Exact Distributed Load Centrality Computation: Algorithms, Convergence, and Applications to Distance Vector Routing

Leonardo Maccari, *Member, IEEE*, Lorenzo Ghiro, Alessio Guerrieri, Alberto Montresor, and Renato Lo Cigno, *Senior Member, IEEE*

Abstract—Many optimization techniques for networking protocols take advantage of topological information to improve performance. Often, the topological information at the core of these techniques is a centrality metric such as the *Betweenness Centrality* (BC) index. BC is, in fact, a centrality metric with many well-known successful applications documented in the literature, from resource allocation to routing. To compute BC, however, each node must run a centralized algorithm and needs to have the global topological knowledge; such requirements limit the feasibility of optimization procedures based on BC. To overcome restrictions of this kind, we present a novel distributed algorithm that requires only local information to compute an alternative similar metric, called *Load Centrality* (LC). We present the new algorithm together with a proof of its convergence and the analysis of its time complexity. The proposed algorithm is general enough to be integrated with any distance vector (DV) routing protocol. In support of this claim, we provide an implementation on top of Babel, a real-world DV protocol. We use this implementation in an emulation framework to show how LC can be exploited to reduce Babel's convergence time upon node failure, without increasing control overhead. As a key step towards the adoption of centrality-based optimization for routing, we study how the algorithm can be incrementally introduced in a network running a DV routing protocol. We show that even when only a small fraction of nodes participate in the protocol, the algorithm accurately ranks nodes according to their centrality.

Index Terms—Multi-hop networks, mesh networks, ad-hoc networks, bellman-ford, load centrality, distributed algorithms, failure recovery

1 Introduction

Topology awareness enables significant optimizations for distributed systems. For example, the influence of nodes placed in strategic locations can be measured, and these measurements can guide the optimal allocation of resources in the system. In this regard, centrality metrics are well-known instruments provided by graph theory to quantify the influence of the nodes in a network. Among the multitude of centrality indexes proposed in the literature, a widely used one is the *Betweenness Centrality* (BC) index: roughly speaking, BC measures the fraction of global shortest paths passing through a vertex [2]. BC has found numerous applications in distributed systems, where it is used for optimal service placement [3], to improve routing [4], [5], [6], for topology control [7], [8], for security [9], [10], and for several other uses [11], [12].

We focus on a particular application of centrality in computer networks, namely on the tuning of control messages for faster routing convergence [4]. The idea is to increase the frequency of control messages for the most central nodes while reducing it for marginal ones: this leads to faster re-routing and reduced traffic loss upon an average failure in the network, without increasing the overall control overhead. This tuning technique based on BC has been initially applied only to link-state (LS) routing protocols, as these protocols enable each router to collect information about the full network topology, which is a requirement for the computation of BC [6], [13]. In distance-vector (DV) routing protocols, instead, routers are not required to know the entire topology. This enables networks managed by a DV protocol to scale up to millions of nodes, but does not provide enough information to routers to compute BC. As a consequence, the lack of an exact distributed algorithm is still preventing the centrality-based optimization of DV protocols.

In this paper we target another centrality metric, called *Load Centrality* (LC) [2], [14], that is very similar to BC, but can be computed in a distributed way. Instead of relying on the number of shortest paths, LC estimates the amount of traffic that insists on a node, assuming that the traffic is fairly distributed among all shortest paths of the network. LC converges to BC in many real-world cases, and we claim that in the rest of the cases LC is a better metric than BC to measure the communication traffic in today's computer networks and should therefore be preferred for distributed applications. We better detail this claim in Section 2.3.

The core of our contribution is the proposal of an efficient distributed algorithm able to exactly compute LC, paving

- *L. Maccari is with the University of Venice, 30123 Venice, Italy. E-mail: leonardo.maccari@unive.it.*
- *L. Ghiro and A. Montresor are with the University of Trento 38122 Trento, Italy. E-mail: {lorenzo.ghiro, alberto.montresor}@unitn.it.*
- *A. Guerrieri is with SpazioDati srl, 56021 Cascina, Italy. E-mail: alessio.guerrieri@unitn.it.*
- *R. Lo Cigno is with the University of Brescia, 25121 Brescia, Italy. E-mail: locigno@disi.unitn.it.*

Manuscript received 6 May 2019; revised 6 Feb. 2020; accepted 11 Feb. 2020. Date of publication 14 Feb. 2020; date of current version 6 Mar. 2020. (Corresponding author: Leonardo Maccari.) Recommended for acceptance by M. Guo. Digital Object Identifier no. 10.1109/TPDS.2020.2973960

1045-9219 © 2020 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

the way for centrality based optimizations in distributed systems. It is defined on top of the Bellman-Ford algorithm, thus can be integrated into any DV routing protocol introducing minimal modifications. The convergence time of our algorithm scales linearly with the network diameter D: after a number of steps bound by $3D$, every vertex knows its own centrality and the centrality of all the other vertices.

One critical step to go from a sound theoretical approach to a real-world application is the study of the applicability of the algorithm to real protocols. In this paper, we address this problem by studying how the distributed algorithm can be incrementally deployed in an existing network updating a running protocol. We also show that the estimation of LC is possible when only a subset of nodes in the network supports the updated protocol. This is essential in real protocol deployments: as the network size grows, it becomes less and less feasible to simultaneously upgrade all of the nodes in that network. We provide a mathematical characterization of the approximation errors as a function of the penetration rate for both the centrality estimates and the centrality rankings, and we show that, even with low penetration rate, our algorithm ranks nodes accurately. Finally, we provide simulations that confirm our theoretical findings.

To further show the practical applicability of our contribution, we extend `babeld`, the open-source implementation of the widely used DV routing protocol for mesh networks specified in RFC 6126 [15]. We show that, exploiting the notion of LC, the protocol convergence time can be improved up to 13 percent without increasing signaling overhead.

2 Background and Contributions

Centrality measures have been used to enhance traffic monitoring [5], [9], intrusion detection [16], resource allocation [11] and topology control [7]. Among all, *Betweenness Centrality* is a well-known centrality index, which can be computed with Brandes' centralized algorithm [17]. Brandes' algorithm executes an instance of Dijkstra's algorithm rooted on each vertex of the graph, while updating in parallel the BC indexes. In a network with n nodes and m weighted edges, the computational complexity of this approach is $O(nm + n^2 \log n)$. Brandes' algorithm can be adapted to compute other centrality indexes based on minimum-weight paths [2]. For example, Dolev *et al.* proposed a generalization of BC to deal with different routing policies [5].

2.1 Centralized Algorithms for Centrality

There are two main problems that hinder the use of centralized algorithms in a network of routers. First, only LS protocols provide information on the whole network topology to nodes; a mandatory requirement to perform the centralized BC calculation. DV protocols are completely excluded. Second, when the network size becomes large, centrality metrics may require excessive computational resources with LS protocols as well. Despite the introduction of several heuristics [18], the online computation of the indexes on low-power hardware requires several seconds and is generally not possible in real-time on large networks [10].

One natural approach to speed up the computation is random sampling [19], [20], [21], [22], [23], [24]. Independently from each other, Jacob *et al.* [22] and Brandes and Pich [20] proposed approximated algorithms that only consider contributions from a subset of vertices sampled uniformly at random. Later proposals can compute BC with adjustable accuracy and confidence [25], [26]. More recently, dynamism has been taken into consideration, with several algorithms able to update BC on evolving graphs [27], [28], [29], [30]. These randomized algorithms are fast, but still centralized.

2.2 Distributed Algorithms for Centrality

Distributed algorithms for the computation of centrality with sufficiently good scalability properties have been proposed, based on a dynamic system approach, but only for specific topologies (DAGs and trees) [31], [32], [33] or with approximated results [34], or are designed under strong assumptions on the synchronism of communications [35], [36], hypotheses that are not met in real-world distributed systems. For example, [31] computes "*the betweenness centrality of an oriented tree [...] taking advantage of the fact that a tree does not contain any loop, and therefore every pair of nodes has at most one shortest path*", while [32], [33] are restricted to "*undirected and unweighted tree graphs*". Also most of the alternative distributed algorithms for BC, such as [35], [36], usually compromise generality as they are derived under the `CONGEST` model [37], which sets the following strong assumptions:

- The network is an undirected connected graph $G(\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ nodes;
- Both nodes and channels are reliable (failure-free);
- A global clock triggers consecutive rounds of message passing and processing, the processing is instantaneous as nodes have infinite power.

To overcome the limitations in terms of generality and exactness of the algorithms for BC, we direct our attention to a different metric, the *Load Centrality* (LC). LC is similar to BC (it is often confused with it [2]) and, above all, is a better metric than BC for modern distributed systems that employ load balancing techniques.

2.3 Load VS Betweenness Centrality

Definition 1 (Load Centrality (LC)). *Consider a graph $G(\mathcal{V}, \mathcal{E})$ and an algorithm to identify the (potentially multiple) minimum weight path(s) between any pair of vertices (s, d). Let $\theta_{s,d}$ be a quantity of a commodity that is sent from vertex s to vertex d. We assume the commodity is always passed to the next hop following the minimum weight paths. In case of multiple next hops, the commodity is divided equally among them. We call $\theta_{s,d}(v)$ the amount of commodity forwarded by vertex v. The* load centrality *of v is then given by*

$$LC(v) = \sum_{s,d \in \mathcal{V}} \theta_{s,d}(v). \tag{1}$$

Normally it is assumed that s, d, v are all distinct and that $\theta_{s,d} = 1$. The latter makes LC a property fully defined by the graph structure and by the algorithm used to discover minimum weight paths. In that case, if the graph is undirected there are $\frac{N(N-1)}{2}$ pairs (s, d) and LC can be normalized as

$$\overline{LC}(v) = \frac{2}{N(N-1)} \sum_{s,d \in \mathcal{V}} \theta_{s,d}(v). \tag{2}$$

BC instead is defined as follows [38]

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

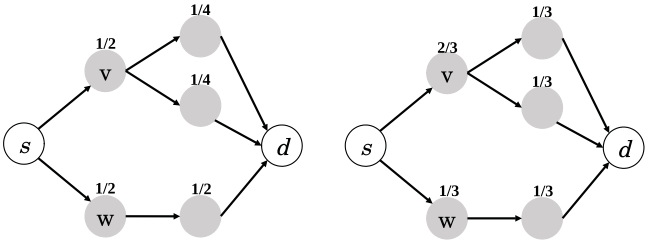


Fig. 1. Difference between centrality computation in the case of load (left) and betweenness (right) in the same network for the same (s, d) couple.

Definition 2 (Betweenness Centrality (BC)). *Let σ_{sd} be the number of minimum weight paths between vertex s and vertex d, and let $\sigma_{sd}(v)$ be the number of those minimum weight paths passing through vertex v (again, s, d, v are distinct). The normalized* betweenness centrality *of v is defined as*

$$\overline{BC}(v) = \frac{2}{N(N-1)} \sum_{s,d \in \mathcal{V}} \frac{\sigma_{sd}(v)}{\sigma_{sd}}. \quad (3)$$

LC and BC are very similar, but they do not coincide as already noted by Brandes [2]. Consider Fig. 1, reporting a sample network annotated with the values of LC and BC on each node, assuming every edge has the same weight. If the commodity moving from s to d is split equally between two next hops that lie on two paths with equivalent total weight, intuitively node v and w will both carry half of it. This is what is measured by LC. BC instead reflects the fact the v, on its right side, has more minimum weight paths towards d compared to w. Since BC counts the fraction of minimum weight paths passing through a node, v turns out to be more central than w.

The reasons why we consider LC instead of BC are primarily three. First, the two metrics diverge when there are multiple paths between couples of nodes, but in all situations in which there is only one minimum weight path between s and d, the two metrics coincide. Most IP-based routing protocols use only one path at a time and thus, LC equals BC in all of them. Second, for protocols that support multipath routing (such as the Stream Control Transport Protocol [39] or Multipath TCP [40]) it is not important how many paths exist between s and d, it is important on how many paths the traffic is distributed upon. This is what LC expresses and in this sense, the semantic of LC captures this behavior better than BC. Third, LC better describes the behavior of DV protocols, such as the Border Gateway Protocol (BGP [41]) that runs the Internet. With BGP every router takes a local decision on which path to use based on its policies. Even the knowledge of the full network graph would not be sufficient to compute BC, as the local policies are not known. With our distributed LC computation algorithm instead, centrality is computed as a consequence of each router's policy and propagated on all shortest paths, which makes it suitable to be used also with BGP (and other DV protocols).

2.4 Contributions

We propose a distributed and exact algorithm to compute LC that is general, as it works on top of any network with arbitrary topology. The only requirement (as explained in Section 3) is the existence of an underlying routing protocol that keeps the routing table up-to-date in each node. The exactness of our algorithm is verified later in Section 7.1, where we show (see Fig. 6) that LC values computed with our distributed algorithm match with the values computed off-line with standard and well known libraries for graph manipulation. Furthermore, our algorithm enables centrality-based optimizations in distributed systems. As a concrete example, in Section 7 we show our implementation on top of Babel, a real-world DV protocol, and show that the LC-enabled optimization effectively reduce losses upon failures and makes the re-routing process faster. Summing up, our contributions and advancements compared to the state-of-art are the following ones:

1) Our algorithm for LC is distributed (relies on local information only), exact and more general than current alternatives;
2) It enables centrality-based optimizations in distributed systems;
3) It can be incrementally deployed on an existing network without requiring all nodes to be updated at the same time.

The last contribution is key to increase the potential for adoption of centrality-based optimizations, as it is usually not possible to fully stop and reboot a running network. Especially when networks are large and managed by more than one entity, a full redeployment requires a great coordination effort, with network administrators that need to agree on a flag-day to perform the simultaneous update of all network nodes. Distributed centrality-based optimization instead can be incrementally deployed and protocols can benefit from it even before all nodes in the network support it.

TABLE 1
Variables Used by Each Vertex v in Algorithm 1

Symbol	Description
$\mathcal{V}$	The set of all vertices
$neighbors$	The set of neighbors of v
NH	For each destination $d \neq v$, the vector $NH[d]$ is the set of vertices used by v as next hops to reach d
PH	For each destination $d \neq v$, vector $PH[d]$ is the set of *previous hops*, i.e., vertices that list v as one of the next hops to reach d
$loadOut$	For each destination $d \neq v$, $loadOut[d]$ is the overall commodity passing through v to reach d
$contrib$	For each destination $d \neq v$, $contrib[d]$ is the contribution that v will send to each of its next hops to reach d, equal to $loadOut[d]/\lvert NH[d]\rvert$
$loadIn^u$	For each neighbor u and for each destination $d \neq v$, $loadIn^u[d]$ is the commodity's contribution that vertex u sends to v toward d (as reported by u to v)
$load$	The approximation of load centrality known so far

3 Distributed LC Computation

The distributed algorithm for LC computation, as executed by vertex v, is shown in Algorithm 1, with Table 1 listing all information maintained by v. Recall that to compute centrality metrics, a routing table with the next hop is in general not sufficient and a full topological knowledge is required.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1 is based on the commodity diffusion process described in Definition 1. Each vertex generates a unitary amount of commodity for all possible destinations; such commodity is split and aggregated along the route to destinations.

The routing protocol keeps an up-to-date list of *next hops* in vector NH, where $NH[d]$ contains the next hops to reach destination d. The algorithm computes the complementary vector PH, where $PH[d]$ contains the *previous hops* from which the commodity going toward d is coming. This is obtained by periodically sending a message $\langle v, NH, contrib\rangle$ to all neighbors of v; when these messages are received by each next hop, PH is updated. The previous hops stored in $PH[d]$ are used to aggregate all the incoming commodity toward d before splitting it among all next hops.

Algorithm 1. General Distributed Protocol (Executed by v)

```
 1: Init:
 2:   load = 0;
 3:   loadOut[v] = contrib[v] = 0;
 4:   foreach d ∈ V − {v} do
 5:     foreach u ∈ neighbors do
 6:       loadIn^u[d] = 0;
 7:     PH[d] = [ ];
 8: Repeat every δ seconds:
 9:   foreach d ∈ V − {v} do
10:     loadOut[d] = 1 + Σ_{u∈PH[d]} loadIn^u[d];
11:     contrib[d] = loadOut[d]/|NH[d]|;
12:     load = load + loadOut[d];
13:   send ⟨v, NH, contrib⟩ to neighbors;
14: on receive ⟨u, NH^u, contrib^u⟩ from u do
15:   foreach d ∈ V − {v} do
16:     if v ∈ NH^u[d] then
17:       PH[d].add(u);
18:       loadIn^u[d] = contrib^u[d];
19:     else
20:       PH[d].delete(u);
```

The rest of Algorithm 1 is designed to maintain information about incoming and outgoing commodity. In particular, dictionary $loadOut$ stores the overall commodity passing through v to reach every possible destination, while $contrib$ stores the commodity's contributions that v sends to each of its next hops. Note that having both $loadOut$ and $contrib$ is redundant; $loadOut$ is introduced only to clarify the algorithm and simplify the proof that $load$ converges to LC.

- During initialization (lines 1-7), the commodity coming from every neighbor is set to 0, while waiting for more up-to-date information to come. PH entries are initialized to an empty vector as well.
- Every δ seconds (lines 8-13), each vertex v re-computes (for every destination d) its contribution to $load$ for its next hops and sends this contribution to all its neighbors with the message $\langle v, NH, contrib\rangle$. The contribution is given by 1 (its unit contribution to the load addressed to d) plus all contributions received so far, divided among all vertices that are next hops for destination d (lines 10-12).
- When a message from vertex u is received (lines 14-20), vertex v first updates the previous hop set PH, by either adding (line 17) or deleting (line 20) u. Then, it copies the contributions toward every d computed by u and received in the message $\langle u, NH^u, contrib^u\rangle$ into $loadIn^u$ (line 18).

4 Convergence Study

We show that at steady state, under sufficiently stable conditions, $load$ in Algorithm 1 converges to the correct LC at each vertex.

4.1 Theoretical Proof

Theorem 1. *Let $G = \{G_d = (\mathcal{V}, E_d) : d \in \mathcal{V}\}$ be the collection of all routing graphs induced by all nodes running an underlying routing protocol*

$$E_d = \{(i, j) : i \in NH_j\}.$$

If G remains stable for a long enough period of time then, for each node v, the 'load' variable maintained by v will eventually converge to the correct LC of v.

Proof. Given a node v, we prove that for each destination d, the commodity that v forwards toward d is eventually computed in the correct way. Since the overall commodity forwarded by v towards any possible destination is periodically aggregated into variable $load$, this proves the theorem.

For each destination d, the routing protocol generates a routing graph: a loop-free directed acyclic graph (DAG) made of all the (potentially multiple) minimum weight paths ending in d. Let $S = \{s = u_0, u_1, u_2, \ldots, u_{|\mathcal{V}|} = d\}$ be a sequence representing a topological sort of the DAG G_d. We prove that each node in the sequence correctly computes the load that is passing through v, by induction on the sequence of nodes.

Given that G_d is a DAG, the set $PH[d]$ of the first node u_0 is empty. Thus, $loadOut[d]$ is set to 1, which is the correct value for the load passing through this node. This load is then divided equally among all nodes in $NH[d]$.

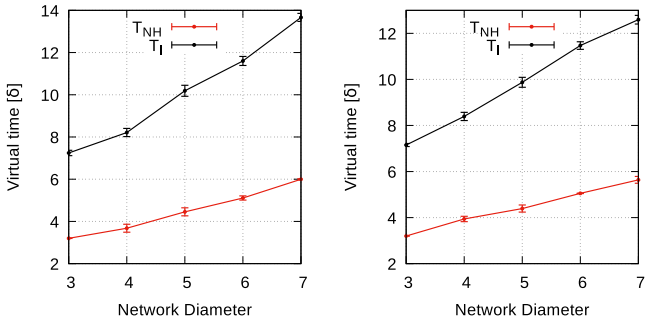
Now, consider node u_k and assume all preceding nodes in the sequence have already computed the correct value for their variable $loadOut[d]$. Each node $u \in PH[d]$ is included in $\{u_0 \ldots u_{k-1}\}$, thanks to the topological sort. Thus, all of them will eventually send a message to v, updating the corresponding entries in variable $loadIn$.

As soon as node v receives all the required information from all nodes in $PH[d]$, variable $loadOut[d]$ contains the correct value. A special case is given by node d, where $loadOut[d] = 0$. □

The theorem assumes that minimum weight paths are stable long enough to allow the centrality computation to converge. In case of dynamism, results can be temporarily different from the correct ones, until the routing paths stabilize again. At that point, given all the needed information is periodically broadcast to all vertices, Algorithm 1 converges again to the correct LC values.

We can estimate the convergence time of Algorithm 1 in the worst case scenario. We assume all clocks are synchronized and time required to propagate data along an edge is very small compared to the period δ, but not null. Therefore, vertex v receives updates from neighbors always after it sent

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.



(a) Barabási-Albert networks. (b) Erdős networks.

Fig. 2. T_{NH} and T_l versus network diameter, with 99 percent confidence interval, for Barabási-Albert and Erdős networks.

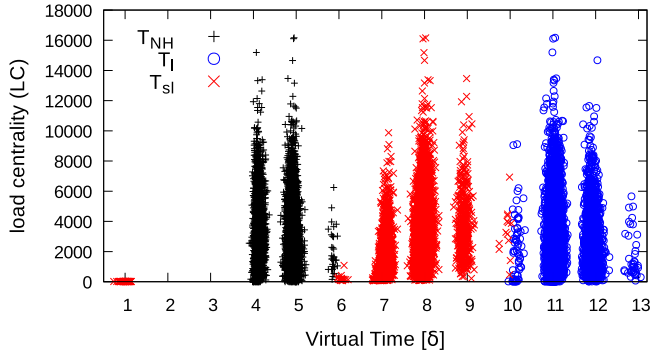


Fig. 3. T_{NH}, T_{sl}, and T_l for all the nodes, 40 simulations, Erdős graphs with diameter 7.

its own updates, and time needed to propagate information on k hops is always $k\delta$ seconds. We also assume that our algorithm starts after the routing protocol convergence. Under this assumption, the following corollary holds:

Corollary 1. *Given a graph G with diameter D, the convergence time Δ_t of our algorithm is in the worst case proportional to $D-1$.*

Proof. A vertex v converges when its *load* sums all contributions from all minimum weight paths crossing v. Consider the load on v generated from s and directed to d for which v is in at least one of the minimum weight paths. If $v=s$ or $v=d$, convergence is immediate, as no contribution will be received. Let S be the graph ordering relative to G_d, and $v=u_k$. If $k=1$ then v converges after receiving the contribution from s, that is, after δ seconds. Otherwise v converges when the load is propagated from s to v, which requires k intervals. In the worst case $v=u_{D-1}$ and the *load* of v converges in $\delta(D-1)$ seconds. □

Generally, at node v the own centrality value is useful only if compared to the other nodes' centrality. This is why, after the complete convergence of centrality in the network, another message must be sent (and forwarded) by each vertex carrying its own value of centrality, implementing a dissemination process. In conclusion, again in the worst case scenario and after routing table convergence, the time required to perform centrality computation and dissemination will be proportional to $2\times D$.

4.2 Time Bounds in Simulation Analysis

We developed a Python simulator that takes the network as input and implements both Algorithm 1 and the underlying dissemination of LC indexes. The simulator uses a virtual clock and triggers a send event once every δ time units, so we can refer to the convergence time simply as multiples of δ. A small random jitter is added to events scheduled by nodes to avoid perfect synchronization.

A simulation ends when all nodes converge to steady state, i.e., when all nodes learned i) all their next-hops, ii) their own load index, and iii) the load indexes of all other nodes. We separately measure:

- The time needed for full convergence of *NH* (T_{NH});
- The time at which each node converges to its own value of centrality, called self-convergence time (T_{sl});
- The time T_l to learn the *load* of all other nodes in the network.

Full convergence is determined by the slowest node. Since we demonstrated that convergence time should grow, in the worst case, linearly with the network diameter D, we generated graphs with growing $D\in\{3,4,5,6,7\}$. For each D, we tested two different graph models: the classical Erdős and the Barabási-Albert model. For both we averaged results over 40 different graphs with 1,000 nodes each.

Fig. 2a and 2b report the average (with 99 percent confidence intervals) of T_{NH} and T_l simulated over Barabási-Albert and Erdős graphs respectively. Both figures show how the growth of convergence time is approximately linear with the diameter, confirming the theoretical result of Section 4.1.

As expected, it takes approx D time units for *RT* to converge, as DV messages need to travel across the whole network (and thus along a whole diameter). Since we add some jitter, generation of distance vectors is not synchronized, thus it may be that node i receives an update from a neighbor j before generating its own. In this case, in the same time unit, the information is sent from j to i and propagated from i to its neighbors, so it can take less than D time units for information to travel on the longest path. This explains why in the figures T_{NH} is generally smaller than D.

After convergence of *NH*, our theoretical analysis predicts a time of $2\times(D-1)$ to achieve full load convergence, while simulations show that it requires much less than that. This improvement is given by the parallelization of the two processes and can be seen in Fig. 3, which reports all values of T_{NH}, T_{sl} and T_l.

Fig. 3 reports all the results for all the 40 Erdős graphs with diameter 7 (networks with different parameters are not shown as they behave similarly) and shows that at time 6, when the last *RT* converges, some nodes already reach self convergence. Similarly at time 10, when all nodes reached self convergence, some nodes already reached full convergence. This is because the three processes are concurrent and thus, on typical networks, full convergence time is smaller than in the worst case scenario. Finally, note the small group of nodes which reach self convergence in only one time unit. This is a minimal fraction of leaf nodes produced by the Erdős generator whose LC is initially set to zero and never changes.

5 Incrementally Deployable Protocol

While so far we discussed the theoretical properties of the proposed algorithm, in this section and in the following one we focus on how our proposal can be implemented extending a real-world DV routing protocol. To this end we describe its

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Notation for Legacy and Upgraded Nodes

Field	Notation	Description
Metric	m	distance of v from d according to a given metric
Buffer	BUF	dictionary mapping destinations d to the list of subTLVs contained in recent advertisements for d
Routing Table	RT	dictionary mapping destinations d to distances expressed in a given metric m
SubTLV	stv	tuple of information, appended to a parent TLV such as a Route-Advertisement
Forwarded subTLVs	fwd	list of subTLVs
Source of load	$source$	id of the upgraded node generating a contribution
Remote NH	rNH	next-hop selected by a remote upgraded node to forward its subTLVs
Splitted Contributions	*splitC*	load contributions that an upgraded-node v sends toward destination d. If v has more *NHs* for d, contributions are evenly split before begin sent

incremental deployment on top of a DV protocol in which two kinds of nodes will coexist: *legacy nodes* running the legacy version of the protocol and *upgraded nodes* running the updated version.

RFC 6709 of IETF [42] provides general considerations about extension mechanism for networking protocols. A fundamental recommendation to design an extendable and incrementally deployable protocol is to employ type-length-value (TLV) packets to define protocol messages. Using TLVs it is easy to enrich a protocol and build a new and backward-compatible version, simply defining new TLVs. This means that upgraded nodes will not violate the syntax and semantics of all legacy protocol messages, but instead they are able to include additional information in them. Legacy nodes are typically designed to store and silently forward the information they are not able to interpret.

Algorithm 2. Logic of a Legacy Node v

```
1: Init:
2:    RT[v].m = 0; RT[v].NH = [ ]; BUF[v] = [ ]
3: repeat every δ seconds:
4:    Cleaning(now)    ▷ Clean expired subTLVs from BUF
5:    foreach u, d ∈ neighbors × RT() do
6:       if u ∈ RT[d].NH then
7:          fwd = BUF[d]    ▷ forward subTLV only on SPs
8:          send ⟨d, RT[d].m, fwd⟩ to u
9: on receive ⟨d, m, fwd⟩ from u do
      // Bellman-Ford
10:   if d ∉ RT() OR m + C[u] < RT[d].m then
11:      RT[d].NH = [u]
12:      RT[d].m = m + C[u]
13:   else if m + C[u] == RT[d].m then
14:      RT[d].NH.append(u)
15:   Cleaning(u, d)    ▷ Clear old subTLVs from u to d
16:   foreach stv ∈ fwd do
17:      stv.holdTime = now + ε
18:      BUF[d].append(⟨u, stv⟩)    ▷ Buffer subTLVs
   // Dictionary C[·] contains the cost of the links to neighbors
```

Many popular protocols designed for extensions implement this scheme. Three noteworthy examples are OLSR, Babel and BGP that, with its *Optional Transitive Attributes*, constitutes a prominent example of the so called *Silent Propagation Scheme*. These three protocols represent all possible routing categories, respectively: LS, DV and path-vector (PV).

In our context, legacy nodes perform only a basic DV routing protocol based on Bellman-Ford: therefore they will silently propagate the additional messages that upgraded nodes generate and process to perform the algorithm for LC. To do an incremental deployment Algorithm 1 is not suitable, and we need supplementary notation as described in Table 2 to define the behavior of legacy and upgraded nodes. The logic of a legacy node is represented in Algorithm 2, while an upgraded node extends this logic as shown in Algorithm 3.

Algorithm 3. Logic of an Upgraded Node v

```
1: Init:
2:    RT[v].m = 0; RT[v].NH = [ ]; RT[v].loadIn = { }
3: repeat every δ seconds:
4:    Cleaning(now)    ▷ Remove expired contributions
5:    foreach u, d ∈ neighbors × RT() do
6:       if u ∈ RT[d].NH then
7:          loadOut = 1    ▷ Generate/send contrib on SPs
8:          foreach u ∈ RT[d].loadIn() do
               // Aggregate received contributions
9:             loadOut += RT[d].loadIn[u]
10:         splitC = loadOut/|RT[d].NH|
            // subTLV with source v sent via u
11:         stv = (v, u, splitC)
12:         send ⟨d, RT[d].m, stv⟩ to u
13: on receive ⟨d, m, fwd⟩ from u do
      // Bellman-Ford as in Algorithm 2, plus
      // Process list of subTLVs attached to d
14:   foreach stv ∈ fwd do
15:      source, rNH, C = stv
         // Index received contributions by source and rNH
16:      RT[d].loadIn[source, rNH] = splitC
17:      RT[d].loadIn[source, rNH].holdTime = now + ε
   /* The load centrality of v is given summing up all contributions
in RT().loadIn. */
```

As for all DV routing protocols (such as the classical RIP, or Cisco EIGRP, or the Babel protocol adopted later in this work), the Bellman-Ford algorithm is used to maintain a Routing Table (RT) mapping each destination to a next hop and a distance. An RT is a dictionary mapping destinations d to distances expressed in some given metric m. We use the square brackets operator to retrieve the information regarding a particular destination d recorded in RT, i.e., $RT[d] = (m, NH)$. We use the dotted notation to access single fields of the tuple, as in $RT[d].NH$. For a dictionary, the '()' operator returns the list of the keys of the dictionary, i.e., $RT() = [d_0, d_1 \dots d_N]$. The '[]' and '{}' symbols refer to the empty list and dictionary, respectively. Note that NH is an array of next hops, therefore we support routing protocols that implement multipath routing.

A legacy node owns a buffer (BUF) to store the list of subTLVs that neighbors may attach to route-advertisements. On line 9 of Algorithm 2, a legacy node starts the routine to process a generic advertisement containing a destination d with cost m together with fwd, i.e., a list of subTLVs that the

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

sending neighbor u may have silently forwarded. On lines 10-14, the node performs the classic update of its RT selecting the NH that offers the shortest-path to reach the advertised destination. Multipath is supported by maintaining more next-hops if they offer paths with equal cost (line 14). While processing a received advertisement, a legacy node is not able to parse attached subTLVs, therefore it can only put them all in its buffer as shown in lines 16-18. In particular, subTLVs are indexed by d and marked with the id of the sending neighbor and have a limited time validity. Before storing subTLVs in BUF buffer, the legacy node discards previous information received by the same neighbor for the same destination (line 15).

The silent propagation of subTLVs is implemented when sending route advertisements (line 8), sent periodically when the node sends DVs to all its neighbors.[1] Beyond the fundamental advertisement of the destination d with the best known cost $RT[d].m$, legacy nodes also forward all the subTLVs that were contained in advertisements announcing d. The subTLVs to propagate are retrieved from the buffer (lines 5-7). Due to the control performed on line 6, subTLVs are forwarded only to valid neighbors: assuming routing convergence this means that subTLVs flow through shortest paths only.

Compared to a legacy node, an upgraded one is also able to parse subTLVs. SubTLVs define what we call *centrality contributions*, which are the minimal amount of information required to run the distributed protocol for the computation of LC. The logic of an upgraded node is described by Algorithm 3. Fundamentally, an upgraded node is able to generate and send contributions (lines 5-12), and also to aggregate and store the contributions it may receive (lines 14-17): summing up all the received contributions an upgraded nodes determines its own LC index.

When an upgraded one receives a route-advertisement (line 13), it does not ignore the attached subTLVs, but it rather parses them as shown on line 15. Line 15 defines the pieces of information that compose a centrality contribution, which are:

- $source$: the id of the remote node (in fact, it may not be a direct neighbor) that generated this contribution;
- rNH : the NH towards which the remote node sent the contribution;
- $splitC$: the amount of forwarded load, with the same meaning of $\theta_{s,d}(v)$ of Definition 1.

The received contributions are stored in RT indexed by their $source$ but also by their rNH (line 16): rNH is required to distinguish contributions split remotely by the same $source$ that need to be summed up in aggregation points reached after flowing through different paths. Stored contributions have a limited time validity (line 17). If a node stops being part of a given path, because of a new routing decision by a remote node, after some time this node will properly forget the contribution received previously on that path. Expired contributions are discarded invoking periodically a cleaning routine (line 4).

1. It is usually required to send different DVs to different neighbors, think for instance of the classic split-horizon technique widely used to mitigate the well-known *count-to-infinity* problem.

An upgraded node is able to create subTLVs to send centrality contributions. Similarly to the advertisement sent by legacy nodes, also upgraded nodes customize their announcements for their various neighbors. They offer their unitary load contribution (line 7), and the aggregation of all other contributions directed to the same d (lines 8-9), to those neighbors that lie on the shortest paths towards d. Compared to Algorithm 1, the aggregated load directed to d is equally split among all possible NH before being sent (line 10); finally a subTLV defining a centrality contribution is generated (line 11).

6 Performance of the Distributed LC Computation With Partial Information

Consider a network in which only a subset $\mathcal{H} \subseteq \mathcal{V}$ of *upgraded* nodes participate to the protocol, while the *legacy* nodes in $\mathcal{W} = \mathcal{V} \setminus \mathcal{H}$ do not participate. A node $h \in \mathcal{H}$ generates one unit of commodity towards every other node $v \in \mathcal{V}$, while a node $w \in \mathcal{W}$ does not generate any commodity. Every node h also aggregate and split commodity contributions when necessary and, above all, h estimates its own centrality and those of all other nodes in $\mathcal{H}$. Legacy nodes store the contribution field of the update packets they receive in their routing table as opaque metadata, and pass it to the next hop when they generate their own update packets. In this scenario, the upgraded nodes can only compute an underestimate of their LC indexes, and it is interesting to analyze how quickly this estimation reaches the true value increasing the number of updated nodes. We start by proving the following corollary:

Corollary 2. *Given a graph G and a set $\mathcal{H} \subseteq \mathcal{V}$ of nodes that support the protocol, if G remains stable for a long enough period of time then, for each node $h \in \mathcal{H}$, the 'load' variable maintained by h will eventually converge to*

$$LC'(h) = \sum_{s \in \mathcal{H}} \sum_{d \in \mathcal{V}} \theta_{s,d}(h). \tag{4}$$

Proof. The proof is straightforward from Theorem 1. When considering a topological sort and a node u_k, then from all nodes in $\{u_0 \dots u_{k-1}\}$ only those that belong to $\mathcal{H}$ generate the load contribution after collecting the messages. The total number of contributions that u_k receives for destination d is thus given by $|\{u_0 \dots u_{k-1} | u_i \in \mathcal{H}\}|$ and $loadOut[d] = \sum_{s \in \mathcal{H}} \theta_{s,d}(u_k)$. The same happens for every other destination, which means that with partial deployment, Algorithm 3 computes a partial version of the load centrality corresponding to Eq. (4). □

6.1 Error Estimation With Partial Coverage

Corollary 2 is simple but powerful; it shows how, with reasonable assumptions on the legacy nodes, we can exploit LC incrementally and we can estimate the load centrality ranking even with partial information. In the rest of the section, we go one step forward and give a theoretical analysis of the average error introduced in the estimation. An experimental study of how this error impacts the nodes' ranking in terms of centrality is presented in Section 6.2.

6.1.1 Error Function Definition

Given an arbitrary $\mathcal{H}$, with $H = |\mathcal{H}|$, we state the following:

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

Definition 3 (Average Normalized Load Centrality).

$$\overset{\triangle}{LC} = \frac{1}{N}\sum_{v\in\mathcal{V}} \overline{LC}(v) = \frac{2\sum_{v\in\mathcal{V}} LC(v)}{N^2(N-1)}. \quad (5)$$

Definition 4 (Average Normalized Partial Centrality).

$$\overset{\triangle}{LC'} = \frac{1}{H}\sum_{h\in\mathcal{H}} \overline{LC'}(h) = \frac{2\sum_{h\in\mathcal{H}} LC'(h)}{H^2(N-1)}. \quad (6)$$

Considering a partial deployment (i.e., looking at Eq. (6)), the average is computed only over nodes belonging to $\mathcal{H}$ (which are H instead of all N nodes); the normalization coefficient changes accordingly. We are interested in computing the average normalized relative error $E_{\mathcal{H}}$ defined as:

Definition 5 (Average Normalized Relative Error).

$$E_{\mathcal{H}} = \frac{\overset{\triangle}{LC'}}{\overset{\triangle}{LC}}. \quad (7)$$

We characterize analytically the relative error defined in Eq. (7), rewriting it in terms of two main factors that describe the "*missing load contributions*", which are lost because only a subset of nodes, namely $H \leq N$, perform the algorithm. This reformulation enables us to derive lower and upper bounds for $E_{\mathcal{H}}$ and understand how it converges to zero for H approaching N. To this purpose, we start computing the expected overall load in the network with the following theorem.

Theorem 2 (Overall Network Load).

$$\sum_{v\in\mathcal{V}} LC(v) = N(N-1)\bar{l}, \quad (8)$$

where $N(N-1)$ is the number of (s,d) pairs in $\mathcal{V}$ with $s \neq d$ and $\bar{l}$ is the average shortest path length in $\mathcal{V}$.

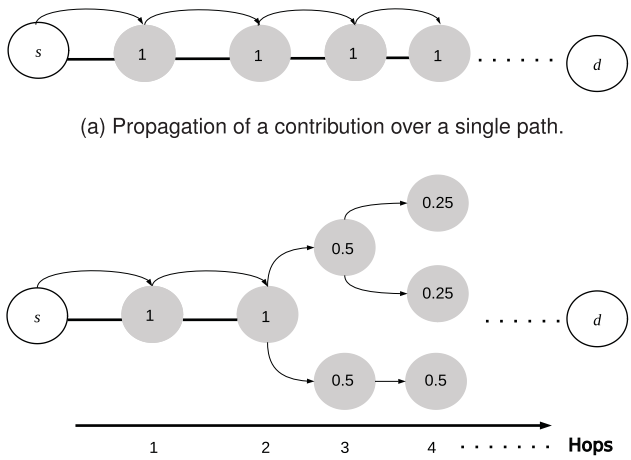
Proof. Consider a shortest path between s and d as a sequence of nodes $\{s,\dots d\}$ and the path length as $(|\{s\dots d\}|-1)$ without loss of generality.[2]

Consider the generic propagation of a commodity unit from node s to node d on a single shortest path as depicted in case Fig. 4a. All the nodes that forward the commodity increment their load centrality metric by the amount of commodity they have forwarded, which is 1. This means that the commodity sent by source s to target d, by traveling on the shortest path from s to d, induces an overall increment of load in the network equal to the length of the shortest path linking s to d $(|s\dots d|-1)$.

If there are multiple shortest paths, as in case of Fig. 4b, we assign load 1 to s itself (distance 0). At some distance $i > 0$ from s the path splits in k equivalent paths, then there are k nodes at distance $i+1$ from s, and by definition the sum of the load centrality of nodes at distance $k+1$ equals the sum of load centrality at distance k. By induction, given the load at distance 0 equal to 1, the sum of the load of the nodes at any distance $k+1$ is 1 and the total load induced on the network equals $(|s\dots d|-1)$.

Extending this observation to all the $N(N-1)$ possible (s,d) couples the *overall load in the network* is equal to the sum of the lengths of all the shortest paths, which by definition is $N(N-1)\bar{l}$. □

2. If there are multiple minimum shortest paths for some endpoints s and d, then they all have the same path length. For s directly connected to d the path length is 0. The distance definition implies that we exclude from the computation of LC for node v the contributions of paths terminating in v.



(a) Propagation of a contribution over a single path.

(b) Splitting of a contribution over multiple paths.

Fig. 4. Considering the load contribution derived from the commodity generated by s and sent to d, this contribution transits through all nodes drawn in solid color.

Combining Eqs. (8) and (5), we can rewrite the Average Normalized Load Centrality as

$$\overset{\triangle}{LC} = \frac{2\bar{l}}{N}. \quad (9)$$

We now need to reformulate the term $\sum_{h\in\mathcal{H}} LC'(h)$ in the right hand side of Eq. (6) adopting an approach similar to the one used to prove Theorem 2. In the partial deployment scenario, only shortest paths linking nodes $h \in \mathcal{H}$ to destinations in $v \in \mathcal{V}$ are accounted for to estimate the load due to the commodity propagation; consequently, the average path length $\bar{l}$ used in Theorem 2 should be computed on the proper subset of shortest paths that are actually used. We call $\bar{l'}$ the average path length from all sources $h \in \mathcal{H}$ to destinations in $v \in \mathcal{V}$. Furthermore, we measure the centrality for the subset of nodes in $\mathcal{H}$, which are the only ones implementing the algorithm, based only on the commodity they generate. This means we need to discard all contributions that transit through legacy nodes in $\mathcal{W}$ if we want to sum up only the load centrality computed by nodes in the $\mathcal{H}$ set. Taking into account these two considerations we define the Overall Estimated Load as follows:

Definition 6 (Overall Estimated Load).

$$\sum_{h\in\mathcal{H}} LC'(h) = H(N-1)\bar{l'} - \sum_{w\in\mathcal{W}} LC'(w). \quad (10)$$

We can now rewrite the definition of Average Normalized Partial Centrality applying Eq. (10) to Eq. (6)

$$\overset{\triangle}{LC'} = 2\frac{H(N-1)\bar{l'} - \sum_{w\in\mathcal{W}} LC'(h)}{H^2(N-1)}, \quad (11)$$

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

and this leads to the main reformulation of the Normalized Relative Error $E_{\mathcal{H}}$

$$E_{\mathcal{H}} = 1 - \frac{N}{\overline{l}} \left(\underbrace{\frac{\overline{l'}}{H}}_{E_1} - \underbrace{\frac{\sum_{w \in \mathcal{W}} LC'(w)}{H^2(N-1)}}_{E_2} \right), \tag{12}$$

where E_1 and E_2 are the two factors accounting for legacy nodes missing commodity.

6.1.2 Error Function Analysis

To gain insight into Eq. (12) we need to understand the roles of E_1 and E_2. Based on the definition from Eq. (10) we observe that E_1 describes the load produced by commodity generated by the upgraded nodes, while E_2 describes the amount of commodity that can not be measured (must be subtracted) because legacy nodes do not accumulate it as they do not understand its meaning.

We start the analysis setting $E_2 = 0$, which yields the error function in Eq. (13) and corresponds to a system where legacy nodes do not generate commodity and were chosen among the nodes that have zero centrality.

$$E_{\mathcal{H}} = 1 - \frac{N}{H} \frac{\overline{l'}}{\overline{l}}. \tag{13}$$

This is a relevant case because it is equivalent to the estimation of betweenness centrality using only a restricted number of single-source shortest-paths computations from a set of selected pivots [20]. This approach can be used with a centralized algorithm when the number of nodes is too large to compute all possible shortest paths. Assuming pivots are chosen at random, then $\overline{l'}$ quickly converges to $\overline{l}$ and the error is dominated by N/H. There are many works in the literature that deal with this problem, given that the execution of superlinear algorithms can be unfeasible when the graph is in the order of billions of nodes (see the work of Riondato *et al.* and cited bibliography [25]). Our problem is different because we run a distributed protocol, with legacy nodes not generating any load and not even estimating their centrality. This is why, in Eq. (10), we subtracted contributions traversing nodes in $\mathcal{W}$, and hence $E_2 \geq 0$.

Let us now go back to consider the original scenario in which we have a non-negligible number of legacy nodes that are not able to estimate their centrality. How big can E_2 be? Consider these two extreme cases:

1) $\mathcal{W}$ includes all and only the nodes with zero centrality;
2) $\mathcal{W}$ includes all and only the nodes with non zero centrality.

In the first case $\sum_{w \in \mathcal{W}} LC'(w) = 0$ and we fall back to Eq. (13). In the second case, the sum of the centrality of all nodes in $\mathcal{W}$ accounts for the overall load generated in the network, which is: $H(N-1)\overline{l'}$. Then $E_1 = E_2$ and the relative error is 1: Essentially we are trying to estimate the centrality of nodes in $\mathcal{H}$ which do not lie on any shortest path. No matter how many shortest paths we consider, the relative average error will be always 1. Consider for instance a star graph: for any size of $\mathcal{H}$, if $\mathcal{H}$ does not include the star center then the relative error will always be 1.

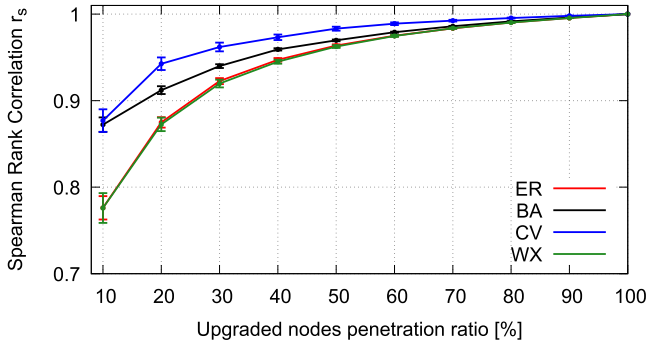


Fig. 5. Spearman's r_s for Barabási-Albert, Erdős, Waxman, and Caveman graphs for 1000 nodes and diameter 5 with 95 percent confidence intervals.

In between these extremes, we note that, from Eq. (10)

$$\overline{l'} = \frac{\sum_{h \in \mathcal{H}} LC'(h) + \sum_{w \in \mathcal{W}} LC'(w)}{H(N-1)} = \frac{\sum_{v \in \mathcal{V}} LC'(v)}{H(N-1)}, \tag{14}$$

then we have

$$E_1 = \frac{\overline{l'}}{H} = \frac{\sum_{v \in \mathcal{V}} LC'(v)}{H^2(N-1)} \geq E_2, \tag{15}$$

which means that $E_{\mathcal{H}}$ always lies in between the two extreme cases we just described, but also that upgrading more central nodes the error done is smaller.

Summing up, the convergence of the relative normalized error is quite slow. In the best case where $E_2 \simeq 0$, then $E_{\mathcal{H}} \simeq 1 - \frac{N}{H}$, meaning that with 80 percent nodes coverage we underestimate with a 25 percent error. Anyhow, the average error does not say much on how the ranking is influenced and, in general, centrality rankings are more important than the single centrality values. Estimating the error on rankings is less straightforward than the average error, but crucial. In Section 6.2 we compute rank correlation coefficients and show that, even for very small fractions of upgraded nodes, the algorithm preserves correct ranking with great accuracy.

6.2 Centrality Ranking with Partial Information

We implemented the new distributed, incrementally-deployable algorithm in a Python simulator. We used this simulator to study the Spearman's rank correlation coefficient r_s [43] and so evaluate the degree of similarity between rankings computed respectively with the exact and the approximated algorithms we provided (i.e., with Algorithms 1 and 3). Spearman's r_s measures the correlations of two rankings of the same population, and ranges in $[-1, 1]$. When $r_s = 1$ the two rankings are perfectly correlated, when $r_s = -1$ one ranking is perfectly specular to the other, when $r_s = 0$ there is no correlation between the two rankings.

We analyzed graphs generated using four well known generators: the already cited Barabási-Albert and Erdős plus the Waxman [44] and Caveman [45] models. As in the previous experiments, we generate 10 graphs of various size for each diameter (from 3 to 7, for a total of 50 graphs). The results shown in Fig. 5 are obtained following this methodology:

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

- For each graph we compute the exact centrality ranking using the algorithm with $\mathcal{H} = \mathcal{V}$;
- For each graph we also choose 5 instances of $\mathcal{H}$ at random, with $\mathcal{H}$ size ranging from 10 to 100 percent of $\mathcal{V}$ size, and we run a simulation. Each run produces an estimated ranking for the elements of $\mathcal{H}$.
- Finally, for each run we compute Spearman's coefficient comparing the estimated and the exact rankings (limited to the nodes in $\mathcal{H}$), and we average it on the 50 runs for that diameter (10 graphs times 5 random choices of $\mathcal{H}$).

For brevity, we report here only the results with 1,000 nodes and diameter 5, but we obtained results with similar trends also for all the other diameter values and with graphs of 400 nodes.

Fig. 5 reports the Spearman's correlation coefficient between the exact ranking and the estimated ranking, on the 4 families of graphs. It shows that for any kind of graph, with $\mathcal{H}$ covering 30 percent of $\mathcal{V}$, the correlation r_s is already close to 0.8, while with a coverage of 60 percent the correlation goes over 0.9 for all graphs. Fig. 5 confirms that estimated ranking accurately classifies nodes for their importance; they can therefore be used to fine-tune network protocols even in presence of a small fraction of updated nodes.

7 Tuning Babel With Centrality

Out of the many contexts in which our algorithm can be exploited, we focus on advantages it can give to optimize DV routing protocols in wireless mesh networks. To perform link sensing in wireless networks, routing protocols define a timer $t_{\mathtt{H}}$ used by each node to control the generation of link-level HELLO messages. This timer is crucial for a fast re-convergence in case of failures. A trade-off must be found between a short timer, which guarantees fast detection of link failures but subtracts link capacity to data traffic, and a long timer, that is more resource-aware but makes route convergence slow.

Maccari and Lo Cigno have introduced an optimization of $t_{\mathtt{H}}$ based on betweenness centrality [4]. Initially, they define the average overhead per link (O_H) when every node is configured with the same $t_{\mathtt{H}}$. Then, they introduce a loss metric to express the average estimated loss due to a node failure

$$L(k) = V_{\mathtt{H}} t_{\mathtt{H}}(k)\, N(N-1) b_k, \tag{16}$$

where $V_{\mathtt{H}}$ is a protocol parameter (the number of consecutive lost packets after which a link is considered broken). Finally they show that keeping O_H constant, the average loss can be minimized if $t_{\mathtt{H}}$ is configured per-node as follows:

$$t_{\mathtt{H}}(i) = \frac{\sqrt{d_i}}{\sqrt{b_i}} t_{\mathtt{H}} \frac{\sum_{j=1}^{N} \sqrt{b_j d_j}}{\sum_{j=1}^{N} d_j} \propto \frac{\sqrt{d_i}}{\sqrt{b_i}}, \tag{17}$$

where b_i and d_i are the betweenness and degree of node i. In practice, if a node knows d_i and b_i for all nodes, it can autotune its $t_{\mathtt{H}}(i)$ to achieve a convergence time distribution that minimizes the average network disruption after a node failure, keeping a constant signaling overhead in the network.

The authors used this technique for the OLSR protocol but, in principle, it can be applied to any link-state protocol where every node is aware of the whole topology. Conversely, it cannot be applied to DV routing protocols because, in this latter case, nodes have a limited topological knowledge and cannot compute Eq. (17). Algorithm 1 does not mandate nodes to know the entire topology and it is fully distributed, therefore its implementation on a DV routing protocol is straightforward and we effectively integrated it with the Babel protocol [15], a well-known DV routing protocol.[3]

We implemented the distributed centrality computation algorithm in `babeld`, the open source implementation of Babel, in order to verify that centrality can be correctly computed. The evaluation strategy and performance metrics are those proposed by the authors of [4]; here, we just briefly recall them, while a detailed description is provided in the original paper. Finally, note that to use Eq. (17), the propagated distance vector should also contain the degree of node i. If we drop this requirement, then we can set $t_{\mathtt{H}}(i) = \frac{\sqrt{t_i}}{\sqrt{b_i}} K$ for some constant K. This will still optimize the timers and keep a constant level of global overhead, but it will not produce exactly the same overhead O_H of the default configuration. In return, it greatly simplifies the protocol as long as every node can take decisions based only on its own centrality. For our purpose, we used the value of K that generates the value of O_H corresponding to $t_{\mathtt{H}} = 1s$.

We run the code in an emulated network using the Mininet platform. At time t_f, we trigger the failure of node k, and we let all routing tables stabilize again; we repeat this procedure for a subset of $N_f \leq N$ nodes. Note that generally $N_f < N$ because we exclude two categories of nodes: leaf nodes (their centrality is zero so their failure does not impact any other node) and cut-points (nodes whose centrality may be high but they partition the graph in disconnected components, so that it is not possible to route around the failed node). During experiments we dump the routing tables $RT_i^j[d]$ every 0.5 s: a dump contains the matching between a destination d and the next hop nh for node i at time t_j (only one path is used in Babel). When the emulation is over, we group the dumped routing tables according to timestamps, next we recursively navigate each group to take a snapshot of all shortest paths from every source s to any destination d. We call L_j the number of broken shortest paths that, for $t_j > t_f$, are incomplete or still pass through node k and we define $L_{babel}(k) = \sum_j L_j$ the total loss value when the emulation runs with the original `babeld` and $L_{cent}(k)$ the same value but computed using the optimized timers. Finally we compare the two approaches, computing the relative loss value averaged over all possible failures as

$$L = 1 - \frac{\sum_{k=0}^{N_f} L_{cent}(k)}{\sum_{k=0}^{N_f} L_{babel}(k)}. \tag{18}$$

3. Eq. (17) uses *betweenness* centrality, while our approach computes *load* centrality. However, in a mesh network links are weighted by their quality (with any metric the protocol supports), which makes it hard to have multiple paths with the same exact weights, therefore, in real-world mesh networks load centrality converges to betweenness centrality. This said, we do not attempt any comparison with [4]: comparing a DV and a link-state protocol goes well beyond comparing centrality metrics.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
Loss Reductions in Real Networks

Network	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	N_f	Loss Reduction	Type
Interoute	110	148	63	8.37%	Wired
Ion	125	146	58	3.10%	Wired
GtsCe	149	193	98	6.05%	Wired
TataNld	145	186	68	7.34%	Wired
Ninux	126	147	17	10.65%	Wireless
FFGraz	141	200	19	13.11%	Wireless
Auerbach	123	223	70	11.29%	Heterogeneous
Adorf	123	225	65	13.27%	Heterogeneous

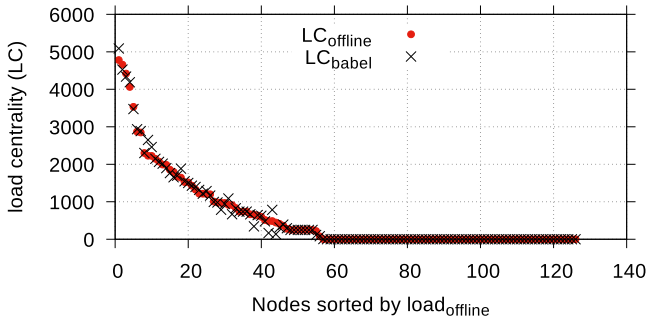


Fig. 6. Comparison of LC computed on-line with and off-line with networkx on the same topology computed by Babel.

If $L > 0$, then the tuned version of babeld, averaged over N_f failures, produces lower loss compared to the non-modified version, always keeping the same overhead due to control messages.

Before presenting detailed results, it is worthwhile to discuss some modifications to Algorithm 1 that are necessary to implement it in babeld and are in general required for any real DV protocol.

Nodes versus Routers. The common approach it the literature focused on centrality is to treat nodes as sources, targets and forwarders of traffic. In real networks, sources and targets are IP addresses and routers have several interfaces with distinct IP addresses. To overcome this issue, we aggregate all route-updates coming from the same node based on the "router-id" field defined by Babel to uniquely identify a router. This field is included in all packets generated by a router and is propagated by all others, therefore we can aggregate the centrality contributions pertaining to different interfaces of the same router and do a mapping between IP addresses and graph nodes.

Load Estimation. In our implementation, every router generates a unit of traffic $\theta_{s,d} = 1$, but in real networks this value can be arbitrarily tuned. It can be proportional to the dimension of attached subnets (assuming more IPs will generate more traffic) or it can be replaced with an estimation of the real outgoing traffic measured locally. This way load centrality would effectively represent the expected load on the node.

Protocol-Specific Heuristics. In our tests we used networks that have more than one shortest path with the same weight connecting the same endpoints (s, d). The version of Babel that we used does not support multipath routing: in these cases Babel performs a tie-break to select one path over another. We also noticed that babeld sometimes selects paths that are not minimum weight. This is probably due to an implemented heuristic that prevents changing from one path to another if their weight is similar, just to avoid route flapping. Our algorithm follows choices taken by babeld, which is the correct behavior on-line even if the computed LC minimally diverges from the theoretical one. Section 7.1 further details and explains this issue.

7.1 Experimental Results With Full Coverage

We test the protocol on several topologies extracted from real-world networks [46]. Two of them are *Ninux* and *FFGraz*[4] the same ones used and published by the authors of [4], which are topologies of two large-scale wireless mesh networks. We were able to collect two more real network topologies, namely *Auerbach* and *Adorf*, analysing information provided by the FreiFunk German community network (CN). FreiFunk is an umbrella name that gathers together hundreds of wireless CNs in Germany: some of them are made of few nodes, some others are made of hundreds, all of them are mesh networks used to offer Wi-Fi connectivity. Information on these network topologies is freely available (with some effort to understand the format) from the community website.[5] In the particular case of *Auerbach* and *Adorf*, these two are heterogeneous networks, with a mix of wireless and wired links inside a single routing domain. Finally, we also use 4 others extracted from the well-known Topology Zoo [47], namely *Interoute, Ion, GtsCe* and *TataNld*; these are 4 wired topologies of similar size that we use to extend our analysis. Table 3, discussed later on, reports the names of all the 8 topologies with their key characteristics and the loss reduction.

Babel is an event-driven protocol where messages are sent in reaction to detected changes and expiration of local timers. This, together with the heuristics mentioned in the previous section, introduces slight differences in the centrality computation compared to the ideal protocol presented in Section 3. Fig. 6 presents a validation of our implementation in babeld and reports a comparison between empirical and theoretical LC values computed for all nodes in a network. On the one hand, LC has been computed on-line using the modified babeld and, on the other hand, running Python networkx libraries off-line on the same topology built by babeld and saved in JSON format. The *Mininet* network emulator allows running experiments with real instances of babeld over real-world networks.[6] For instance, the results shown in Fig. 6 are obtained running experiments on the topology of ninux, a CN of Rome.

First of all we verified that the sum of all LC values is identical in both cases, which means that Babel never uses a minimum weight path that is longer (in terms of hops) than the shortest one computed by networkx. Next, we noticed that nodes' rankings are not exactly the same, but very similar.

Fig. 7 reports the number of broken paths versus time after one of the most central nodes of the ninux topology has failed around time $t = 5$ s. A path is said to be broken if

4. https://www.ninux.org — https://graz.funkfeuer.at/en/about.html

5. See https://api.freifunk.net, and the visualizer https://www.freifunk-karte.de.

6. Experiments cannot be run directly on working networks to avoid disrupting their daily functioning.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

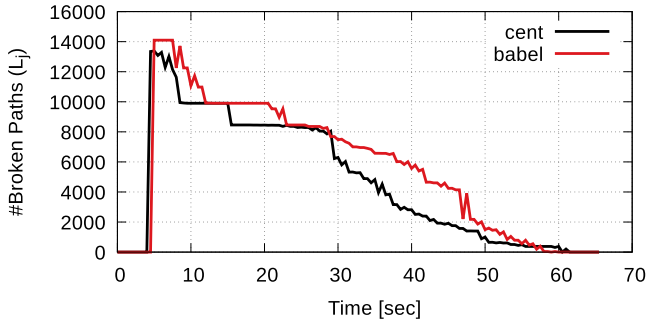


Fig. 7. Comparison of the network loss in ninux topology after the failure of one of the most central nodes.

it contains a node with an invalid next-hop. Babel with centrality reacts slightly faster, but above all recovers more routes in less time compared to standard Babel. Thus we achieve a resilience gain without adding any cost, as the complexity of LC computation is minimal: in fact, the signaling overhead in terms of number of messages is constant, while messages' dimension increases only marginally.

To get exhaustive results, we run the modified version of `babeld` over a total of 8 emulated networks representing real topologies. Table 3 and Fig. 8 report the results summary. Fig. 8 compares $L_{babel}(k)$ and $L_{cent}(k)$ where nodes $k = 1, 2, \ldots, 15$ are the 15 most central ones for each topology. The chosen networks are ninux (Fig. 8a), Graz (Fig. 8b), and Ion (Fig. 8c), because they represent well different classes of networks; however results would not change significantly selecting other networks. As we can see, in general L_{cent} is smaller than L_{babel}, but sometimes the fine-tuned timers' frequency does not provide any gain. However, a closer look to Table 3, which reports the mean loss reduction for all the 8 considered networks, reveals that averaging over all possible failures we obtain a global gain, ranging from 3 percent up to 13 percent depending on the topology; still it is always a clear advantage in favor of tuning timers based on centrality.

In general, wireless and heterogeneous networks achieve larger gains compared to wired and uniform networks due to structural properties of the network graphs. In fact, the optimization level that can be achieved exploiting centrality strongly depends on the array of values of b_i and d_i and on the availability of alternative paths to route around a failure. Consider the extreme case of a ring network, or in general an n-regular network over a torus: in such networks all nodes have same degree and centrality and Eq. (17) returns the same value for all timers. In these cases no optimization is possible.

7.2 Analytical Results With Partial Coverage

We conclude this section showing we can obtain a performance improvement even when only a subset of nodes support centrality-based optimization, applying the algorithms explained in Section 6. We use the Python simulator introduced in Section 6.2 with the following procedure. Given a network with N nodes and $\rho \in (0, 1]$, we randomly select $\lceil \rho N \rceil$ nodes that run Algorithm 3, while all the other nodes run Algorithm 2. We let the network converge and we obtain the estimated values of centrality b_k for the upgraded nodes. Then we compute the average loss exactly as explained in Section 7, but, instead of running full emulations, for performance reasons we compute L_{babel} and L_{cent} in Eq. (18) using

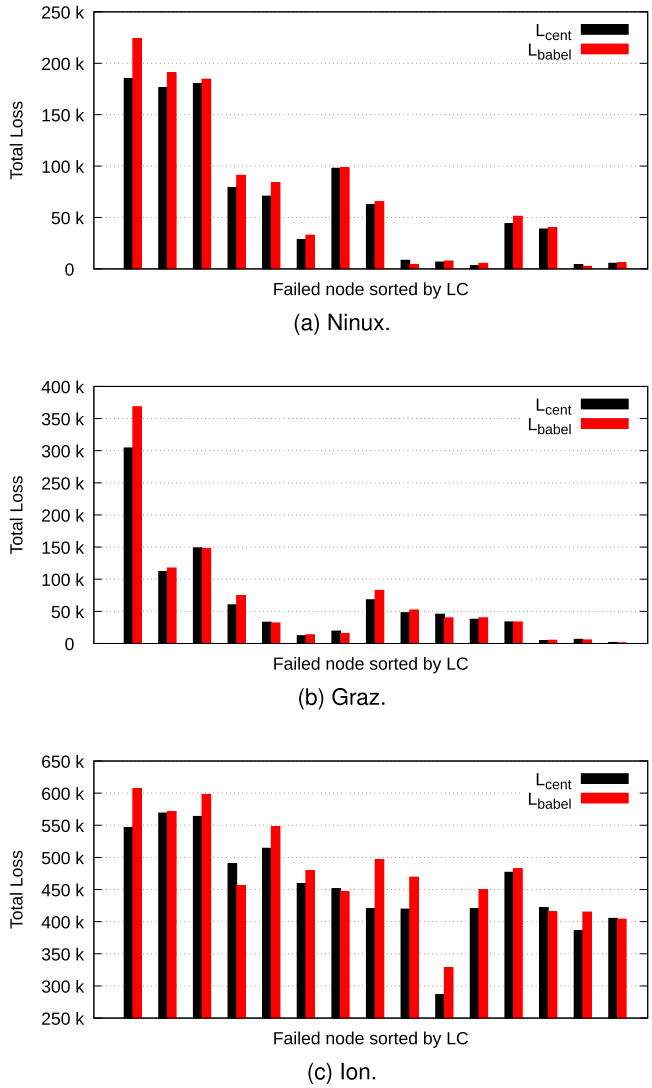


Fig. 8. Comparison of the loss induced by the failure of the 15 more central nodes in ninux (a) Graz (b) and Ion (c) when standard Babel is used (L_{babel}) or the modified version is used (L_{cent}).

Eq. (16). The different methodology makes the analytical value of L different from the data we obtain with emulations. These results must be considered as an upper bound of the possible improvement, and thus Fig. 9 is not directly comparable with Table 3. On the other hand, this allows repeating the process for 40 times, for ten values of ρ and for all the networks in Table 3 in reasonable time.

Fig. 9 shows that with ρ larger than 0.2, in all networks we start to have a tangible improvement compared to the standard behavior. This confirms that even if the convergence to the exact centrality values is slow, it it sufficiently precise to improve the protocol performance with as little as 20 percent of the upgraded nodes. Such a result is very encouraging as it finally shows that we can deploy centrality-based optimizations in an incremental way on existing networks, and obtain a net improvement way before the full coverage is reached. This is a key step in the path to improve protocols in large-scale existing networks.

The fact that some of the curves are not monotonically growing is probably a combination of several effects. First, when we apply the improved protocol to a subset of nodes,

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

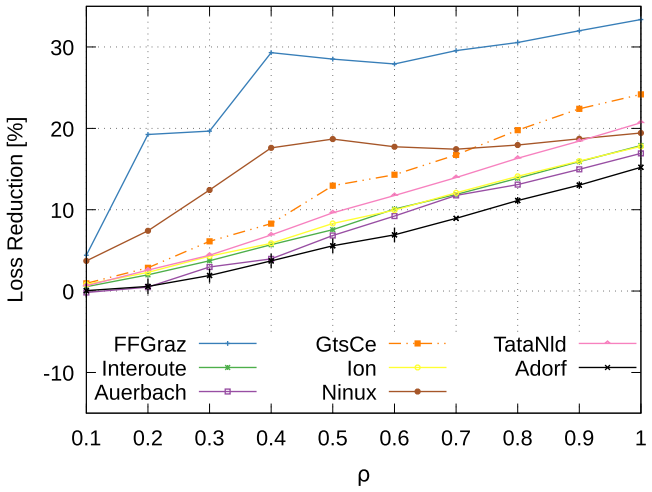


Fig. 9. Comparison of loss on various networks with only a fraction ρ of nodes supporting the improved protocol. For clarity, we report the 95 percent confidence interval for Adorf only, which has the largest ones in average. The intervals are barely visible.

we cannot strictly enforce the condition of constant average overhead O_H. We expect this condition to hold in average as ρ approaches 1. Second, due to the specific properties of each topology the convergence of the centrality estimation may be faster or slower, and could even suffer from a bias. As a result it could be that in Ninux and FFGraz for some choices of ρ we slightly overestimate centrality, and thus we produce a higher gain than with larger values of ρ.

8 Conclusion

Centrality metrics are key for understanding the importance of a node in a network, and they have been extensively used in many scientific fields. Betweenness and load centrality are two of the most popular ones. In networks that do not support multipath routing the two metrics coincide, and this is the case for most communication networks.

In spite of its importance, and before this work, there was no fully distributed algorithm that supports the computation of load centrality in generic networks. In fact, among existing algorithms there are those requiring a full topological knowledge, those that are distributed but only approximated, and those that are exact and distributed but applicable only on special topologies (such as DAGs or trees). For this reason so far it was impossible to exploit betweenness or load centrality in distributed network protocols.

This paper contributes, to the best of our knowledge, the first algorithm for the exact computation of load centrality in a generic graph. We demonstrated its convergence, the worst case convergence time, and we showed it can be directly integrated with minimal modification into a distance-vector (DV) routing protocol. We provided a direct use-case implementing the distributed algorithm in Babel, a widely used standard DV protocol, showing it can tangibly improve the convergence time in case of nodes' failure for all tested topologies, taken from real networks.

Finally, the algorithm does not require all nodes in the network to support it; it can be gradually deployed in an existing network and even with a small fraction of upgraded nodes it yields useful rankings for node centrality. Many more applications than routing can benefit from nodes' rankings based on centrality. Caching is the most obvious, but not the only one. We believe that the availability of efficient centrality computation algorithms can spawn research and applications exploiting it.

Acknowledgments

This work has been partially funded by the European Commission, H2020-ICT-2015 Programme, Grant Number 688768 'netCommons' (Network Infrastructure as Commons) and the H2020 GA No. 645274 "Wireless Software and Hardware platforms for Flexible and Unified radio and network controL (WiSHFUL)" with the project "Pop-Routing On WiSHFUL (POPROW)" financed in Open Call 3. Lorenzo Ghiro PhD grant is partially supported by the IEEE Smart Cities Initiative at Trento. This article revises and extends a article presented at IEEE INFOCOM in 2018 [1], by studying the behavior of the protocol under partial information. This work was initiated when all the authors were with the University of Trento.

References

[1] L. Maccari, L. Ghiro, A. Guerrieri, A. Montresor, and R. Lo Cigno, "On the distributed computation of load centrality and its application to DV routing," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2018, pp. 2582–2590.

[2] U. Brandes, "On variants of shortest-path betweenness centrality and their generic computation," *Soc. Netw.*, vol. 30, no. 2, pp. 136–145, May 2008.

[3] P. Pantazopoulos, M. Karaliopoulos, and I. Stavrakakis, "Distributed placement of autonomic internet services," *IEEE Trans. Parallel Distrib. Syst*, vol. 25, no. 7, pp. 1702–1712, Jul. 2014.

[4] L. Maccari and R. Lo Cigno, "Pop-routing: Centrality-based tuning of control messages for faster route convergence," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[5] S. Dolev, Y. Elovici, and R. Puzis, "Routing betweenness centrality," *J. ACM*, vol. 57, no. 4, pp. 25:1–25:27, Apr. 2010.

[6] L. Maccari and R. LoCigno, "Improving routing convergence with centrality: Theory and implementation of pop-routing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 5, pp. 2216–2229, Oct. 2018.

[7] A. Vázquez-Rodas and L. J. de la Cruz Llopis, "A centrality-based topology control protocol for wireless mesh networks," *Ad Hoc Netw.*, vol. 24.B, pp. 34–54, Jan. 2015.

[8] L. Baldesi, L. Maccari, and R. LoCigno, "On the use of eigenvector centrality for cooperative streaming," *IEEE Commun. Lett.*, vol. 21, no. 9, pp. 1953–1956, Sep. 2017.

[9] P. Zilberman, R. Puzis, and Y. Elovici, "On network footprint of traffic inspection and filtering at global scrubbing centers," *IEEE Trans. Dependable Secure Comput.*, vol. 14, pp. 521–534, Sep./Oct. 2017.

[10] L. Maccari, Q. Nguyen, and R. Lo Cigno, "On the computation of centrality metrics for network security in mesh networks," in *Proc. IEEE Global Commun. Conf.*, 2016, pp. 1–6.

[11] M. Kas, S. Appala, C. Wang, K. M. Carley, L. R. Carley, and O. K. Tonguz, "What if wireless routers were social?" *IEEE Wireless Commun.*, vol. 19, no. 6, pp. 36–43, Dec. 2012.

[12] L. Maccari and R. LoCigno, "Betweenness estimation in OLSR-based multi-hop networks for distributed filtering," *J. Comput. Syst. Sci.*, vol. 80, no. 3, pp. 670–685, May 2014.

[13] D. Papakostas, S. Eshghi, D. Katsaros, and L. Tassiulas, "Energy-aware backbone formation in military multilayer ad hoc networks," *Ad Hoc Netw.*, vol. 81, pp. 17–44, Dec. 2018.

[14] K. I. Goh, B. Kahng, and D. Kim, "Universal behavior of load distribution in scale-free networks," *Phys. Rev. Lett.*, vol. 87, no. 27, pp. 1–4, Dec. 2001.

[15] J. Chroboczek, "The babel routing protocol," Univ. Paris, Paris, France, Tech. Rep. RFC 6126, Apr. 2011.

[16] R. Puzis, M. Tubi, Y. Elovici, C. Glezer, and S. Dolev, "A decision support system for placement of intrusion detection and prevention devices in large-scale networks," *ACM Trans. Model. Comput. Simul.*, vol. 22, no. 5, pp. 1–26, Dec. 2011.

[17] U. Brandes, "A faster algorithm for betweenness centrality," *J. Math. Sociology*, vol. 25, no. 2, pp. 163–177, 2001.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.

[18] R. Puzis, P. Zilberman, Y. Elovici, S. Dolev, and U. Brandes, "Heuristics for speeding up betweenness centrality computation," in *Proc. ASE/IEEE Int. Conf. Soc. Comput. Int. Conf. Privacy Secur. Risk Trust*, 2012, pp. 302–311.
[19] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail, "Approximating betweenness centrality," in *Proc. 5th Int. Conf. Algorithms Models Web-Graph*, 2007, pp. 124–137.
[20] U. Brandes and C. Pich, "Centrality estimation in large networks," *Int. J. Bifurcation Chaos*, vol. 17, no. 7, pp. 2303–2318, Jul. 2007.
[21] R. Geisberger, P. Sanders, and D. Schultes, "Better approximation of betweenness centrality," in *Proc. SIAM Meeting Algorithm Eng. Expermiments*, 2008, pp. 90–100.
[22] R. Jacob, D. Koschützki, K. A. Lehmann, L. Peeters, and D. Tenfelde-Podehl, "Algorithms for centrality indices," in *Network Analysis*, U. Brandes and T. Erlebach, Eds. Berlin, Germany: Springer, 2005.
[23] Y. Lim, D. S. Menasché, B. Ribeiro, D. Towsley, and P. Basu, "Online estimating the k central nodes of a network," in *Proc. IEEE Netw. Sci. Workshop*, 2011, pp. 118–122.
[24] A. Maiya and T. Y. Berger-Wolf, "Online sampling of high centrality individuals in social networks," in *Proc. Pacific-Asia Conf. Knowl. Discov. Data Mining*, 2010, pp. 91–98.
[25] M. Riondato and E. Upfal, "ABRA: Approximating betweenness centrality in static and dynamic graphs with rademacher averages," in *Proc. Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 1145–1154.
[26] M. Baglioni, F. Geraci, M. Pellegrini, and E. Lastres, "Fast exact computation of betweenness centrality in social networks," in *Proc. IEEE Int. Conf. Advances Soc. Netw. Anal. Mining*, 2012, pp. 450–456.
[27] E. Bergamini and H. Meyerhenke, "Fully-dynamic approximation of betweenness centrality," in *Proc. 23rd Eur. Symp. Algorithms*, 2015, pp. 155–166.
[28] E. Bergamini, H. Meyerhenke, and C. L. Staudt, "Approximating betweenness centrality in large evolving networks," in *Proc. 17th SIAM Workshop Algorithm Eng. Experiments*, 2014, pp. 133–146.
[29] N. Kourtellis, G. De Francisci Morales, and F. Bonchi, "Scalable online betweenness centrality in evolving graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 9, pp. 2494–2506, Apr. 2015.
[30] Y. Yoshida, "Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2014, pp. 1416–1425.
[31] K. You, R. Tempo, and L. Qiu, "Distributed algorithms for computation of centrality measures in complex networks," *IEEE Trans. Autom. Control*, vol. 62, no. 5, pp. 2080–2094, May 2017.
[32] W. Wang and C. Y. Tang, "Distributed computation of node and edge betweenness on tree graphs," in *Proc. 52nd IEEE Conf. Decis. Control*, 2013, pp. 43–48.
[33] W. Wang and C. Y. Tang, "Distributed computation of classic and exponential closeness on tree graphs," in *Proc. Amer. Control Conf.*, 2014, pp. 2090–2095.
[34] Q. S. Hua *et al.*, "Nearly optimal distributed algorithm for computing betweenness centrality," in *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst.*, 2016, pp. 271–280.
[35] M. Pontecorvi and V. Ramachandran, "Distributed algorithms for directed betweenness centrality and all pairs shortest paths," 2018, *arXiv: 1805.08124*.
[36] L. Hoang *et al.*, "A round-efficient distributed betweenness centrality algorithm," in *Proc. 24th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2019, pp. 272–286.
[37] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics (SIAM), Jan. 1987.
[38] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, Mar. 1977.
[39] R. Stewart, "Stream control transmission protocol," RFC 4960, IETF, to be published, doi: 10.17487/RFC4960.
[40] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," RFC 6824, IETF, to be published, doi: 10.17487/RFC6824.
[41] Y. Rekhter and T. Li,"A border gateway protocol 4 (BGP-4), section 5," RFC 1771, IETF, to be published, doi: 10.17487/RFC1771.
[42] B. Carpenter, B. Aboba, and S. Cheshire, "Design considerations for protocol extensions," RFC 6709, IETF, to be published, doi: 10.17487/RFC6709.
[43] W. W. Daniel, *Applied Nonparametric Statistics*, 2nd ed. Belmont, CA, USA: Wadsworth Pub Co., Aug. 1989.
[44] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Sel. Areas Commun.*, vol. 6, no. 9, pp. 1617–1622, Dec. 1988.
[45] D. J. Watts, "Networks, dynamics, and the small-world phenomenon," *Amer. J. Sociology*, vol. 105, no. 2, pp. 493–527, 1999.
[46] L. Maccari and R. LoCigno, "A week in the life of three large Wireless Community Networks," *Ad Hoc Netw.*, vol. 24, no. Part B, pp. 175–190, 2015.
[47] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

Leonardo Maccari (Member, IEEE) received the PhD degree with the Faculty of Computer Science Engineering from the University of Florence, Florence, Italy, in 2010 and he is currently an associate professor at the University of Venice, Italy. He has co-authored about 60 publications among refereed conferences, journals, book chapters, and patents. His research interest include network protocols and privacy in large-scale wireless mesh networks with special focus on Community Networks. He is an ACM member.

Lorenzo Ghiro received the master's degree with honours, with a thesis on routing optimization based on centrality metrics, in 2017. He is working toward the PhD degree in computer science at the University of Trento, Italy. His research interests include graph analysis and network algorithms.

Alessio Guerrieri received the master's degree in computer science from the Georgia Institute of Technology, Atlanta, Georgia, with a thesis on distributed clustering algorithms, and the PhD degree on large scale distributed graph processing from the University of Trento, Trento, Italy, in 2015. He is a data scientist at SpazioDati SRL, a small innovative company in Trento, Italy. His research interests include graph algorithms and distributed systems.

Alberto Montresor is an associate professor with the University of Trento, Trento, Italy since 2005. He has previously been with the University of Bologna, Bologna, Italy, (2002–2005). He has authored more than 100 papers on large-scale distributed systems, cloud computing and P2P networks. He is also an associate editor of Springer Computing, and he served as steering committee chair of the IEEE Conference on P2P Computing and as general chair and program chair for DOA, DAIS, SASO and P2P.

Renato Lo Cigno (Senior Member, IEEE) received the degree in electronic engineering with a specialization in telecommunications from Politecnico di Torino, Turin, Italy. He is full professor with the University of Brescia, Italy. During 1998 to 1999 he was a visiting scholar at UCLA; from 2002 to 2019 he was with the University of Trento, Italy. He has been general chair of IEEE P2P, ACM WMASH and IEEE WONS, TPC chair of IEEE VNC, ACM WMASH, IEEE MedComNEt, and IEEE WONS. He is an associate editor for the *IEEE/ACM Transactions on Networking*. His research interests include performance evaluation of wired and wireless networks, modeling and simulation techniques, congestion control, and P2P networks and networked systems in general. He is a senior member of ACM and has co-authored around 200 papers in peer reviewed journals and conferences.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: UNIVERSITA TRENTO. Downloaded on April 12,2020 at 09:26:26 UTC from IEEE Xplore. Restrictions apply.