities (also found by the community detection scheme). The source node either uses the greedy or the stepwise strategy, but generally not both, since they may include paths with similar nodes and generate cycles (for example, the paths shown in Fig. 2 can form the cycle `I`$\rightarrow$`G`$\rightarrow$`F`$\rightarrow$`I` if paths `I`$\rightarrow$`G`$\rightarrow$`F`$\rightarrow$`B` (from greedy strategy) and `I`$\rightarrow$`F`$\rightarrow$`B` (from stepwise strategy) are used. A combination of both strategies can be used, only if the paths defined by them contain no cycles, but this does not hold generally.

When multicasting from a node to members of a community, there are no link contentions since one node uses the entire multicast tree. The multicasting pattern is described as follows: We start from the source node and we use all the paths found by our greedy or stepwise strategy. The maximum level number between all these paths is m (maximum length among the paths). If level j exists on a path (not all paths have the same length), then the data is distributed from the source to the node with coordinates (x_{ij}, y_{ij}), which is the jth node on the selected path i. This node becomes the new source, that will distribute the data to the next node $j+1$ of path i. For each path, the procedure stops when level j does not exist for path i. When all transmissions finish (we are at level m of some path i) then, we have reached the relays or the target nodes. Then, there remains to distribute the data inside the community. This procedure is described in Algorithm 4

Complexity Analysis Let α be the startup cost for each data packet distributed from node to node, ε be the number of bytes transmitted and φ be the transmission time per byte. Then, each data packet costs $\alpha + \varepsilon\varphi$ time. Thus, the maximum time required to complete multicasting is $m \times (\alpha + \varepsilon\varphi)$, since a data packet will travel at most m nodes on the tree and all the paths distribute the data in parallel.

Algorithm 4: Multicast Pattern for Case 1

```
input : selection between greedy and stepwise strategy,
        or combination of both
        m // the maximum length (level) found
        between the paths
        $(x_{ij}, y_{ij})$ // position coordinates of relay on
        path i, level j
output: Multicasting from a node to a community

Select source node;
for $i = 1$ to $number\_of\_detected\_paths$ do
    for $j = 1$ to $m$ do
        if level j exists in the path then
            transmit from source to $(x_{ij}, y_{ij})$;
            $path.next.source = (x_{ij}, y_{ij})$;
        else
            relays or targets reached // end of path
            distribute to multicast receivers inside
            the desired community;
        end
    end
end
```

B. Case 2: Multicasting From Many Nodes to a Community Using the Same Data Paths

In this subsection, we describe the case when more nodes need to multicast to the members of a community using the same data paths. For example, when a node just enters a network and has not yet been identified as a member of a community, there are no similarity paths between it and the other communities. Moreover, if this node is connected to just one or very few nodes, it may require to use the same similarity paths with its connections, when it wishes to multicast. Problems arise when two or more nodes wish to multicast towards the members of a community at the same or almost the same time. In previous multicast algorithms there is no care taken for such cases and each node has to wait until the previous node completes its multicast.

In this paragraph, we show how to fully occupy the multicast tree paths by pipelining the data distributions from different sources. Our scheme gives optimal distribution time in cases of multicasts from multiple nodes. Assume that there are k nodes that wish to multicast to a certain community at the same time. We define a *segment* as the period of time when a source transmits to nodes found at a certain level of the multicast tree. Then, our multicast scheme follows the pattern described below:

Segment 1: Source $n = 1$ distributes data to level $j = 1$ nodes of each path from i to $number\_of\_detected\_paths$, in the way described by Algorithm 4.

Segment 2: Source $n = 1$ distributes data to level $j = 2$ nodes of each path from i to $number\_of\_detected\_paths$, while source $n = 2$ occupies level $j = 1$ nodes of each path, in the way described by Algorithm 4.

....

....

Segment s: Source $n = 1$ distributes data to level $j = s$ nodes of each path from i to $number\_of\_detected\_paths$ (if level s does not exist in some paths, the relays or the targets are already reached through this path), while source $n = 2$ occupies level $j = s - 1$ nodes of each path, etc. in the way described by Algorithm 4.

From the pattern described above it follows that, when we reach segment s, the first source finishes its multicasting while $s - 1$ more multicasts are executed in parallel. This fact reduces the total distribution time as described in the complexity analysis that follows:

Complexity Analysis: Clearly, the first source completes its multicasting in $s \times (\alpha + \varepsilon\varphi)$ time. Then, the remaining $k - 1$ multicasts will be completed at most in $(k-1)(\alpha+\varepsilon\varphi)$. Thus, the total transmission time is $s \times (\alpha+\varepsilon\varphi)+(k-1)(\alpha+\varepsilon\varphi) = (s+k-1)(\alpha+\varepsilon\varphi)$. Now, if k multicasts execute sequentially (one after the other) on the same multicast tree, their cost will be $ks(\alpha+\varepsilon\varphi)$ and we have $ks(\alpha+\varepsilon\varphi) > (s+k-1)(\alpha+\varepsilon\varphi)$ for any value of $s, k > 1$. Thus, the data distribution time of the pipeline based communication pattern is optimal.

In this analysis we assumed that, all transmissions involve equal, in terms of size, data. If this is not the case, the multicast pattern can still work, but we may need to introduce some delays between the segments (e.g., a smaller file may need to wait until it uses the next link on the path, which is currently occupied by a largest file).

Table I summarizes the most important parameters of this work.

TABLE I
NOTATIONS USED IN THIS PAPER

Parameter	Meaning
G	A weighted, undirected graph
V, E	Nodes and edges of the graph
NCD_i	Network connectivity degree of user i
C	a community
R_{new}	New root when applying the greedy strategy
L	List of links to the root in descending order of link weights
a	maximum similarity in the greedy scheme
m'	Average path lengths for the stepwise strategy
ACC_C	Average network connectivity of community C
r	social distance between two nodes computed as the taxicab distance of these nodes
d_{n_1,n_2}	the taxicab distance between nodes n_1 and n_2
p	Symbol for a path of nodes

VI. Simulation Results and Discussion

In this section, we verify the functionality of the community detection algorithm which is used for data distribution across the network. Then, we evaluate the performance of the data distribution algorithm proposed and we compare it to other important algorithms found in the literature.

TABLE II
SIMULATION PARAMETERS

Parameter	Value
Number of nodes N	1000-2000
Mixing parameter μ	0.1-0.4
Node degree power law distribution exponent γ	2 to 3
Minimum node degree, k_{min}	10
Maximum node degree, k_{max}	30
Average node degree, k	20
Community size power law distribution exponent β	1 to 2

A. Community Detection

To verify the community detection scheme, we carried out a series of simulations implemented using a Java simulator and we compare it to well-known algorithms in the literature. To generate the network, we used the benchmark provided by Lancichinetti et al. [26]). The following parameters are required by the benchmark:

- The degree of each node, which is taken by a power law distribution with exponent γ, with k_{min} and k_{max} being the extreme values of the distribution chosen in such a way, that the average degree is k. Typically, $2 \leq \gamma \leq 3$.
- The mixing parameter, that is, the average ratio of external degree/total degree for each node, which is denoted by μ. Specifically, each node shares $1 - \mu$ of its links with the members of its community and μ with members of other communities.
- The exponent for the community size power law distribution β. Typically, $1 \leq \beta \leq 2$. The community sizes should sum to N, the number of nodes of the graph.

For our experiments, we set the above parameters as shown in Table II. When a node is generated, our community detection scheme tries to place it in a community based on its similarity with the nodes it is linked. To quantify the similarities between nodes, we used the online comment-based metric system presented by Bu et al. [3], where a number of emotional phrases used by social network users is ranked from 0 to 1, and a rank close to 0 indicates opposing behavior while a rank close to 1 indicates supportive behavior. For example, a comment including the word “Fantastic” is measured as a highly supportive, while a comment including the phrase “Idiot” may be measured as highly opposing. After several iterations of our community detection scheme, a node may be found to overlap to other communities or even to belong to a different community.

To verify the functionality of our community detection scheme, we compared our scheme with two well-known strategies that detect overlapping communities: Conductance [33], a new state-of-the-art path-based algorithm for community detection and COPRA [28], a well-known label propagation based scheme. We use two popular metrics to verify the functionality of our community detection scheme so that, we can use it for data distribution:

- *Modified Normalized Mutual Information* (MNMI): It compares the similarity between different parts of the network, but unlike the traditional NMI, it also takes into account the importance of each node in the community structure [41]. Apparently, this value lies in the interval $[0, 1]$. For a functional community detection algorithm, the MNMI should be as large as possible.
- *Number of Communities* (NoC): Naturally, a functional community detection scheme should be able to find approximately the same number of communities compared to well-established strategies.

MNMI Comparisons: We run sets of simulation for $N = 1000$, using the marginal parameter values for the typical values of β and γ ($\gamma = 2$ and 3, $\beta = 1$ and 2). Fig. 3 shows the experimental results. Each point is an average value taken from 10 random realizations of networks with the aforementioned parameters. From the results we see that, our community detection strategy outperforms the other two strategies in terms of NMIM. This is explained by the fact that the communities detected by our scheme have stronger integration because it takes into account the contribution of all the nodes in the network connectivity (see Eq. 6), while the Conductance strategy considers only the link weights to determine similarities and COPRA strategy uses labels with belonging coefficients that sum to 1, which just indicate the maximum number of communities to which any node can belong but do not indicate the importance of each node to a community. Note that, as the communities tend to have similar size (increasing β) the MNMI values tend to decrease as μ increases (this is true, because when communities have equal sizes then, in general, only parts of them have similarities between them, while a large and a small community might be entirely overlapping). Also, note that the simulation results are slightly better when the degree of each node increases (increasing γ).

NoC Comparisons: In our second set of experiments, we run sets of simulations for $N = 1000$ and $N = 2000$ and we created different networks with average degree values k = 10 to 30. Each point is an average value taken from ten random realizations of networks with the aforementioned parameters. As seen in Fig. 4, in the first set of results, the threshold value is $\mathcal{T}$ = 0.5 (see our overlapping communities detection strategy) and we conclude that, the Conductance strategy detects more communities than the other strategies. This is due to the fact that, our strategy forms communities in a more strict way (recall that the greed strategy examines paths including many links, whose values should be as large as possible, to determine possible overlapped communities). The two strategies outperform the COPRA strategy. In the second set, we set a lower threshold value, so our strategy approaches the performance of Conductance. Also note that, when $\mu > 0.4$, the performance of all strategies tends to decrease because a community can’t be strongly defined when an increasing number of its links is outside the community (thus, linking to other communities).

To summarize, our scheme tends to detect more strongly integrated communities with better MNIM performance compared to Conductance and COPRA. This is because our strategy takes into account the degree to which all the nodes participate in the formulation of each community, unlike

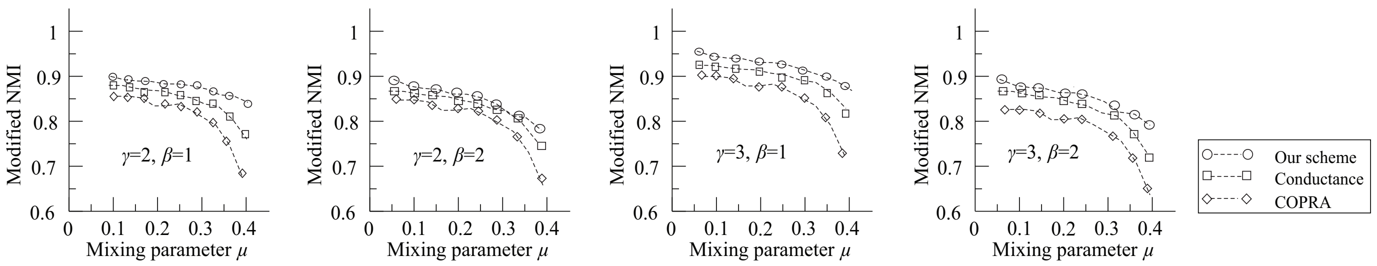

Fig. 3. MNIM values for Conductance, COPRA and our strategy

the other methods. The Conductance strategy detects more communities than our strategy, but this is mainly due to the strict criteria our algorithm uses to verify the membership of a node more than a community. We can infer that the number and connectivity of the communities detected by our scheme are satisfactory enough, so they can be used for the purpose of multicasting data across the network.

B. Data Distribution Via Multicasting

The data packets from a source to a destination node may need to be routed through a number of relays, that have a maximum social distance r_{max} between them. Consequently, the paths created by our greedy strategy are mostly used in cases when a small number of nodes found in the path are not located within a proper distance r_{max} between them. Similarly, direct transmission to the destination occurs only under the same condition (source and target are directly connected and they are within an appropriate social distance).

In our simulations, we assumed that each node knows the locations of all its neighbors and our transmission strategy obeys the following rules:

1) The source node transmits a packet directly to the destination node (such paths are found by our community detection scheme anyway), if the two nodes have social distance of at most r_{max}.
2) We use the paths found by the stepwise strategy (usually the ones with fewer nodes) when the nodes have an appropriate distance between them, otherwise, the paths of the greedy strategy are preferred, since they can heal social distance issues (see also Proposition 4 in Section IV).
3) The similarity paths information has to be refreshed regularly, so that alternative choices present themselves in case one of the neighboring nodes on the path is temporarily off.

For the simulations of this section, we assume that all the packets are of equal size. The value of r_{max} was set to from 3 to 4 and the remaining parameters are as presented in Table II. We compare our data distribution algorithm to well-known algorithms that perform multicasting with social criteria, Multi-CSDR [12], SPM [8], and Multi-Sosim [11], with respect to the delivery ratio and the latency.

Delivery Ratio: The delivery ratio refers to the total number of successfully delivered packets, divided by the number of packets expected to reach their destination. The advantage of our scheme, which will be displayed by the simulation results, is that each node of our multicast trees can have more than two children (thus, there are more paths available), while the most efficient of the compared schemes, Multi-CSDR and its variation Multi-Sosim generate binary trees because at each step, the destinations are split explicitly into two nodes. In order to have a fair comparison with the other schemes, we set the maximum TTL of each packet to 1,440 minutes and we run simulations for ten different TTL values (for example a value of 0.1 means 144 minutes, a value of 0.2 means 128 minutes, etc.). With a node degree of 25 and a mixing parameter of 0.2, we had an average of five links to members of other communities for each node, thus, an average of five destinations for each multicast. With the same node degree and a mixing parameter of 0.4, we had an average of ten destinations to each multicast (see Fig. 5).

When the value of r_{max} was set to four (that is, nodes with social distance less than 4 or similarity greater than 0.6 can communicate directly), we note that all four strategies converge to a delivery ratio of 100% when the TTL value approaches 0.5. Our scheme has a constant delivery ratio of 100% for any TTL value, because it permits larger number of forwardings through the multiple communication paths that it generates (more than one paths can be used for the same data and the total load can be split evenly between these paths). However, as the number of destinations increase to 10, the paths can not longer be used in the sense described previously, resulting in lower delivery ratios, especially when TTLs are small. Note that our strategy approaches a delivery ratio of 100% for TTLs > 0.5 while for the other strategies this occurs for TTLs that approximate 0.6 or higher. The performance of all strategies decreases when the value of r_{max} decreases. For example, when $r =$ three (that is, only nodes with social distance less than three or similarity greater than 0.7 can communicate directly), we note that our strategy has a constant delivery ratio of 0.9 for 5 destinations, and a maximum of ≈ 0.82 for TTLs > 0.5. This is due to the fact that, the greedy strategy is more intensively used when the limit regarding the nodes permitted to communicate becomes stricter. The problem is even more intense for Multi-CSDR, Multi-Sosim and SPM [8] that have fewer paths available for packet forwarding.

Multicast Latency: The multicast latency is the time elapsed from the time the source starts the distribution to the time when all targets have received the data. From Fig. 6, we can

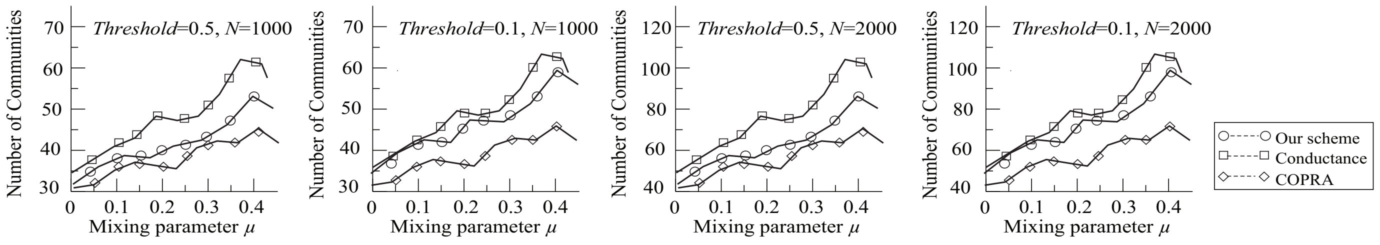

Fig. 4. Number of communities for Conductance, COPRA and our strategy

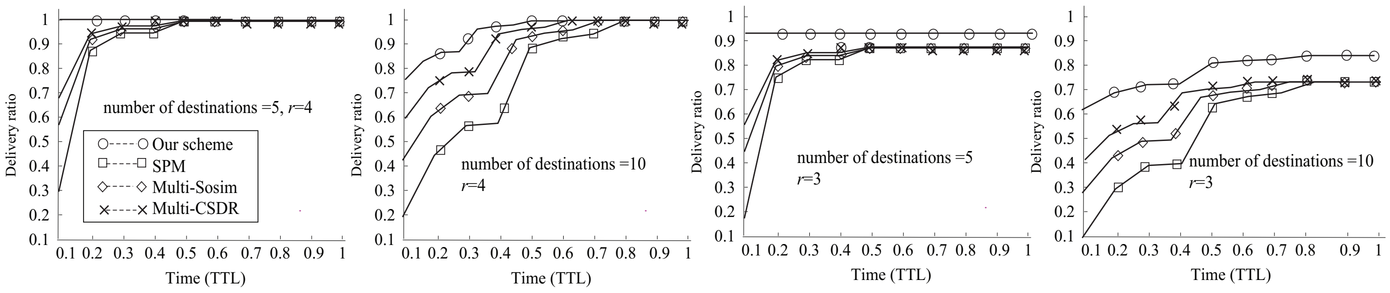

Fig. 5. Delivery ratios for SPM, Multi-Sosim, Multi-CSDR, and our scheme with respect to the packet TTL

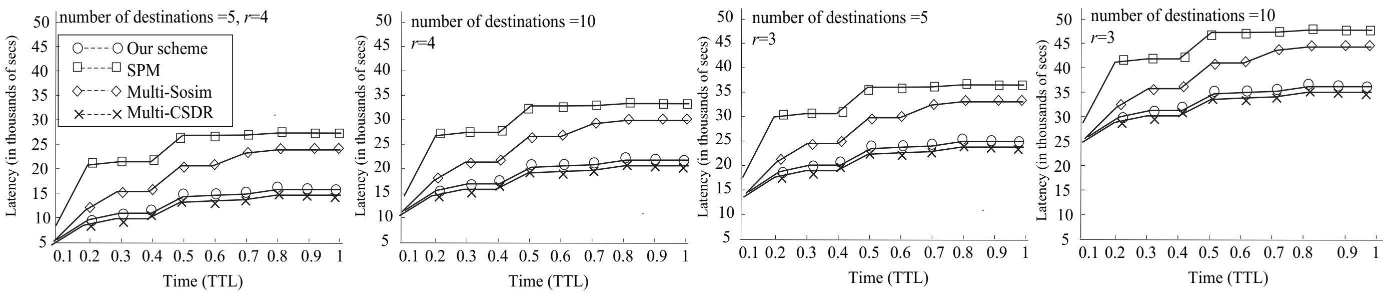

Fig. 6. Multicast latencies for SPM, Multi-Sosim, Multi-CSDR, and our scheme with respect to the packet Time to Live (TTL)

see that the SPM strategy has the highest latency and that the Multi-CSDR scheme outperforms Multi-Sosim, because it embodies the social relationships among destinations in the Multi-Sosim binary compare-split, thus facilitating multicast. The latency for our strategy higher, but very close to the one found for the Multi-CSDR scheme. This is due to the fact that our scheme guarantees of higher delivery rate, thus, it delivers more packets compared to the Multi-CSDR scheme, so it necessarily requires more time, especially in cases when the most lengthy paths of the greedy strategy are used. When r_{max} decreases from four to three, the performance of all strategies decreases because the relays may often need to wait for a proper neighbor (with social distance greater than 0.7) and they need to refresh their neighborhood information. In such cases, as we have seen from the previous set of simulations, the delivery ratio also decreases.

In the last set of simulations, we studied again the multicast latency, but in this case, we considered a set of three sources (new community members that use the existing links) trying to multicast to the same number of nodes (five or ten, with $r = 4$ and $r = 3$), using the same paths. In such a scenario, our pipeline-based scheme presented in Section (Case 2: Multicasting From Many Nodes to a Community Using the Same Data Paths) outperforms the Multi-CSDR scheme. Specifically, the latencies of the SPM, Multi-Sosim, and Multi-CSDR strategies are almost tripled (see Fig. 7, while our scheme retains a smooth increase of its latency, since a high number of relaying messages overlap.

VII. Conclusions - Future Work

In this paper we proposed a depth-first based community detection scheme, which generates multiple similarity paths in a stepwise and greedy manner, in its effort to detect possible overlaps. Then, we showed how to use these paths to generate multicast trees and how the cycles are eliminated from our multicast structures; to prove that they are indeed trees. Finally, two strategies were developed: one to multicast from one node to multiple receivers and one to multicast from multiple nodes to multiple receivers using the same paths. Simulation results showed that, the proposed community detection algorithm detects a little bit fewer but more strongly connected communities compared to the Conductance scheme, but more communities compared to the COPRA scheme. The multicasting scheme outperforms the comparable strategies as far as the delivery ratio is concerned, because it generates more paths available for multicasting at the expense of having a little

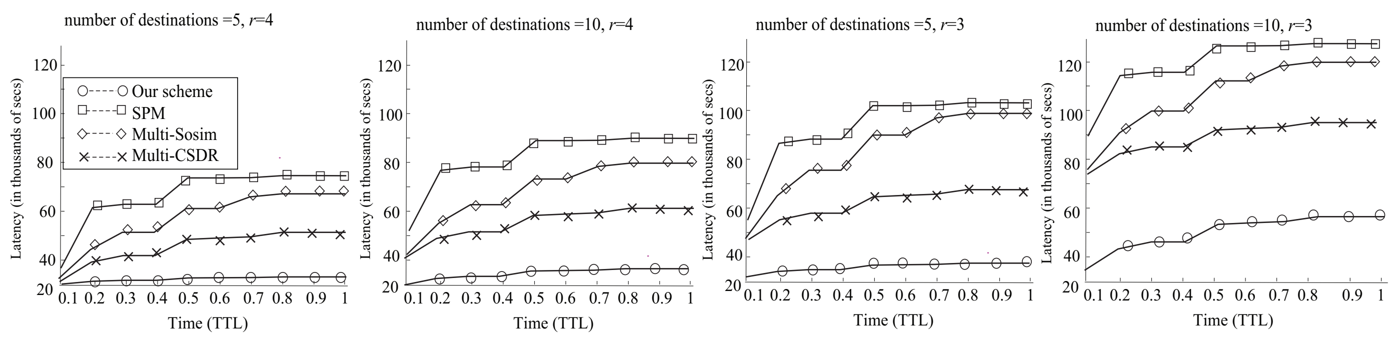

Fig. 7. Multicast latencies for SPM, Multi-Sosim, Multi-CSDR, and our scheme with respect to the packet TTL, with multiple sources sharing the same paths

higher latency (for the first strategy, multicast from one node to multiple receivers). Also, the second strategy (from multiple nodes to multiple receivers using the same path) is superior compared to the other schemes as far as latency is concerned, since it is pipeline-based, allowing for overlapping between forwardings on the same links.

In our future work, we plan to test the community detection and the multicasting scheme on a real network. Also, we need to incorporate the on-off model, in which some nodes may be temporarily off. One idea is to keep ordered lists based on the similarity of the paths and if a node or some nodes are off, choose the next best path, based on the similarity value. Finally, another research direction is to implement community-based data replication [42], [43].

References

[1] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, no. 6, p. 066111, 2004.

[2] Statista. (2017) Last time accessed on June 4th, 2017. [Online]. Available: http://www.statista.com/statistics/272014/global-social-networks-ranked by-number-of-users

[3] Z. Bu, C. Zhang, Z. Xia, and J. Wang, "A fast parallel modularity optimization algorithm (FPMQA) for community detection in online social network," *Knowledge-Based Systems*, vol. 50, pp. 246–259, 2013.

[4] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, 2010.

[5] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review E*, vol. 76, no. 3, p. 036106, 2007.

[6] Y. Baddi and M. D. E.-C. El Kettani, "A fast dynamic multicast tree adjustment protocol for mobile IPv6," in *5th IEEE International Conference on Next Generation Networks and Services (NGNS)*, Cassablanca, Morocco, 28-30 May 2014, pp. 106–113.

[7] Y.-H. Chen, E. H.-K. Wu, and G.-H. Chen, "Bandwidth-satisfied multicast by multiple trees and network coding in lossy manets," *to appear in IEEE Systems Journal*, 2015.

[8] X. Deng, L. Chang, J. Tao, J. Pan, and J. Wang, "Social profile-based multicast routing scheme for delay-tolerant networks," in *Proc. of IEEE International Conference on Communications*, Budapest, Hungary, 9-13 June 2013, pp. 1857–1861.

[9] H. Gong, L. Fu, X. Fu, L. Zhao, K. Wang, and X. Wang, "Distributed multicast tree construction in wireless sensor networks," *IEEE Transactions on Information Theory*, vol. 63, no. 1, pp. 280–296, 2017.

[10] Z. Lu, Y. Wen, and G. Cao, "Community detection in weighted networks: Algorithms and applications," in *Proc. of the IEEE International Conference on Pervasive Computing and Communications*, San Diego, California, USA, 18-22 March 2013, pp. 179–184.

[11] Y. Xu and X. Chen, "Social-similarity-based multicast algorithm in impromptu mobile social networks," in *Proc. of the IEEE Global Communications Conference*, Austin, TX USA, 8-12 December 2014, pp. 346–351.

[12] C. Shang, B. Wong, X. Chen, W. Li, and S. Oh, "Community and social feature-based multicast in opportunistic mobile social networks," in *Proc. of the 24th IEEE International Conference on Computer Communication and Networks*, Las Vegas, NV, USA, 3-6 August 2015, pp. 1–8.

[13] S. Souravlas and A. Sifaleras, "Binary-tree based estimation of file requests for efficient data replication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 1839–1852, 2017.

[14] H. Gong, L. Zhao, K. Wang, W. Wu, and X. Wang, "A distributed algorithm to construct multicast trees in WSNs: An approximate Steiner tree approach," in *Proc. of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Hangzhou, China, 22-25 June 2015, pp. 347–356.

[15] J. A. Sanchez, P. M. Ruiz, and I. Stojmnenovic, "GMR: Geographic multicast routing for wireless sensor networks," in *Proc. of the 3rd Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, vol. 1, Reston, VA, USA, 25-28 September 2006, pp. 20–29.

[16] J. Matsumoto, "Multicast tree construction algorithm for stabilization of power quality in smart grid," in *Proc. of the 17th IEEE International Conference on High Performance Switching and Routing*, Yokohama, Japan, 14-17 June 2016, pp. 122–127.

[17] X. Zheng, C. Cho, and Y. Xia, "Algorithms and stability analysis for content distribution over multiple multicast trees," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1217–1227, 2015.

[18] D. Chen, M. Shang, Z. Lv, and Y. Fu, "Detecting overlapping communities of weighted networks via a local algorithm," *Physica A: Statistical Mechanics and its Applications*, vol. 389, no. 19, pp. 4177–4187, 2010.

[19] W. Gao, Q. Li, B. Zhao, and G. Cao, "Social-aware multicast in disruption-tolerant networks," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1553–1566, 2012.

[20] W. Gao, G. Cao, T. La Porta, and J. Han, "On exploiting transient social contact patterns for data forwarding in delay-tolerant networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 1, pp. 151–165, 2013.

[21] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: Social-based forwarding in delay-tolerant networks," *IEEE Transactions on Mobile Computing*, vol. 10, no. 11, pp. 1576–1589, 2011.

[22] Z. Bu, Z. Xia, and J. Wang, "A sock puppet detection algorithm on virtual spaces," *Knowledge-Based Systems*, vol. 37, pp. 366–377, 2013.

[23] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review E*, vol. 69, no. 2, p. 026113, 2004.

[24] M. E. Newman, "Analysis of weighted networks," *Physical Review E*, vol. 70, no. 5, p. 056131, 2004.

[25] J. Leskovec, K. J. Lang, and M. Mahoney, "Empirical comparison of algorithms for network community detection," in *Proc. of the 19th ACM International Conference on World Wide Web*, Raleigh, North Carolina, USA, 26-30 April 2010, pp. 631–640.

[26] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Physical Review E*, vol. 80, no. 1, p. 016118, 2009.

[27] J. W. Berry, B. Hendrickson, R. A. LaViolette, and C. A. Phillips, "Tolerating the community detection resolution limit with edge weighting," *Physical Review E*, vol. 83, no. 5, p. 056119, 2011.

[28] S. Gregory, "Finding overlapping communities in networks by label propagation," *New Journal of Physics*, vol. 12, no. 10, p. 103018, 2010.

[29] N. P. Nguyen, T. N. Dinh, S. Tokala, and M. T. Thai, "Overlapping communities in dynamic networks: Their detection and mobile applica-

tions," in *Proc. of the 17th Annual ACM International Conference on Mobile Computing and Networking*, Las Vegas, Nevada, USA, 19-23 September 2011, pp. 85–96.

[30] N. P. Nguyen, T. N. Dinh, Y. Xuan, and M. T. Thai, "Adaptive algorithms for detecting community structure in dynamic social networks," in *Proc. of the 30th IEEE International Conference on Computer Communications*, Shanghai, China, 10-15 April 2011, pp. 2282–2290.

[31] J. Xie, S. Kelley, and B. K. Szymanski, "Overlapping community detection in networks: The state-of-the-art and comparative study," *ACM Computing Surveys*, vol. 45, no. 4, 2013, Article No. 43.

[32] A. Lancichinetti, S. Fortunato, and J. Kertész, "Detecting the overlapping and hierarchical community structure in complex networks," *New Journal of Physics*, vol. 11, no. 3, p. 033015, 2009.

[33] Z. Lu, X. Sun, Y. Wen, G. Cao, and T. La Porta, "Algorithms and applications for community detection in weighted networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 11, pp. 2916–2926, 2015.

[34] I. Farkas, D. Ábel, G. Palla, and T. Vicsek, "Weighted network modules," *New Journal of Physics*, vol. 9, no. 6, p. 180, 2007.

[35] J. M. Kumpula, M. Kivelä, K. Kaski, and J. Saramäki, "Sequential algorithm for fast clique percolation," *Physical Review E*, vol. 78, no. 2, p. 026109, 2008.

[36] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, "Link communities reveal multiscale complexity in networks." *Nature*, vol. 466, no. 7307, pp. 761–764, 2010.

[37] A. Padrol-Sureda, G. Perarnau-Llobet, J. Pfeifle, and V. Muntés-Mulero, "Overlapping community search for social networks," in *Proc. of the 26th IEEE International Conference on Data Engineering (ICDE)*, 2010, pp. 992–995.

[38] R. Cazabet, F. Amblard, and C. Hanachi, "Detection of overlapping communities in dynamical social networks," in *Proc. of the 2nd IEEE International Conference on Social Computing (SocialCom)*, 2010, pp. 309–314.

[39] S. Souravlas and A. Sifaleras, "On the detection of overlapped network communities via weight redistributions," in *Advances in Experimental Medicine and Biology*, P. Vlamos, Ed. Springer, Cham, 2017, vol. 988, pp. 205–214.

[40] A. Peiravi and H. T. Kheibari, "A fast algorithm for connectivity graph approximation using modified manhattan distance in dynamic networks," *Applied Mathematics and Computation*, vol. 201, no. 1, pp. 319–332, 2008.

[41] V. Labatut, "Generalised measures for the evaluation of community detection methods," *International Journal of Social Network Mining*, vol. 2, no. 1, pp. 44–63, 2015.

[42] S. Souravlas and A. Sifaleras, "On minimizing memory and computation overheads for binary-tree based data replication," in *22nd IEEE Symposium on Computers and Communications (ISCC 2017)*, Heraklion, Greece, 3-6 July 2017, pp. 1296–1299.

[43] ——, "Trends in data replication strategies: a survey," *International Journal of Parallel, Emergent and Distributed Systems (to appear)*, 2017.

Stavros Souravlas received the PhD degree in Computer Science from the University of Macedonia, Thessaloniki, Greece. He is currently an Assistant Professor of Computer Architecture and Digital Logic Design at the Department of Applied Informatics, School of Information Sciences, University of Macedonia, where he joined in 2014. His research interests include computer architecture and performance evaluation, parallel and distributed systems, grid computing, cloud computing, systems modeling and simulation. He is a member of the IEEE.

Angelo Sifaleras is an Assistant Professor at the University of Macedonia, Department of Applied Informatics, School of Information Sciences, Greece, Thessaloniki. His research focuses on mathematical programming and network optimization and he is also a senior member of ACM.