Distributed Current Flow Betweenness Centrality

CONFERENCE PAPER · JANUARY 2015

READS
22

4 AUTHORS:

Alessandro Lulli
Università di Pisa
6 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Laura Ricci
Università di Pisa
70 PUBLICATIONS **192** CITATIONS
SEE PROFILE

Emanuele Carlini
Italian National Research Council
27 PUBLICATIONS **68** CITATIONS
SEE PROFILE

Patrizio Dazzi
Italian National Research Council
56 PUBLICATIONS **267** CITATIONS
SEE PROFILE

Distributed Current Flow Betweenness Centrality

Alessandro Lulli*†, Laura Ricci*†, Emanuele Carlini†, Patrizio Dazzi†
*University of Pisa, Italy
{surname}@di.unipi.it
†ISTI, CNR, Pisa, Italy
{name.surname}@isti.cnr.it

***Abstract*—The computation of nodes centrality is of great importance for the analysis of graphs. The *current flow betweenness* is an interesting centrality index that is computed by considering how the information travels along all the possible paths of a graph. The current flow betweenness exploits basic results from electrical circuits, i.e. Kirchhoff's laws, to evaluate the centrality of vertices. The computation of the current flow betweenness may exceed the computational capability of a single machine for very large graphs composed by millions of nodes. In this paper we propose a solution that estimates the current flow betweenness in a distributed setting, by defining a vertex-centric, gossip-based algorithm. Each node, relying on its local information, in a self-adaptive way generates new flows to improve the betweenness of all the nodes of the graph. Our experimental evaluation shows that our proposal achieves high correlation with the exact current flow betweenness, and provides a good centrality measure for large graphs.**

I. Introduction

In the last years the broad diffusion of the internet and, more in general, the communication networks, allowed people, information, computing devices and even simpler "things" to be always reachable, according to the "always-on" paradigm. Operatively, this world of connections is often supported by social networks and/or the underlying physical communication networks. The web of interactions among the entities composing such environments can be even more interesting; both from the amount and richness of information viewpoints. Such network of interactions is often represented as a graph, that if analysed can provide useful information, both regarding the graph as a whole or regarding specific vertices. Specifically, centrality measures are an important tool in graph analysis, as they provide information on the structural prominence of nodes and edges and, as a consequence, they support the identification of the key elements of the network. For instance, in a social network it could be interesting to find the most important actors in large social interaction graphs; or in a data network could be worth to find the nodes that are subject to the highest traffic to prevent network congestion and disruption.

Over the years, many different centrality measures have been proposed. One of the most popular measures is the vertex *betweenness centrality*. The Betweenness Centrality [1] measures the importance of a node as the number of times it lies on the shortest path between two other nodes. It is an appropriate index in networks where information flows mostly along geodesic paths, and it is currently adopted for the analysis of a wide set of complex networks. A common limit of shortest-paths based measures is that they do not take into account the information spread occurring along non-shortest paths, hence, they are not the best choice when information conveying is governed by other rules. To overcome this limitation, in literature have been proposed betweenness measures that are based on the information flow along the graph.

Freeman *et al.* [2] suggested a sophisticated betweenness measure, the *flow betweenness*, that includes contributions from some non-geodesic paths. The flow betweenness is computed by considering a network where a maximal amount of information (with respect to edges capacity) is continuously injected between all the source vertices and the target vertices. However, using this measure there might be paths that are central in the networks, but are not crossed by any flow unit for particular pairs of sources and targets. To address these issues, Newman [3] and Brandes et al. [4] proposed to model the network as a electric circuit where a current flow is injected to a source node and exits to the target node. The resulting index, the *current flow betweenness* named also *random-walk betweenness* is able to compute contributions from all paths existing between the source and the target node. The current state-of-the-art for the computation of the current flow betweenness includes both exact [5], [4] and approximated [4], [6], [7] approaches. However, these solutions are all centralized, and their applicability is limited only to small graphs.

To overcome this limitation, this paper proposes DUCKWEED, a novel algorithm for the distributed computation of the approximated current flow betweenness centrality. In DUCKWEED, each node, using only locally available information, exploits, in a self-adaptive way, the Kirchhoff's laws of electrical circuits to incrementally compute the current flow betweenness of the graph. Our algorithm can be implemented in both distributed frameworks for the analysis of large graphs (such as Apache Spark [8] or Hadoop [9]) and in peer-to-peer networks.

The paper is organised as follows. Section II reports an analysis of the related work. Section III presents some preliminary notions, focusing in detail on random walks and current flow betweenness. Then, in Section IV, is present DUCKWEED, our proposed solution for the distributed estimation of current flow betweenness. In Section V are reported our empirical findings. Finally, Section VI reports our conclusions.

II. Related work

The computation of the betweenness centrality index is intrinsically expensive. In fact, to be determined it requires the computation of all the shortest paths. The naive centralized algorithm requires $\theta(n^3)$ time and $\theta(n^2)$ space, where n is the number of vertices. An improvement to the basic solution has been proposed by Brandes [10] in 2001. In his proposal, he introduces the notion of vertex dependency, recursively defined on the whole structure of the graph. Following this approach it is no longer required the combinatorial counting of all paths and $O(n+m)$ space and $O(nm)$ time bounds, respectively, are obtained for undirected graphs (with m the number of edges). Another approach is based on the definition of algorithms for computing approximated values of the index. Riondato *et al.* [11] proposed two efficient randomized algorithms for the estimation of betweenness based on random sampling. These algorithms offer probabilistic guarantees on the quality of the approximation. To bound the size of the sample exploited to achieve their approximated result, they rely on the results from the Vapnik-Chervonenkis theory, which allows to use small sample sizes.

The current flow betweenness centrality has gained momentum in the last years as an alternative index to measure centrality of nodes in a graph. Similarly to the classical betweenness centrality, the straightforward algorithm to determine current flow centrality is to compute information flows for all the possible pairs of node in the graphs. Newman [3] and Brandes *et al.* [4] provided a formulation derived from Kirchoff's law of current conservation, in which edges are resistors with a given conductance, and the nodes are junctions between resistors. They propose an algorithm for estimating the current flow betweenness centrality having a computational complexity of $O(I(n-1)+mn^2)$, where $O(I(n-1))$ is the complexity to invert a $n \times n$ matrix. Moreover, they propose an approximated version of the algorithm that selects uniformly at random a small fraction of all pairs $s \neq t \in V$. The bound on the number of pairs is computed by exploiting the Hoeffding bound [12].

Bozzo and Franceschet [6] [13] proposed an algorithm for current flow betweenness centrality that reduces the complexity deriving from the inversion the Laplacian matrix describing the graph by choosing a subset of its eigenvalues and eigenvectors. Avrachenkov *et al.* [7] introduce the α-current flow betweenness centrality by adding a "ground node" to the original graph, and connecting each node to it. The introduction of the ground node simplifies the computation and leads to a reduction of the complexity of the matrix calculations. In addition, they approximate the betweenness by randomly selecting a subset of all possible pairs, similarly to Brandes *et al.* [4]. They also introduce the truncated α-current flow betweenness, an approach in which the scores on the edges starting from the source of the flow are not considered in the computation of the betweenness. They proved empirically that the resulting estimation increases the correlation with the exact current flow betweenness.

Despite the optimizations introduced by the solutions mentioned above, the computational cost to determine path-based centrality indices requires the adoption of distributed solutions when dealing with large graphs. Lehmann and Kaufmann [14] propose a general framework for defining decentralized algorithms that compute centrality indices. To this end they take into account four different centralities, closeness, stress, graph and betweenness, with emphasis on the betweenness centrality.

As far as we know, there is no distributed algorithm for the current flow betweenness centrality. Rather, novel definitions of the centrality indices have been proposed, which are suitable to be computed in a distributed environment. Wehmuth and Ziviani [15] redefined the closeness centrality index of a vertex by considering its h-neighbourhood, i.e., the vertices within a radius h around it. The centrality value of each vertex in the network is defined as the sum of the degrees of the vertices in its h-neighbourhood, i.e., the volume of its h-neighbourhood. This notion of centrality is equivalent to the degree centrality when $h = 0$. The distributed algorithm consists in a TTL-restricted flooding of the degree of each node within its h-neighbourhood. The experimental results show a high degree of correlation between this notion of centrality and the closeness centrality. Kermarrec *et al.* [16] introduces a novel notion of centrality based on random walks, called *second order centrality*. According to it, each node computes its centrality index by starting an unbiased random walker and counting how much time the walker requires to return to the node. Their approach collects a certain number of this measure in order to calculate the standard deviation. This calculation is performed incrementally each time the walker returns to the node, improving the estimation of the centrality of the nodes over time.

III. Preliminaries

In this section we introduce the notions required for the definition of the main theoretical framework at the basis of our approach. We first introduce the Random Walk Betweenness and then we present the relation between elementary electric network theory and random walks, which is at the basis of the definition of the Current Flow Betweenness.

A. Random Walk Betweenness

Let us consider a network and suppose that a node s generates an information (conveyed as a message) whose target is node t. Each node receiving the message propagates it to one of its neighbours, chosen uniformly at random. This strategy is usually referred as random walk. The notion of Random Walk Betweenness Centrality introduced by Newman [5] measures, for a given a node n, the expected net number of times a random walker passes through n on its way from s to t, averaged on all s and t. As Newman states, when computing the net number of times, two visits of the walker to the same vertex coming from opposite directions must be cancelled out. This avoid scenarios where the walker passes forth and back a vertex many times, without actually going anywhere.

B. Current Flow Betweenness

Doyle and Snell [17] give an exhaustive presentation of the relations existing between random walks end electric networks. We briefly summarize the main results of their work in this section.

Let $G = (V, E)$ be an undirected graph, where $E \subseteq V \times V$, with $n = |V|$ vertices and $m = |E|$ edges. We assume that there are no self-loops from one vertex to itself and no pairs of vertices connected by multiple edges. Each edge $(i, j) \in E$ connecting vertices i and j has a weight $w_{i,j}$. The matrix $A_{i,j}$ is the adjacency matrix of the graph, i.e. $A_{i,j} = 1$ if and only if there exist an edge connecting vertices i and j.

An electrical network may be represented by a graph by assigning to each edge a positive weight indicating the conductance of the corresponding electric wire (or the resistance of the wire, which is the inverse of the conductance). According to this representation, the vertices of the graph are junctions between resistors. For the sake of simplicity, we consider unitary conductance (or resistance), i.e., $w_{i,j} = 1$, $\forall i, j$.

In particular, we are interested in how current flows through the network, when injected to a source node s and picked up at a target node t, for all possible choices of s and t.

Definition 1. We define a current flow $F^{(s,t)}$ over the graph G as follows:

- s is the source of the flow, the current enters the network through it;
- t is the target of the flow, the current leaves the network through it;
- $u_i^{(s,t)}$, called supply vector, is a vector such that $\sum_i u_i = 0$ and $u_s = -u_t = 1$:

Let $v_i^{(s,t)}$ be the potential at node i for $F^{(s,t)}$. Kirchhoff's law of current conservation states that the current that enters into a node is equal to the current that flows out of it. This implies that the potentials of a node satisfy the following equation for every node i:

$$\sum_j A_{ij}(v_i^{(s,t)} - v_j^{(s,t)}) = u_i^{(s,t)} \tag{1}$$

$$v_i^{(s,t)} = \frac{\sum_j A_{ij} v_j^{(s,t)} + u_i^{(s,t)}}{\sum_j A_{ij}} \tag{2}$$

By considering only unitary resistances as stated above, we have $A_{ij} = 1$ if exists the edge (i, j), otherwise 0 and that $\sum_j A_{ij} = \deg(i)$.

Given the above, let us consider the current flow that passes through the vertex when a unit of current is injected in a source vertex and removed from a target vertex, averaged over all source-target pairs [4], [5]. Doyle *et al.* [17] show that this current flow is equal to the net expected number of times that a walker, starting at s and walking until it reaches t, will pass through that vertex. The random walker betweenness can be therefore computed by considering the electrical circuit associated to the graph and by applying the laws of electrical circuits. The resulting betweenness index is also referred as Current Flow Betweenness.

To compute the current flow betweenness of a vertex i, $\neq s, t$, it is therefore required to compute the current flowing through i, which, given a flow $F^{(s,t)}$, is defined as half of the sum of the absolute values of the currents flowing along the edges incident on that vertex:

$$I_i^{(s,t)} = \frac{1}{2} \sum_j A_{i,j} \mid v_i^{(s,t)} - v_j^{(s,t)} \mid \tag{3}$$

The current-flow betweenness centrality b_i [4], [5] is the average of the current flows over all the source-target pairs:

$$b_i = \frac{\sum_{s<t} I_i^{(s,t)}}{(1/2)n(n-1)} \tag{4}$$

Bozzo *et al.* [13] show how to solve the system of equations 2 by exploiting classical matrix calculus when considering all the possible source and target pairs. However, when the graph is very large, this calculus may exceed the computational capability of a single machine. On the other hand, we note that a distributed computation of the system of equations 2 is feasible, because each node requires only local information, i.e. information about its neighbours, to iteratively compute the value of its potential for a given s and t pair, and calculates its betweenness by means of its currents flowing according to Equation 3. This observation is at the base of the distributed approach we present in the next section.

IV. DUCKWEED

As far as we know, current state-of-the-art approaches for current flow betweenness centrality (CFBTW) are centralized, and their applicability is limited to small graphs [4], [7]. In this work we present our proposal to fill this gap. We propose DUCKWEED, a novel distributed approach that computes an estimation of the current flow betweenness centrality, which is suitable for large graphs. Approximate CFBTW has a practical applicability, essentially for two reasons: (i) computing a precise CFBTW for a large graph is very time consuming, and (ii) many applications of centrality indexes require to find a ranking of the top-most central nodes, rather than the exact values of centrality for each vertex.

A. Computational Model

The computation performed by DUCKWEED follows a vertex-centric approach in which the nodes of the graph are considered as the unity of computation. We assume that nodes of the graph are processed periodically and asynchronously. Nodes do not have access to the entire graph, but only to their immediate neighbours in the graph, and to a small set of other nodes, provided by means of information diffusion protocols. DUCKWEED does not require any kind of shared memory, as nodes communicate only through explicit messages. This makes DUCKWEED suitable for graph computing frameworks, and for peer-to-peer networks, in case each node is assigned to a peer of the network.

Algorithm 1: Flow Computation

Data: F: the set of flows known by the node

```
R ← receive flows from neighbours
N ← flowCreation()
F ← F ∪ R ∪ N
forall the f ∈ F do
    update potential for f
    Δf ← difference with potential of the previous iteration;
    if Δf < 𝒟_ε then
        mark f as completed
    end
end
send F to neighbours
CentralityComputation()
F ← F \ {f is completed}
```

B. *The* DUCKWEED *approach*

The main idea behind DUCKWEED is to exploit Kirchhoff's law to calculate the electric potentials of all the vertices, as building blocks for the computation of CFBTW for the entire graph. The computation is organized in such a way that each node, locally and autonomously, can compute its own electric potential and its own value of CFBTW centrality. DUCKWEED is based on the following two main modules: the **Flow computation** and the **Centrality computation**.

The Flow computation drives the creation of flows and the computation of potentials. In particular, for a given $F^{(s,t)}$, the corresponding potential can be computed using Equation 2 and exploiting only the knowledge about the neighbours potentials (due to the fact that the conductance between two nodes is zero if they are not neighbours, see considerations in Section III). The detailed description of the Flow computation is given in Section IV-C.

For Centrality computation, each node collects the potentials of its neighbours to compute independently the actual values of the CFBTW. In particular, to compute the incremental CFBTW for a vertex i, Centrality computation exploits Equations 3 and 4 in the following way:

$$b_i^k = \frac{(k-1)b_i^{k-1} + I_i^{(s,t)}}{k} \tag{5}$$

where k is the amount of computed flows known by the vertex. The fact that DUCKWEED computes the CFBTW incrementally yields two relevant impacts. First, it is possible to have an idea of what are the most central nodes without waiting for the computation to be completed. Second, it is possible to define an automatic mechanism of termination, so that DUCKWEED stops the computation when it reaches a given level of approximation. A detailed description of the Centrality computation is provided in Section IV-D.

C. *Flow computation*

This section describes in details how the computation of a single generic flow $F^{(s,t)}$ is realized in DUCKWEED. The computation of a single flow is then extended to the concurrent computation of n flows, considering that each flow $F^{(s,t)}$ can be identified uniquely using its source and target vertex, respectively s and t, with $s < t$.

Algorithm 1 shows the pseudo code of the Flow computation in a node. The node first receives the information about the flows computed at the previous iteration by its neighbours, and then, if necessary, creates new flows. Subsequently, it updates the potential relative to each flow and marks a flow as terminated if the changing in potential is under the threshold $\mathscr{D}_\epsilon$. Further, the node sends the set of updated flows to its neighbours, it updates its current flow betweenness centrality value, and removes the completed flows from the local state. For the sake of the explanation, we divide the Flow computation into three main steps: *creation*, *update*, and *termination*. In the following we describe these three steps.

1) Creation: The creation of a flow $F^{(s,t)}$ is done locally by each node, by considering itself as the source s and choosing a target t among the other nodes. Since the algorithm is fully distributed, deciding *when* to start a new flow, and *which* node to choose as target are relevant aspects.

To define when a node shall create a flow, we designed DUCKWEED such that the number of concurrent flows in the network are probabilistically limited in any given point in time, so to avoid to oversaturate the nodes. To this end, we define the system-wide parameter φ as the number of maximum concurrent flows active on each node; the number of flows active on a generic node i is defined as φ_i. In addition, to avoid the re-creation of an already computed flow, each vertex u maintains a list of all the flows processed for which u is the source or the target of the flow.

The flow creation relies on a combination of different gossip-like protocols, organized in layers (as usual), to realize the distributed computation of the flows. These protocols are popular in peer-to-peer networks, as they proved to be efficient solutions to tackle very different problems, ranging from distributed data clustering [18], [19], [20], [21], resource and service discovery [22], [23], online games [24]. Moreover, they can also be implemented and exploited in graph computing frameworks [25]. These protocols are based on a *random node sampling* layer that provides a selection of random nodes from the graph, similarly to the random peer sampling gossip protocols used in distributed applications [26], [27]. In addition, the Flow computation exploits the following protocols: (i) the *size estimator* implements a well-know protocol to count the number of peers in a gossip fashion scenario [28], [29]; note that, at any given time, a node has its own estimation of the graph size, which in general is different from the one of the other nodes. In the following, we refer to the size estimation of node i as n_i. (ii) The *average betweenness* provides an estimation of the average CFBTW of the whole graph. This protocol is similar to the *size estimator* protocols, and we refer to the CFBTW average estimation of node i as BTW_i. (iii) The *k-partitioning* performs a distributed k-way partitioning on input graph, similar to the one proposed in [30]. The result of the partitioning is to colour the nodes of the graph with different colours according to their partition. The *k-partitioning* and *average betweenness* are optional, as they depends on the particular strategy used for the generation of the flow.

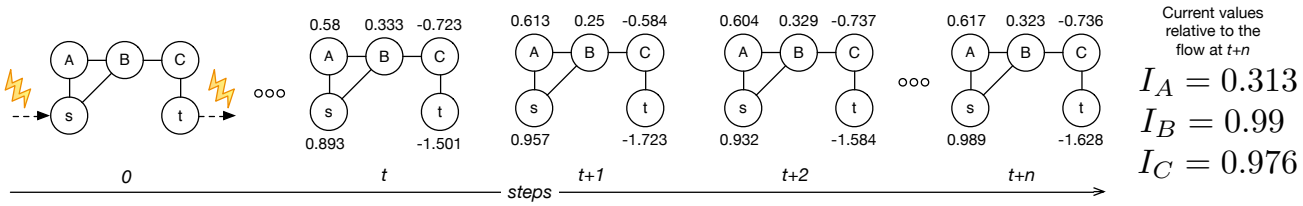


Fig. 1: Evolution of a flow from s to t over time

Using these concepts, we defined three different strategies to generate a new flow:

- *random*: a node generates a new flow $F^{(s,t)}$ with a certain probability p_{random}, computed as the following:

$$p_{random} = max\left(0, \frac{\varphi - \varphi_i}{n}\right) \tag{6}$$

  where n is the number of (estimated) nodes in the graph. If the node is supposed to create a flow, selects t from its random sampling view.
- *adaptive*: the aim of this strategy is to favour the creation of flows between nodes that are at the border of the graph, and to disfavour flow betweens nodes in the center. Recall that to be source (or target) of a flow, do not improve its own betweenness value. In other words, this strategy tries to accelerate the computation of the nodes with already an higher betweenness. To implement this strategy, a generic node i considers the local estimation of the average CFBTW for the whole graph (BTW_i). If i has its current centrality value larger than BTW_i, then it generates no flow. Otherwise, it creates a flow with probability $p_{adaptive}$, defined as the following:

$$p_{adaptive} = max\left(0, \frac{\varphi - \varphi_i}{n/2}\right) \tag{7}$$

  Note that $p_{adaptive}$ is twice p_{random}, so to compensate the node with no chance to create a flow. A node u elected to generate a new flow sets $s = u$ and t equals to the node having the smallest CFBTW value in its random sampling view.
- *partitioner*: the aim of this strategy is to favour the generation of flows with s and t being in different partitions of the graph. Intuitively, nodes on the path between two nodes of different partitions are more likely to have higher betweenness centrality. A node u generates a new flow with a probability equals to p_{random} and $s = u$, and exploits the colouring from the *k-partitioning* protocol to choose t from the random sampling view, such as t's colour is different from s's colour.

2) Update: During every step, each vertex collects all the flows received by its neighbourhood in the previous step and applies Equation 2 to update the potential of each flow. The updated potentials, along with the flow identifiers, are then sent to all the neighbourhood to continue the computation in the next step. This computation can be easily extended to handle weighted graphs introducing the weights in Equation 2 as described by Bozzo et al [13] using as A the weighted adjacency matrix.

Figure 1 depicts a flow computation where a unit of current is injected in node s and removed from node t. In step $t+1$ the node B receives from its neighbours $\{A, C, s\}$ the potentials calculated on step t equals to $\{0.62, -0.73, 0.94\}$. It applies Equation 2 and it obtains a potential equals to 0.28. This potential will be available to its neighbours in the following step.

3) Termination: The termination of a flow is regulated by the system-wide parameter $\mathscr{D}_\epsilon$, which represent the minimum difference in the potential of a flow between two consecutive steps. Therefore, on a generic vertex i, the flow $F^{(s,t)}$ is completed when both the following conditions are verified:

- $v_i^{(s,t)}$ has converged to at least $\mathscr{D}_\epsilon$;
- all the neighbours of i have converged to at least $\mathscr{D}_\epsilon$.

When a flow is marked as completed, the vertex stops the propagation of the potential relative to such flow. This eventually terminates the update of the flow $F^{(s,t)}$ in all the vertices of the graph.

D. Centrality computation

In the previous section, we described how the computation of a single flow is performed in DUCKWEED. This section describes how DUCKWEED combines the results coming from the computation of multiple flows to incrementally compute the CFBTW for the nodes of the graph.

When the computation of a generic flow $F^{(s,t)}$ is completed (as described in Section IV-C2), vertex i uses Equation 3 to compute $I_i^{(s,t)}$ for the $F^{(s,t)}$. Vertex i then updates its current-flow betweenness b_i using $I_i^{(s,t)}$ and Equation 5. Note that the source s and target t do not consider $I_s^{(s,t)}$ and $I_t^{(s,t)}$ for their betweenness calculation. Since we provide an estimation of the CFBTW by computing only a subset of the all possible flows, this assumption is required to avoid biased computation of centrality, which would occur if considering any value for $I_s^{(s,t)}$ and $I_t^{(s,t)}$, for example equals to 0 or 1 as suggested in [6]. Figure 1 (on the right) show the current flowing on each node of the graph calculated with Equation 3 when the flow is completed at step $t + n$. Considering node B we obtained:

$$\begin{aligned} I_B &= \frac{1}{2}(|0.323 - 0.617| + |0.323 - 0.949| + |0.323 + 0.736|) \\ &= 0.99 \end{aligned} \tag{8}$$

This value will be used by node B in Equation 5 to incrementally calculate its current flow betweenness value.

The precision of the CFBTW increases as more and more flows are updated and completed. To detect when the results of the CFBTW are precise enough, and hence to terminate the generation of new flows, it is possible to define a parameter k to stop the generation of new flows when all the nodes have already processed k flows. Otherwise it is possible to introduce more sophisticated techniques to let each node decide autonomously when stopping flow generation and hence terminate the computation. Note that the termination of the computation of the CFBTW is *different* from the termination of a single flow. In the former case we refer to the termination of the whole system (i.e. no more flows are created), in the latter (described in Section IV-C3) we refer to the termination of a *single* flow.

In order to equip each node with the possibility of locally terminating the computation of the CFBTW, we adapt the centralized mechanism introduced by Brandes et al. [4] to a distribute context. This method exploits the Hoeffdings bound [31] to identify a value of k such that the error ϵ_{BTW} on the betweenness values b_v on each node is sufficiently small. Hoeffding's bound gives:

$$\mathbb{P}\left(\left|\frac{c^*}{k}\sum_{i=1}^{k} X_v^{(i)} - b_v\right| \geq \epsilon_{BTW}\right) \leq \frac{2}{n^{2l}} \tag{9}$$

when choosing:

$$k = l \cdot \lceil (c^*/\epsilon_{BTW})^2 \log_n \rceil \tag{10}$$

for arbitrary l, where $c^* = n/(n-2)$ and $X_v^{(1)}, ..., X_v^{(k)}$ are independent random variables that return $F^{(s,t)}$, for a pair $s \neq t$, picked uniformly at random. We adapted the above calculation to be suited for the distributed environment of DUCKWEED. To pick a flow uniformly at random we exploited a random peer sampling layer and each node, for instance using the random flow creation heuristic, has the same probability to generate a flow with a node taken from the random peer sampling. Also, each node has its estimation k_i about the number of flows already computed in the system equals to the number of flows that i has computed. The size of the network n can be estimated in the same way explained in Section IV-C1. Given the above, each node can autonomously calculate k and stop the flow generation when $k_i > k$.

V. Experimental evaluation

The aim of the evaluation is to verify the effectiveness of DUCKWEED both in terms of the quality of results, and on its applicability on large graphs. The evaluation of DUCKWEED was conducted by means of simulations. We implemented DUCKWEED, and all the associated protocols, on the discrete-event PeerSim [32] simulator.

A. Evaluation of Correlation

To conduct an evaluation of DUCKWEED we measured the correlation of the results provided by our approach with the ones given by the other algorithms. To measure the correlation, we use the KENDALL TAU metrics, which is a measure of rank correlation. It measures the degree of similarity between two distinct rankings by assigning a value in the range $[-1, 1]$. If two rankings have the same values, the coefficient equals to 1, whereas if the disagreement between the two rankings is perfect (i.e., one ranking is the reverse of the other) the coefficient has value -1. For this evaluation we generated 3 graphs of 1000 nodes each, using the Snap library [33] with the following strategies: preferential attachment (Barabasi-Albert), random (Erdos-Renyi) and RMAT[1] [34]. These graphs are purposely small to ease the computation of exact values for all the centrality measures.

1) Validation against NetworkX: In this first set of experiments we validate the precision of the current flow betweenness centrality provided by DUCKWEED against the one provided by NetworkX [35]. NetworkX is a popular tool to analyse network structure and it provides an implementation of the algorithm presented by Brandes *et al.* [4] to calculate the current flow betweenness centrality. We run NetworkX on the three aforementioned graphs to compute the exact CFBTW. Then, we run DUCKWEED on the same graphs, computing all flows and varying the decimal precision of $\mathscr{D}_\epsilon$ up to five decimals. We compared the *top* 100 nodes of NetworkX and DUCKWEED with the KENDALL TAU metrics. It is worth to point out that in the context of all our experiments we refer to the *top X* nodes to indicate the *X* nodes that received the highest value of centrality according to a given measure.

Results are presented in Figure 2. It is evident that in all datasets the correlation increases when increasing the decimal precision. This is an expected result because increasing the precision on each flow leads to a more precise calculation of the current flow betweenness. The value of centrality, computed by DUCKWEED on the preferential attachment graph, exhibits the best correlation in all the configurations. In fact, even with a single-decimal precision the value of its correlation with NetworkX is around 0.9. Conversely, the random graph exhibits a correlation of 0.7 when adopting single-decimal precision, however the correlation value rapidly increases when using additional decimal precision. It is worth to notice that with a precision of 4 decimals, all the datasets exhibits a correlation value greater than 0.9, suggesting that DUCKWEED correctly approximates the current flow betweenness centrality also with a reasonable value of $\mathscr{D}_\epsilon$.

2) DUCKWEED *complete vs approximated:* This test compares the complete version of DUCKWEED (i.e. the one in which all flows are computed), against the approximated version, in which only a portion of all the flows is computed. The results presented in Figure 3 show the KENDALL TAU correlation of the top 100 nodes between the complete and the approximated version of DUCKWEED, when the amount of flows executed varies.

The experimental results show that the correlation increases quite fast becoming closer to the complete one, even using a

[1]RMAT graphs were generated with parameters $(.6, .1, .15, .15)$

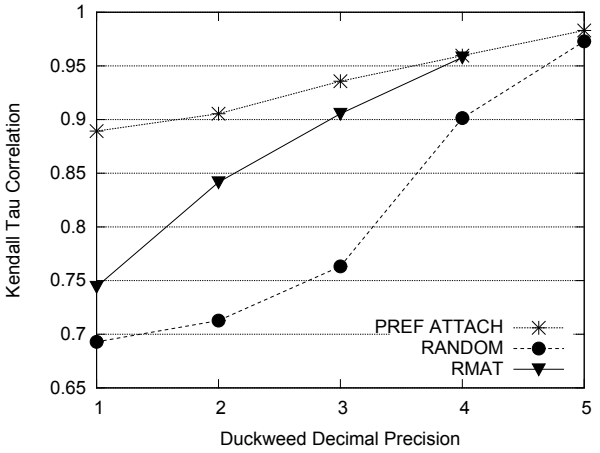


Fig. 2: Kendall Tau Correlation with NetworkX

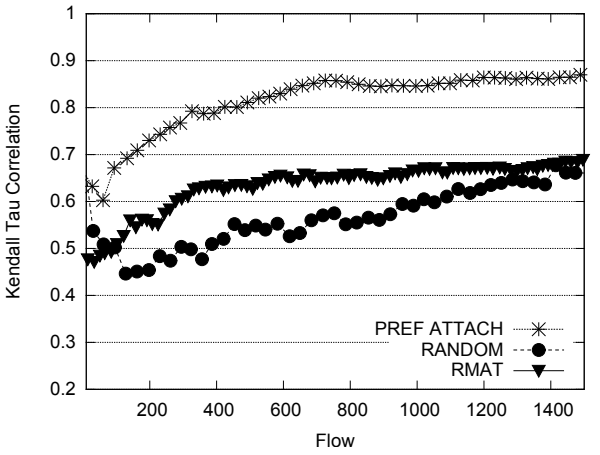


Fig. 3: Kendall Tau Correlation: complete vs approximated

number of flows that represents less than the 1% of all the possible flows. We obtained the best results when using the preferential attachment graph. As can be observed, it starts from 0.6 and reach 0.9 in step 200, whereas both the RMAT and random graphs achieve a correlation value not greater than 0.7.

B. Convergence Time

In Section V-A1 we evaluated the results of DUCKWEED when varying the $\mathscr{D}_\epsilon$ parameter. We found that the results achieved using an increased value of $\mathscr{D}_\epsilon$ have an increased correlation with the exact values of current flow betweenness centrality. Here, we evaluate the impact of $\mathscr{D}_\epsilon$ on the time spent to *terminate* a flow. We measure this time in terms of STEP, i.e., the number of times DUCKWEED is executed on each node.

Figure 4 presents the number of STEP required by adopting different values of $\mathscr{D}_\epsilon$ for the analysis of the very same graph. The results presented are computed as an average of 1000 flow computations. As expected, an higher decimal precision corresponds to an higher amount of STEP. In particular, a

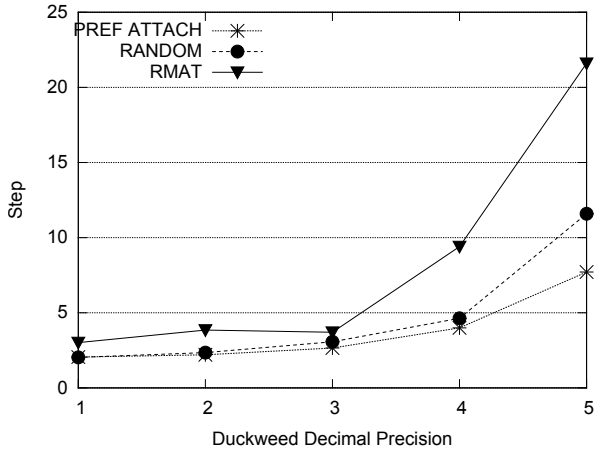


Fig. 4: Convergence Time

decimal precision greater than 3 leads to a sensible increase of STEP, that reach its maximum of 22 STEP with a decimal precision of 5 on the RMAT graph. Anyhow, it is worth to notice that with a decimal precision below 3 the amount of STEP is always less than 5 for any given datasets. This is quite interesting, in fact with a decimal precision of 3, DUCKWEED already achieves good correlation values with the exact values.

C. Centrality on Large Graphs

The evaluation of certain centrality measures in large graphs is an issue by itself. In fact the exact computation of some centrality measures can be very costly, from a computational viewpoint, when the size of the graph is large. A common strategy to evaluate the centrality of a given set of nodes, without having to compute exact results, is to remove the whole set from the graph and measure the amount of connected components (CCs) characterising the graph after this process [7], [36].

Clearly, an higher number of CCs corresponds to a better identification of nodes responsible to maintain the network connectivity. In the following we refer to this metrics as CC NUMBER.

Using this metrics we conducted two different sets of experiments. The first testbed is aimed at evaluating, in terms of CC NUMBER, the quality of the results produced by DUCKWEED with respect to the result obtained by the DEGREE centrality. Previous results for CFBTW approximation provided result for graphs in the order of 10^3 nodes in Brandes *et al.* [37] and 3×10^3 in Avrachenkov *et al.* [7]. We chose the DEGREE centrality as the baseline since it is the least expensive in terms of computational complexity among popular centrality measures, and therefore suitable to be computed on large graphs.

For our evaluation we considered three graphs taken from the SNAP [38] website: (i) the *RoadPA* graph represents the roads network of Pennsylvania; (ii) the *DBLP* graph provides a co-authorship network of paper indexed by the DBLP service; (iii) the *Google* graph represents web pages and hyperlinks

released in 2002 by Google.

From these graphs we extracted each largest connected component, and we use these as the input for our evaluation. The connected components were extracted using [39], and measure respectively $1,087,562$, $317,080$ and $855,802$ nodes.

The results we achieved show that DUCKWEED is able to provide good results, i.e., identifies nodes with a centrality index sensibly higher than the one provided by DEGREE centrality.

The second experiment evaluates the approximation provided by DUCKWEED during the simulation. More in details, Figure 6 shows the amount of CC NUMBER that would be introduced if the top 100, 50 and 25 rankings were removed by the graph. The timespan considered focuses on the initial 500 STEP of the simulation. As expected, the top 25 nodes require less STEP to be identified with respect to the top 100. In fact, let us consider, for instance, Figure 6a. It can be noticed how after 50 STEP the top 25 curve identifies 20 CC NUMBER, a value that increases only marginally in the remaining of the simulation. Conversely, the top 100 curve rapidly increases till STEP 200 and then remains stable right to the end. In conclusion, the results presented in Figure 6a show that DUCKWEED is able to achieve a good level of approximation in a reasonable number of STEP (around 200). The results reported in figures 6b and 6c, are slightly different but still confirm the ability of DUCKWEED in quickly finding nodes that if removed lead to the creation of a consistent amount of connected component.

D. Message Volume

In order to evaluate the cost of DUCKWEED, in terms of messages required for its execution, we measured the amount of messages as the total number of flows sent during a single STEP, and we called this metrics MSG VOLUME. To conduct our evaluation we considered three types of graph, each one generated according the following models: preferential attachment, random and RMAT. For each type we generated five graphs each having a different size: $\{10000, 20000, 40000, 80000, 160000\}$. We run the experiments by varying the number of concurrent flows, per step, in the following set: $\{10, 20, 40\}$.

As can be noticed in Figure 7a the value of MSG VOLUME for the RMAT graph increases linearly with the graph size (the results obtained using preferential attachment and random graphs are not included because are equivalent to the RMAT ones). This is easy to see in Figure 7a, in fact with 10 concurrent flows the MSG VOLUME value equals to 2.5×10^6 for a graph size of 80000 nodes and about 5×10^6 for a graph of 160000 nodes. Similarly, an increment on the number of concurrent flows leads to a linear increment of MSG VOLUME.

Figure 7b presents the results achieved using different types of graphs and fixing the number of concurrent flows to 10. We observed that all the graphs behave a similar way when increasing the network size. However, the preferential attachment graph requires the largest MSG VOLUME, whereas RMAT the smallest. This is essentially due to the total amount of edges in each graph: the greatest in preferential attachment and the lowest in RMAT.

Figure 7c shows the MSG VOLUME *per node*, with the number of concurrent flows fixed to 10. The results we achieved show that the network size affects only marginally the MSG VOLUME per node. In particular, with the preferential attachment and RMAT the MSG VOLUME remains constant, whereas it slightly increases with random graph. These results suggest that DUCKWEED scales in terms on MSG VOLUME with the size of the graph, which makes it suitable for computation on large graphs.

E. Flow Creation Strategy

This last set of experiments is aimed at evaluating the impact of the flow creation strategies (random, adaptive, partitioner, described in Section IV-C1) on the identification of central nodes. To this end we executed a simulation of 5000 flows with all the graph types, but fixing the graph size to 160000 nodes. The CC NUMBER achieved by DUCKWEED were sampled every 10 completed flows by removing the top-$\{25,50,100\}$ most central nodes. From the sample, we computed the minimum, maximum, average and standard deviation. An overview of the results for the Random graph are reported in Table I (the results with the other graph types are omitted as the results were almost the same).

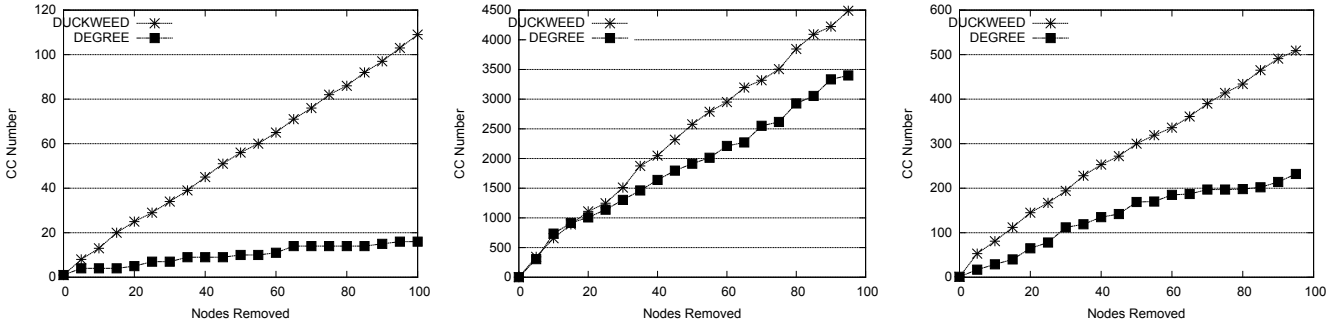
Even if Brandes *et al.* [37] observed that from a theoretical viewpoint a random selection of the flow performs better than more complex heuristics, from the results we achieved it can be observed that the adaptive strategy provides the best result on average. However, some aspects are worth to notice: (i) the random strategy considered by Brandes *et al.* exploits a "perfect" uniform random, rather, the random nodes in DUCKWEED are taken from the random peer sampling service, that only from a theoretical point of view converges to an uniform random; (ii) from a temporal analysis of the results, we observed that the differences among the strategies is more evident during the initial steps of the simulation when the number of completed flows is still low. Instead, at the end of the simulation all the strategies achieved similar results.

Overall, these considerations suggest that in a distributed context where an uniform random distribution is not easily available, a more elaborate strategies can have an impact on the computation of the CFBTW, especially if does not require to compute complex global measures of the graph. Instead, the considerations of Brandes et al. still hold in a controlled environment that provides an uniform random.

A last consideration can be made about the partitioner strategy, which appears to be, on average, the least performer. In this case the introduction of a gossip layer to compute the distributed partitioning of a graph appears to be not justified, according to the poor performances of the strategy.

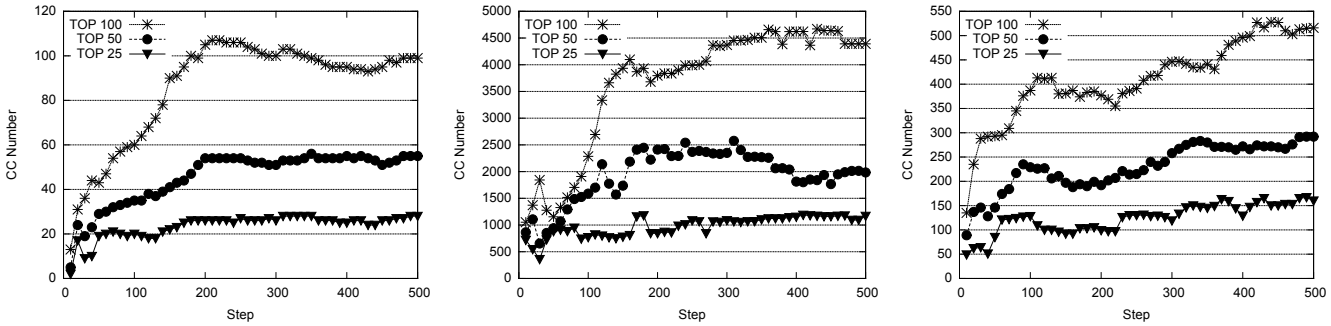
VI. Conclusion

In this paper we presented DUCKWEED, a distributed approach to calculate the current flow betweenness centrality. To the best of our knowledge, our approach is the first solution



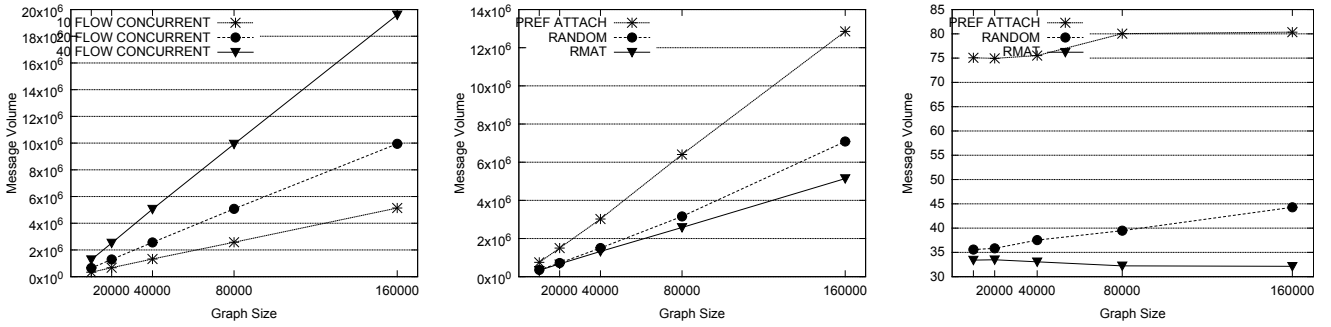
(a) Road PA (b) Google (c) Dblp

Fig. 5: CC Number (large number is better)



(a) Road PA (b) Google (c) Dblp

Fig. 6: CC Number: Graph Cut While Increasing Duckweed approximation



(a) Increasing graph size (b) Different Graph Model (c) Volume Per Node

Fig. 7: Message Volume Evaluation

able to compute a centrality index, based on global properties of the graph, in a fully distribute way.

Each node relies only on local information, i.e. on information stored by itself or by its neighbours, to adaptively estimate the value of its current flow betweenness. We empirically prove that DUCKWEED is able to provide a good approximation of the CFBTW and correctly identifies central nodes which guarantee network connectivity. DUCKWEED delivers good results also for large graphs over millions of nodes, while previous approaches considered only thousands of nodes, and is characterized by a good scalability.

As a future work, we plan to conduct further studies to analyse the behaviour of our approach for dynamic graphs. Furthermore, we believe our approach is able to return good results also in the context of the cluster based data intensive computation. In this case, we plan to investigate whether Kirchhoff's circuit simplification rules can be exploited to detect block of nodes which can be reduced to a single node, with the aim of optimizing the overall computation.

REFERENCES

[1] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, pp. 35–41, 1977.

[2] L. Freeman, S. Borgatti, and D. White, "Centrality in valued graphs: A measure of betweenness based on network flow," *Social Networks*, vol. 13, no. 2, 1991.

	TOP 25				TOP 50				TOP 100			
	MIN	MAX	AVG	SDEV	MIN	MAX	AVG	SDEV	MIN	MAX	AVG	SDEV
RANDOM	2	11	8.49	2.25	4	12	10.28	1.79	8	18	13.46	2.3
ADAPTIVE	3	14	**9.45**	2.74	4	17	**11.63**	2.51	11	21	**15.54**	2.68
PARTITIONER	3	11	7.48	1.37	5	14	9.24	1.79	7	19	12.28	3

TABLE I: Comparison of the flow creation strategies for the Random graph (in terms of CCs created)

[3] M. Newman, "A measure of betweeness centrality based on random walks," *Social Networks*, vol. 27, 2005.

[4] U. Brandes and D. Fleischer, "Centrality measures based on current flow," in *Lecture Notes in Computer Science*, vol. 3404, 2005, pp. 533–544.

[5] M. E. Newman, "A measure of betweenness centrality based on random walks," *Social networks*, vol. 27, no. 1, pp. 39–54, 2005.

[6] E. Bozzo and M. Franceschet, "Approximations of the generalized inverse of the graph laplacian matrix," *Internet Mathematics*, vol. 8, no. 4, pp. 456–481, 2012.

[7] K. Avrachenkov, N. Litvak, V. Medyanikov, and M. Sokol, "Alpha current flow betweenness centrality," in *Algorithms and Models for the Web Graph*. Springer, 2013, pp. 106–117.

[8] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, 2010, pp. 10–10.

[9] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[10] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[11] M. Riondato and E. M. Kornaropoulos, "Fast approximation of betweenness centrality through sampling," in *Proceedings of the 7th ACM Conference on Web Search and Data Mining, WSDM*, vol. 14, 2013.

[12] W. Hoeffding, "Probabilty inequalities for sums of bounded random variales," *Journal of american Statistical Association*, vol. 58, pp. 13–30, 1963.

[13] E. Bozzo and M. Franceschet, "Resistance distance, closeness, and betweenness," *Social Networks*, vol. 35, no. 3, pp. 460–469, 2013.

[14] K. A. Lehmann and M. Kaufmann, "Decentralized algorithms for evaluating centrality in complex networks," Department of Computer Science, Michigan State University, Wilhelm-Schickard-Institut, Tech. Rep. WSI-2003-10, October 2003.

[15] K. Wehmuth and A. Ziviani, "DACCER: distributed assessment of the closeness centrality ranking in complex networks," *Computer Networks*, vol. 57, no. 13, pp. 2536–2548, 2013.

[16] A. Kermarrec, E. L. Merrer, B. Sericola, and G. Trédan, "Second order centrality: Distributed assessment of nodes criticity in complex networks," *Computer Communications*, vol. 34, no. 5, pp. 619–628, 2011.

[17] P. G. Doyle and J. L. Snell, "Random walks and electric networks," 2006.

[18] R. Baraglia, P. Dazzi, M. Mordacchini, and L. Ricci, "A peer-to-peer recommender system for self-emerging user communities based on gossip overlays," *Journal of Computer and System Sciences*, vol. 79, no. 2, pp. 291–308, 2013.

[19] R. Baraglia, P. Dazzi, M. Mordacchini, L. Ricci, and L. Alessi, "Group: A gossip based building community protocol," in *Smart Spaces and Next Generation Wired/Wireless Networking*. Springer Berlin Heidelberg, 2011, pp. 496–507.

[20] P. Dazzi, P. Felber, L. Leonini, M. Mordacchini, R. Perego, M. Rajman, and É. Rivière, "Peer-to-peer clustering of web-browsing users," *Proc. LSDS-IR*, pp. 71–78, 2009.

[21] M. Mordacchini, P. Dazzi, G. Tolomei, R. Baraglia, F. Silvestri, and S. Orlando, "Challenges in designing an interest-based distributed aggregation of users in p2p systems," in *Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on*. IEEE, 2009, pp. 1–8.

[22] E. Carlini, M. Coppola, P. Dazzi, D. Laforenza, S. Martinelli, and L. Ricci, "Service and resource discovery supports over p2p overlays," in *Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on*. IEEE, 2009, pp. 1–8.

[23] R. Baraglia, P. Dazzi, B. Guidi, and L. Ricci, "Godel: Delaunay overlays in p2p networks via gossip," in *IEEE 12th International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2012, pp. 1–12.

[24] E. Carlini, L. Ricci, and M. Coppola, "Reducing server load in mmog via p2p gossip," in *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2012, p. 11.

[25] E. Carlini, P. Dazzi, A. Esposito, A. Lulli, and L. Ricci, "Balanced graph partitioning with apache spark," in *Euro-Par 2014: Parallel Processing Workshops*. Springer, 2014, pp. 129–140.

[26] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. Van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems (TOCS)*, vol. 25, no. 3, p. 8, 2007.

[27] S. Voulgaris, D. Gavidia, and M. Van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, 2005.

[28] L. Massoulié, E. Le Merrer, A.-M. Kermarrec, and A. Ganesh, "Peer counting and sampling in overlay networks: random walk methods," in *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*. ACM, 2006, pp. 123–132.

[29] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems (TOCS)*, vol. 23, no. 3, pp. 219–252, 2005.

[30] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, "Ja-be-ja: A distributed algorithm for balanced graph partitioning," in *Self-Adaptive and Self-Organizing Systems (SASO), 2013 IEEE 7th International Conference on*. IEEE, 2013, pp. 51–60.

[31] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.

[32] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *Peer-to-Peer Computing, 2009, Ninth International Conference on*. IEEE, 2009, pp. 99–100.

[33] J. Leskovec and R. Sosič, "Snap.py: SNAP for Python, a general purpose network analysis and graph mining tool in Python," http://snap.stanford.edu/snappy, Jun. 2014.

[34] D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-mat: A recursive model for graph mining." in *SDM*, vol. 4. SIAM, 2004, pp. 442–446.

[35] D. A. Schult and P. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, vol. 2008, 2008, pp. 11–16.

[36] U. Demšar, O. Špatenková, and K. Virrantaus, "Identifying critical locations in a spatial network with graph theory," *Transactions in GIS*, vol. 12, no. 1, pp. 61–82, 2008.

[37] U. Brandes and C. Pich, "Centrality estimation in large networks," *International Journal of Bifurcation and Chaos*, vol. 17, no. 07, pp. 2303–2318, 2007.

[38] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[39] A. Lulli, L. Ricci, E. Carlini, P. Dazzi, and C. Lucchese, "Cracker: Crumbling large graphs into connected components," in *20th IEEE Symposium on Computers and Communication (ISCC) (ISCC2015)*, Larnaca, Cyprus, Jul. 2015.