On the Computation of Centrality Metrics for Network Security in Mesh Networks

Leonardo Maccari, Quynh Nguyen, Renato Lo Cigno
DISI – University of Trento, Italy

***Abstract*—Betweenness centrality is a popular metric in social science, and recently it was adopted also in computer science. Betweenness identifies the node, or the nodes, that are most suitable to perform critical network functions, such as firewalling and intrusion detection. However, computing centrality is resource-demanding, we can not give for granted that it can be computed in real time at every change in the network topology. This is especially true in mesh networks that generally use devices with few computation resources. This paper shows that using the fastest state-of-the-art heuristic algorithm, it is indeed possible to compute network centrality even in real, low-power networking hardware in a network made of up to 1000 nodes. The observation of a real mesh network not only shows that centrality does not need to be updated at every topology change, but also that it can be safely re-computed with an interval in the order of the tens of minutes. Our findings confirm that centrality can be effectively and successfully used as a building block for security functions in mesh networks.**

I. Introduction

Network centrality metrics measure how much a node can be considered in the *core* or in the *periphery* of the network graph. In the past forty years they have been largely used in social science to identify the individuals that play an important role in a social network [1]. Depending on the definition of centrality, an individual with high centrality is the one that can behave like a broker or a mediator in the network, or that can reach the other individuals with the minimum number of "hops".

Centrality metrics have been recently introduced in computer networks, that are the focus of this paper, and thus we use the term network or graph centrality as equivalent ones. In networks, centrality can be used to study the robustness of the network topology or to identify the nodes in the network that represent points of failure. This kind of analysis can be carried on off-line, once the network topology is known, as social scientist do on social networks [2] and it is helpful to identify network nodes or links that need to be treated with special attention by the network manager.

Recently, some research works proposed to compute centrality in real-time to adapt network functions to topological changes, particularly to wireless mesh network. Two relevant examples are firewalling and Intrusion Detection (ID): In a wireless mesh network there is not a single point where all the traffic is forced to pass and it is impractical to enforce those functions on all the nodes for performance reasons (we will review the related literature in section II-A). A possible strategy consists of choosing the node, or the subset of nodes that is able to monitor the largest possible fraction of the whole network traffic. This is exactly what the so-called Betweenness Centrality (BC) identifies and represents, and it is the focus of this paper. However, in wireless mesh networks link conditions may vary radically and fast due to shadowing, failures, overload of the links, addition or removal of nodes. At every topology change, the centrality metrics need to be re-computed: A node that was central may become peripheral, and it can be convenient to disable firewalling and ID on that node and enable these functions on another one.

Computing BC can be done in polynomial time. So in principle, it is not too computationally intensive. Nevertheless, recent technological advances allow in-production mesh networks to grow to encompass hundreds of wireless nodes [3], and each node is realized with off-the-shelf hardware with very limited computing power. Thus, one key question addressed by this paper is: How time-consuming is it to compute centrality metrics? We answer this question by implementing the fastest state-of-the-art heuristic algorithm [4] and testing it on a wireless router frequently used in mesh networks. We analyze the performance of the algorithm applied to synthetic topologies and to real topologies, and we show that computation of centrality metrics is possible in real hardware, but still requires a non-negligible amount of time.

Thus, a second question that this paper asks and answer to is: How often do we need to re-compute betweenness centrality of a node in real networks? We answer this question by analyzing the data extracted from one real, in-production, wireless mesh network, and showing that changes are relatively slow. Thus, we conclude that even in large networks BC can be computed in soft real-time, which reinforces the research that is being done in this field.

II. Background and Contribution

In this section we review some works in the literature that apply centrality to networking problems, and to security in particular. Then, we recap the state-of-the-art algorithm for the computation of BC, which finally helps us to detail the contribution of this paper.

A. Centrality and Networking

In a typical network set-up firewalls and intrusion detection systems ("security functions" from now on, for simplicity)

This work was financed partially by the University of Trento under the grant "Wireless Community Net-works: A Novel Techno-Legal Approach" - Strategic Projects 2014, and partially by the European Commission, H2020-ICT-2015 Programme, Grant Number 688768 'netCommons' (Network Infrastructure as Commons).

are placed on the gateway nodes where they can collect the whole traffic entering and exiting a network. In a decentralized network instead, there is not one single place where the network manager can enforce security functions to control 100% of the traffic. The administrator should place security functions on every node in the mesh. However these functions are resource-demanding, and when implemented in hardware with limited computational power they can introduce relevant delays, and possible instabilities [5]. Therefore alternative, collaborative approaches were proposed in the literature [6], [7], [8].

Some proposals identify only a subset of the nodes on which the security functions can be enabled, and leverage the concept of betweenness centrality, to select the most suitable ones. Centrality, and BC in particular has been used in social science since the '70s to identify the most influential elements in social networks [1], but was not applied to communication networks until recently [2]. A centrality based approach to place network monitors was used by Dolev et. al. for performance monitoring [9], [10], but also for intrusion detection [11]. In [5] a dynamic, centrality-based firewall system is introduced, and more recently centrality has been used for cloud-based filtering in large networks [12]. While these approaches do not allow the inspection of 100% of the traffic, they are still effective. In fact, in real networks made of hundreds of nodes as few as 5 nodes are in the position to intercept about 90% of the potential traffic flows [3]. Besides security applications, centrality can help resources allocation [13], can be used to evaluate network robustness [14], to perform real-time dynamic allocation of services in a network [15], to optimize the configuration of link-state routing protocols [16] and to coordinate peer-to-peer protocols [17], [18].

Many of these proposals rely on the capacity of each network node to compute BC, that is possible under two conditions: i) every node must be aware of the full network topology, and ii) the computation must be practical on real hardware devices. The first condition is easily achieved when the network uses a link-state routing protocol, such as OSPF for a wired network or OLSR for a wireless network. Even in networks using distance-vector protocols, some other instruments are often used to reconstruct the topology, since it is useful to perform network management and troubleshooting. The second condition is what the rest of this paper explores.

B. Algorithms for BC Computation

Betweenness centrality is the fraction of all the shortest paths that pass through a single node. Given a network graph $G(\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ is the set of all nodes (of size N) and $\mathcal{E}$ is the set of edges (of size E) we call $p_{i,j} = \{n_i, \ldots, n_j\}$ the sequence of nodes that represent the shortest path from node n_i to node n_j. Since the graph represents an IP network we assume that at each instant only one shortest path from n_i to n_j is used, so we consider $p_{i,j}$ unique. We call b_k the betweenness centrality of node n_k defined as:

$$b_k = \frac{1}{N(N-1)} \sum_{i=1}^{i=N} \sum_{\substack{j=1 \\ j \neq i}}^{j=N} I_{p_{i,j}}(n_k) \qquad (1)$$

Where $I_{p_{i,j}}(n_k)$ is the indicator function that takes value 1 if $n_k \in p_{i,j}$ and 0 otherwise. In a directed connected graph without self loops, for each node n_k there are exactly (N-1) destinations. At one extreme, if n_k is a leaf node there are 2(N-1) paths in which n_k is present[1]. At the other extreme, if n_k is the center of a star topology, n_k is present in all the N(N-1) possible shortest paths, thus $b_k \in [\frac{2}{N}, 1]$.

Given a weighted graph with N nodes and E links, the computation of the shortest path rooted at a node with Dijkstra's algorithm scales as $\mathcal{O}(E + Nlog(N))$. BC computation with a straightforward application of Dijkstra's algorithm scales as $\mathcal{O}(N^3)$. The fastest known algorithm was proposed by Brandes and computes the exact centrality with a complexity of $\mathcal{O}(EN + N^2log(N))$ [19].

Recently, Puzis et al. proposed two heuristic speed-ups for centrality computation [4]. The first one is based on the aggregation of nodes that are structurally equivalent (they have the same neighbors and link weights). In a mesh networks links are weighted based on some link-quality metric that is dynamically computed by the nodes and depends on a multitude of factors (antenna, radio, frequency, traffic…) which makes it unlikely to have nodes that are structurally equivalent. For this reason we do not use this heuristic and we focus only on the second one which we abbreviate as HBC, "heuristic BC" (and we call BBC the original Brandes' algorithm). To describe HBC we need to introduce some terminology:

- A cut-point n_i in a graph G is a node that, if removed, partitions the original graphs in two or more components.
- A bi-connected graph is a graph with no cut-points (i.e. at least two nodes must be removed to partition the graph).
- An induced subgraph B of G is called a bi-connected component of G if two properties are true: i) it is bi-connected ii) any connected component $B' \supset B$ is not bi-connected.

HBC is made of the following steps:

1) Identify the bi-connected components of the network graph, compress each bi-connected component into a block, decompose the network graph into a block-cut-points tree. Any graph can be represented as a tree made of blocks that correspond to each bi-connected component connected via cut-points, and this operation can be performed in linear time. An example taken from the original paper is reported in fig. 1.
2) For each bi-connected component, use BBC to compute the centrality of each node in the block. The advantage of HBC on the original algorithm is that the computation of BBC is split in smaller problems. The computation time now depends on N': the size of the largest bi-connected component (and not on N).
3) Update the centrality of each cut point, which can be performed in linear time with the number of cut-points, exploiting the fact that computing BC on a tree is trivial.

If a network can be decomposed into blocks of significant size, then $N' << N$ and there is a tangible gain for using

[1]Some authors do not include the endpoints in the computation, without any loss of generality we use a variant that includes also the paths that have one endpoint in n_k.

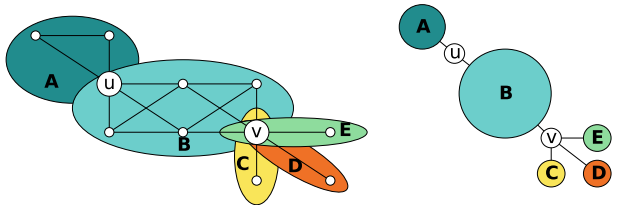


Fig. 1: An example decomposition of a graph into a block-cut-point tree.

HBC. Instead if $N' \simeq N$, HBC may perform worse than BBC, since it adds to BBC the computation time to perform step 1. Note that HBC computes the exact centrality b_k, it is heuristic in the sense that the theoretical worst-case complexity of BBC is not reduced.

C. Contribution

The first question we want to answer is if it is feasible to compute BC on real hardware for networks that scale up to 1000 nodes. This is an arbitrary upper bound we set for a mesh network size, but, to our knowledge, there are no larger networks that use a flat, non-hierarchical approach[2]. We expect performances to be correlated with the results of step 1 of the algorithm, which strongly depends on the underlying network topology, and we want to quantify the gain introduced by HBC. Note in fact that any link-state routing algorithm already implements Dijkstra's algorithm so it is straightforward to implement BBC, while it is non-trivial to implement HBC. We show that on computationally-limited hardware and realistic topologies, we can compute centrality with HBC on larger networks than with BBC. HBC is in certain cases almost two orders of magnitude faster than BBC and thus it is worth the effort of implementing it.

But how fast is *fast*? Consider that using a link-state routing protocol every network node may receive tens of topology updates per second, many of which minimally modify the weight of some links. Nevertheless in principle, network centrality should be re-computed at every change in the network. As an extreme example consider a ring topology: the breakage of one link will re-route half of the shortest paths in the network and radically change the centrality of almost all the nodes. Even if HBC can be computed in a few seconds (and this is the case) embedded devices generally come with only one core and need to perform many other tasks, therefore we need to set a limit to the frequency of the updates. This introduces the second question: what is the best trade-off between update frequency and accuracy in the centrality measure? In practice we expect real networks to be pretty stable and to require centrality computation at most every few minutes. In the rest of the paper we verify this expectation using data observed from one real network.

[2]The AWMN community network in Greece [20], and the Freifunk network [21] are close to that size while the Guifi network is even larger [22], but it is organized as a backbone with stub networks.

	Server Machine	Ubiquiti Nanostation M2
CPU:	Intel E5-2630 2.4GHz	Atheros AR7240 390MHz
RAM:	4 × 16GB 1866 MHz	32MB
OS:	Ubuntu Server 14.04	OpenWRT 15.05

TABLE I: Hardware and software specs.

Graph type	Number of nodes	Edges / Nodes Ratio
ER	[100, 1000] step 100	3
PL	[100, 1000] step 100	2
CN	[100, 1000] step 100	1.2

TABLE II: Synthetic graphs generated for the experiment

III. Evaluation Set-Up

BBC is already present in the Boost C++ Graph Library[3], so we exploited this library to implement HBC. Our implementation is validated and tested on two platforms, a powerful server machine and a low-power wireless router, whose characteristics are summarized in table I. Source code for the HBC implementation is available[4].

As explained in section II-B the performance of HBC depends on the underlying topology that influences the value of N'. Thus, we run BBC and HBC on a set of synthetic topologies of three types: Erdos graphs, Power Law graphs (using the Barabasi-Albert generation algorithm), and Community Network-like graphs (labelled ER, PL, and CN, respectively). The first and the second kinds are widely known in literature, while the third type was inspired by the analysis of real large mesh networks. It uses two different statistical distributions for the core of the network and for the assignment of leaf nodes, and the results match the profile of several portions of the Guifi.net mesh network [23]. For this reason we consider it closer to topologies of in-production networks. All the graphs are generated using the open-source code made available by the *NePA TesT* framework [24], please consult the original paper for more details on the code and the generators.

For each type of topology we generate 300 random graphs, split in 10 groups with increasing size, from 100 to 1000 nodes, table II reports some of the generation parameters. For each graph we measure the minimum running time on ten repetitions, to filter out interference with other processes. We also measure the number of bi-connected components and the value of N'.

Finally, we test the algorithms on three real topologies, taken from a previous data collection [25], [3]. This dataset includes a collection of topology snapshots of three in-production mesh networks (named Ninux, FFW, FFG), with one snapshot every 10 minutes for a week. In the collection phase we analysed the topologies, polished them in order to remove redundant information and worked with the communities to verify the goodness of the results. We extract one snapshot from each network to measure the running time of HBC and BBC.

We use the data set also to understand how often we have to re-compute the centrality. We use 200 snapshots of the FFG

[3]http://www.boost.org/doc/libs/1_60_0/libs/graph/doc/index.html

[4]See the Prince Open Source project available at: https://github.com/gabri94/poprouting/

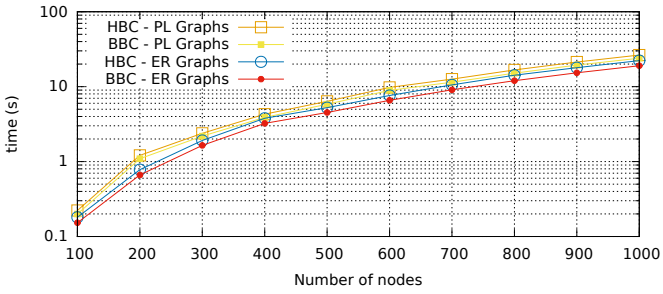


Fig. 2: Computing time using BBC and HBC on the server for synthetic ER and PL topologies. The curves are almost overlapping, with BBC perfoming slightly better than HBC.

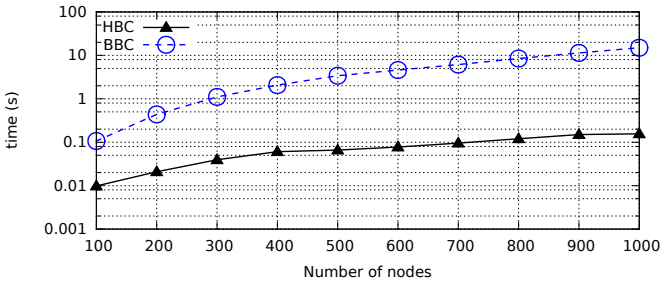


Fig. 3: Computing time with BBC and HBC on the server for synthetic CN topologies.

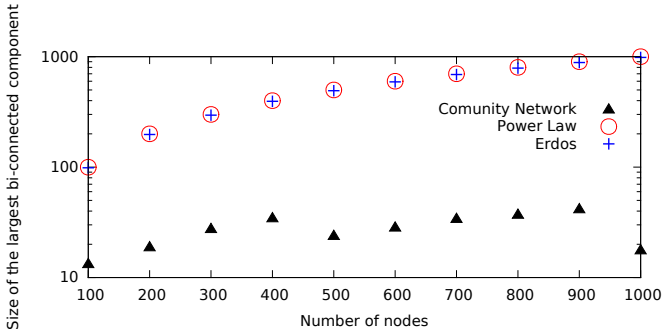


Fig. 4: The average value of N': the number of nodes in the largest bi-connected component for each topology.

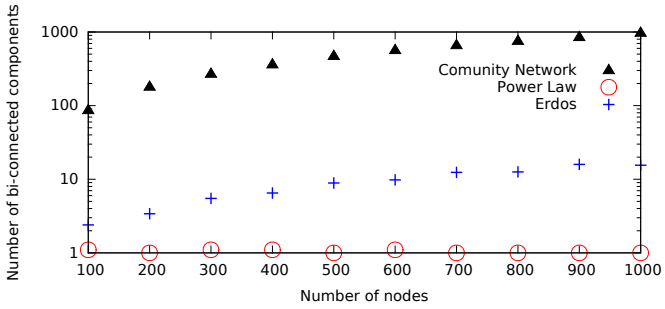


Fig. 5: Average number of bi-connected components in each topology.

network, being it the one that changes more frequently, and we compute two metrics, the first is the average and the standard deviation of the centrality value. The second is the top-k overlapping metric $P_k(i,j)$, defined as follows. For every i-th network snapshot we compute the centrality of each node, and we sort the array $n(i)$ of all the nodes in decreasing order of BC. Given a value k (smaller than the number of nodes in the network) we call $n_k(i)$ the top k elements of $n(i)$. The top-k overlapping metric is defined as

$$P_k(i,j) = \frac{100}{k}||n_k(i) \cap n_k(j)||$$

This metric tells how much the top-k elements of an array have changed from one sample to another, without considering the value of the corresponding centrality. It is a meaningful metric in our scenario because we are mostly interested in the ordering of the centrality array, rather than the single values, and we focus on the top elements that route most of the traffic in the network.

IV. Results

Figure 2 shows the time needed to compute the BC of all the nodes with both BBC and HBC with the server machine for ER and PL synthetic topologies. Figure 3 reports instead the same figure for synthetic CN topologies. The scale is logarithmic for readability. In ER and PL topologies HBC performs slightly worse than BBC, BBC execution time ranges from about 200 ms for 100 nodes to about 20 s for 1000 nodes. In CN topologies instead, HBC can reduce the running time of almost two orders of magnitude, with computing times that range between 20 and 200 ms.

This behavior is explained by figs. 4 and 5, that show, for each group of 10 random graphs, the average N' and the average number of bi-connected components produced by step 1 of HBC. For both PL and ER there is one large component that includes almost all the nodes in the network, so $N' \simeq N$. The speed-up of step 2 is negligible, and the time needed for step 1 makes HBC slightly slower than BBC. CN topologies instead have many bi-connected components and the average size of the largest one is $N' << N$. To explain this we must observe that CN topologies include a number of leaves that is much higher than both ER and PL graphs, which tend to have only a single, huge bi-connected component. The more realistic CN topologies include not only many leaves, but also several mid-size bi-connected components.

Figure 6 reports the computing time on the Ubiquiti router only for the CN graph types, which shows two key facts. The first is that again there is a difference of almost two orders of

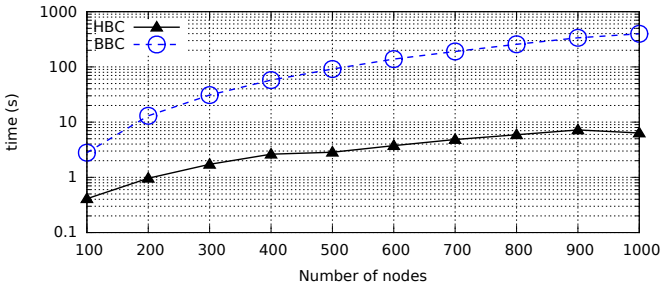


Fig. 6: Computing time with BBC and HBC on the Ubiquiti router for synthetic CN topologies.

Network	BBC	HBC	Nodes
Ninux	4.8	1.1	126
FFW	22.1	7	235
FFG	5.3	0.97	126

TABLE III: Computing time in seconds for HBC and BBC on the Ubiquiti router for three real topologies.

magnitude from BBC to HBC. The second is that in the largest network the absolute running time goes from approximately 397s for BBC down to 6.3s for HBC, which, contrarily to what measured in a powerful server, marks the difference between something impossible and something practical in a real application. As a further confirmation we run BBC and HBC on the router using the topologies of the three real networks. Results are reported in table III. The improvement, albeit smaller than in synthetic graphs is still remarkable. Summing up, this first set of results shows that HBC makes it possible to compute centrality on a real wireless mesh router. However, table III shows that time needed to perform computation does not allow a hard real-time computation of centrality.

It is thus fundamental to evaluate what is a realistic time interval to recompute centrality, based on data from a real network. Figure 7 reports the average value and the standard deviation of BC for each node of FFG that are not leaf nodes. It shows that centrality is not evenly distributed since as few as 20 nodes have centrality higher than 0.1, which makes it reasonable to concentrate security functions only in a subset of nodes.

Thus, fig. 8 shows the top-20 percentage overlap computed between a sample $n(i)$ and its successor $n(i+1)$, based on a set of 200 samples measured once every 10 minutes. Figure 8a shows that the percentage overlap for a 10 minutes interval is always higher than 85% meaning that at most 3 nodes over 20 change from one sample to another. Figure 8b and fig. 8c show the same metric re-sampling the data set at time intervals of 20 and 30 minutes and we can see that the variations are smoothed. This means that not only there is a small number of nodes with high centrality, but also that there is a core of about 17-18 nodes that rarely exit the top-20. We believe that

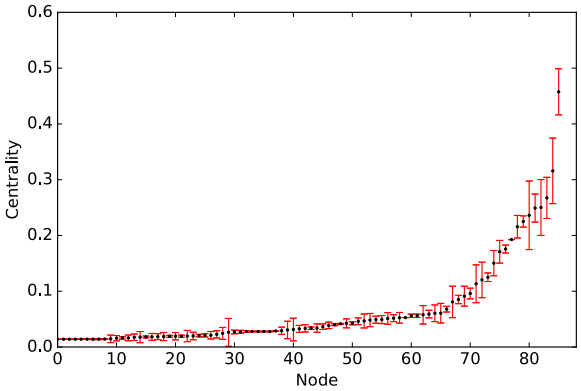
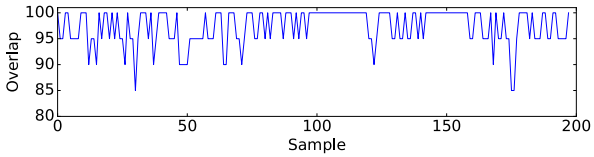
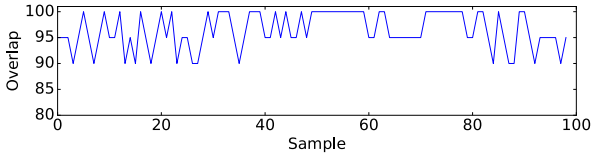


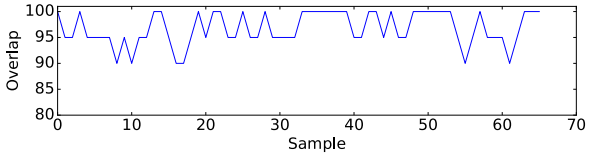
Fig. 7: Average value (black) and standard deviation (red) of the BC for FFG.



(a) One sample every 10 minutes.



(b) One sample every 20 minutes.



(c) One sample every 30 minutes.

Fig. 8: The top-20 percentage overlap for the centrality array in FFG, sampled with different intervals.

recomputing betweenness at intervals of tens of minutes, or even a hour, is a practical trade-off between saving computing resources and the accuracy of security functions.

V. Conclusions

There is a growing interest in centrality metrics for networking and security, but there is little experience on their application. This paper partially fills this void and shows that centrality metrics can be calculated, even with low-power hardware, on topologies up to 1000 nodes. HBC plays a fundamental role in this, because it reduces the computation time to a value that allows soft real-time computation in mesh networks routers. With the original BBC the computation time increases of almost two orders of magnitude and makes the computation impractical. This is true for topologies that resemble real networks, or that were extracted from real networks, while it gives no advantages on random synthetic topologies.

This is per-se an interesting observation: Synthetic topologies based on popular algorithms are often taken as representative of real networks. In this case their use fails to capture some very important consequences on real implementations.

Finally, from the data we analyzed, it emerges that a real, in-production mesh network is a stable network where centrality does not radically change from one minute to another. Thus, re-computing centrality with an interval of tens of minutes is a reasonable choice. In some cases the network conditions may suddenly vary due to a change in the network topology that impacts a large number of shortest paths: when this happens, centrality should be re-computed as soon as possible. An approach that can be suitable for this task is to perform every few seconds a fast pre-processing of the network graph in order to detect macroscopic changes, and thus trigger the re-computation of centrality asynchronously when it is necessary. We leave this activity as a future work for our research.

REFERENCES

[1] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, Mar. 1977.

[2] D. Katsaros, N. Dimokas, and L. Tassiulas, "Social network analysis concepts in the design of wireless ad hoc network protocols," *IEEE Network*, vol. 24, no. 6, pp. 23–29, Dec. 2010.

[3] L. Maccari and R. Lo Cigno, "A week in the life of three large wireless community networks," *Ad Hoc Networks*, vol. 24, Part B, pp. 175–190, 2015.

[4] R. Puzis, P. Zilberman, Y. Elovici, S. Dolev, and U. Brandes, "Heuristics for Speeding Up Betweenness Centrality Computation," in *Proc. of ASE/IEEE Int. Conf. on Social Computing, Int. Conf. on Privacy, Security, Risk and Trust*, 2012.

[5] L. Maccari and R. Lo Cigno, "Waterwall: a cooperative, distributed firewall for wireless mesh networks," *EURASIP Jou. on Wireless Communications and Networking*, vol. 2013, no. 1, pp. 1–12, 2013.

[6] M. Taghizadeh, A. R. Khakpour, A. X. Liu, and S. Biswas, "Collaborative firewalling in wireless networks," in *INFOCOM, 2011 Proceedings IEEE*, April 2011, pp. 46–50.

[7] R. Fantacci, L. Maccari, P. N. Ayuso, and R. M. Gasca, "Efficient packet filtering in wireless ad hoc networks," *IEEE Communications Magazine*, vol. 46, no. 2, pp. 104–110, February 2008.

[8] H. Zhao, J. Lobo, A. Roy, and S. M. Bellovin, "Policy refinement of network services for MANETs," in *Proc. of 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, 2011.

[9] S. Dolev, Y. Elovici, R. Puzis, and P. Zilberman, "Incremental deployment of network monitors based on group betweenness centrality," *Information Processing Letters*, vol. 109, no. 20, pp. 1172–1176, 2009.

[10] S. Dolev, Y. Elovici, and R. Puzis, "Routing betweenness centrality," *Jou. of the ACM*, vol. 57, no. 4, pp. 25:1–25:27, May 2010.

[11] 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. on Modelling and Computer Simulations*, vol. 22, no. 1, pp. 5:1–5:26, Dec. 2011.

[12] P. Zilberman, R. Puzis, and Y. Elovici, "On network footprint of traffic inspection and filtering at global scrubbing centers," *IEEE Transactions on Dependable and Secure Computing*, vol. PP, no. 99, pp. 1–1, 2015.

[13] M. Kas, S. Appala, C. Wang, K. Carley, L. Carley, and O. Tonguz, "What if wireless routers were social? approaching wireless mesh networks from a social networks perspective," *IEEE Wireless Communications*, vol. 19, no. 6, pp. 36–43, 2012.

[14] G. Nomikos, P. Pantazopoulos, M. Karaliopoulos, and I. Stavrakakis, "Comparative assessment of centrality indices and implications on the vulnerability of ISP networks," in *Proc. of the 26th International Teletraffic Congress (ITC)*, 2014, pp. 1–9.

[15] P. Pantazopoulos, M. Karaliopoulos, and I. Stavrakakis, "Distributed placement of autonomic internet services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1702–1712, July 2014.

[16] L. Maccari and R. Lo Cigno, "Pop-Routing: Centrality-based Tuning of Control Messages for Faster Route Convergence," in *Proc. of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM)*, 2016.

[17] L. Baldesi, M. Leonardo, and R. Lo Cigno, "Optimized Cooperative Streaming in Wireless Mesh Networks," in *The 15th IFIP Networking Conference (NETWORKING)*, 2016.

[18] L. Baldesi, L. Maccari, and R. Lo Cigno, "Improving P2P streaming in Wireless Community Networks," *Computer Networks*, vol. 93, Part 2, pp. 389–403, Dec. 2015.

[19] U. Brandes, "A Faster Algorithm for Betweenness Centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[20] F. Elianos, G. Plakia, P. Frangoudis, and G. Polyzos, "Structure and evolution of a large-scale wireless community network," in *Proc. of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks Workshops, (WoWMoM)*, June 2009.

[21] M. Milic and M. Malek, "Analyzing large scale real-world wireless multihop network," *IEEE Communications Letters*, vol. 11, no. 7, pp. 580–582, July 2007.

[22] D. Vega, R. Baig, L. Cerdà-Alabern, E. Medina, R. Meseguer, and L. Navarro, "A technological overview of the guifi. net community network," *Computer Networks*, vol. 93, pp. 260–278, 2015.

[23] L. Cerda-Alabern, "On the topology characterization of Guifi.net," in *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2012.

[24] L. Baldesi and L. Maccari, "NePA TesT: Network Protocol and Application Testing Toolchain for Community Networks," in *Proc. of the 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, 2016.

[25] L. Maccari, "An analysis of the Ninux wireless community network," in *Proc. of the Second International Workshop on Community Networks and Bottom-up-Broadband (CNBuB)*, 2013.