Pop-Routing: Centrality-based Tuning of Control Messages for Faster Route Convergence

***Abstract*—Fast and efficient recovery from node failure, with minimal disruption of routes and the consequent traffic loss is of the utmost importance for any routing protocol. Link state protocols, albeit preferred to distance vector ones because of faster convergence, still suffer from a trade-off between control message overhead and performance. This work formalizes the routes' disruption following a node failure as an optimization problem depending on the nodes' centrality in the topology, constrained to a constant signaling overhead. Next, it shows that the solution can be found using Lagrange Multipliers. The solution complexity is low enough to be computed on-line on the network routers, thus obtaining the optimal setting of control message timers that minimize the traffic loss following a node failure. The gain obtained is quantified in power-law and synthetic topologies, and it is also tested on real network topologies extending the OLSR protocol to use the modified timers, showing that the inevitable approximations introduced in the analysis do not hamper the very good results achievable through this novel approach. The technique can be applied to any link state protocol, including OSPF, and improves route convergence not only upon failures but on every topology modification.**

***Index Terms*—Multi-hop networks; mesh networks; ad-hoc networks; centrality; signalling overhead; failure recovery.**

I. Introduction

Fast recovery after a node (or link) failure is one of the key features of a link-state routing algorithm. Highly critical networks use redundant hardware configurations, hardware mechanisms to quickly detect the failure of a node, and implement proprietary extensions to fast re-route the traffic upon a failure. In many other cases, from small Autonomous System (AS) to fixed and mobile Wireless Mesh Network (WMN), this is not possible, thus the failure recovery still depends on the exchange of layer-3 control messages.

Albeit Dijkstra algorithm ensures fast and efficient minimum weight paths computation, the protocol convergence is still limited by two factors: *i)* failure discovery; and *ii)* propagation of the new topology information. These two functions are implemented in all major proactive links-state routing protocols through different periodic messages: `HELLO` (`H`) messages sent every $t_{\mathtt{H}}$ s, and `Link-State Advertisement` (`LSA`) messages sent every $t_{\mathtt{A}}$ s. `H` messages are sent by every node on every network interface to announce itself and discover neighbors. `LSA` messages are sent by every node to update the routing, i.e., confirm (or change) the topology of the network and they are propagated in the entire network to allow every node to build and maintain the routing table. `LSA` messages can also be sent upon failure discovery to speed up convergence.

Whenever a node fails, all the routes passing through the failed node also fail, creating temporary malfunctions and traffic loss until the surrounding nodes identify the node failure (through missing and unanswered `H` messages) and start recomputing routes and propagate the changed topology through `LSA` messages. Moreover, temporary routing loops can be created before the information is correctly propagated to the whole network [1], leading to further service disruption. Fast convergence requires very small $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$; however, this not only imply a large overhead, but also triggers the risk of oscillations in case of short, temporary failures, and fast modifications of link costs, as possible for instance, in wireless networks with bandwidth- or load-based routing.

There is a clear trade-off between increasing performance (minimizing route disruption and loops after a failure) and keeping $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$ large enough to keep the overhead to a reasonable level and avoid oscillations. Finding such trade-off can be though of as an optimization problem: given a target overhead find node-dependent values of $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$ that maximize the speed of route convergence.

The contribution of this paper is exactly the formulation and solution of this optimization problem. We derive a methodology that enables every node n_i of the network to locally solve it, and to find the value of $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ that, based on the node *centrality* in the topology, maximize the network performance. The solution is exact and finds the optimum; however, the modeling process introduces some inevitable approximations, thus to validate the model and to show the viability of our proposal, we implemented it in the Optimized Link State Routing (OLSR) daemon to test its performance on real topologies taken from working large scale WMNs.

The name Pop-Routing (abbreviated PopR) comes from a similarity with equalization presets that can be found on media players: they increase the loudness of central frequencies and decreases the loudness of extreme frequencies when listening to pop music. Since we increase the amount of information generated by central nodes and decrease it for peripheral nodes, we call our approach Pop-Routing.

II. Related Works

Most of the works concerned with our problem have studied wireless networks, where the problem is more important and overhead more critical, thus we start with these works, and specially with those concerning OLSR. Initially the values of $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$ have been studied to understand how they influence the delivery of packets [2], [3]. In [2] the authors introduce a measure called Route Change Metric that quantifies the impact

of the H timer in terms of routes' reliability. The results confirm the intuition that the timers strongly influence the routes convergence speed with after a modification of the topology. It also shows that the effect of tuning $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$ strongly depends on the network characteristics. An improvement has been to pre-calculate optimal values for the timers, which has been investigated in a series of works. la latest is [4] using optimization techniques and meta-heuristics. This approach assumes that there is an optimal tuning of parameters for a large family of networks. Instead, we dynamically adjust the parameters based on the properties of each node.

A few works try to autonomically tune the timers in mobile networks. A network cartography approach is used in [5], requiring the knowledge of the position of the nodes, while Network parameters are changed as a function of the network size in [6]. Protocol parameters have been studied for their obvious impact on the convergence times of routes and energy consumption in heterogeneous networks [7]. None of these works use centrality metrics or apply an approach similar to PopR.

The extreme case of timers tuning is setting them to ∞ for some nodes, building a *virtual backbone*: only a subset of nodes generates LSA messages. There is a very rich literature on virtual backbones, with two well studied approaches: Connected Dominating Sets (CDS) and Multi-Point Relays (MPR) (see [8] for a recent review on CDS, and [9] for a survey on MPRs) and recent works [10]–[12] exploring MPR and CDS nodes selection.

PopR does not define numerable categories of nodes, but increases or decreases $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$ with a continuous computed locally by each node, does not need negotiations (as MPRs and CDNs elections) and changes naturally with the evolution of the network. Moreover, PopR is perfectly compatible with any other approach as long as the routing protocol allows differentiated timers: indeed, PopR can also be applied together with CDS, MPRs, or in general clustering techniques.

Fisheye routing [13] is a smart technique to reduce overhead, but it does not directly compare with PopR. LSA messages are sent with a constant timer, but with a variable time-to-live (TTL) field. TTL governs the number of hops that the LSA message will go through, so that nodes that are close to LSA origin receives frequent updates, while nodes that are far away receives sporadic updates. Fisheye has a serious drawback: whenever two nodes in the network have a different view of the topology, they might take contrasting decisions and introduce routing loops [14]. PopR does not suffer from this problem.

Convergence speed of link-state routing protocols is important also in wired networks. Route convergence for Open Shortest Path First (OSPF) has been largely studied, since the $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$ cannot be arbitrarily decreased. A survey on the issues related to convergence of OSPF and the proposed workarounds can be found in [15]. Among these we mention IP fast re-routing or incremental update of link weights [1], [16]. These techniques still rely on failure detection and can be coupled with PopR.

A. Centrality in Networks

A centrality metric estimates how much a node is in the *core* or in the *periphery* of the network, with a meaning that is highly dependent on the context of the analysis. Centrality has been used in social science since the '70s to identify the most influential elements in social networks [17], but was not applied to communication networks until recently [18]. Centrality can be used to enhance network monitoring [19], intrusion detection and firewalling [20], [21], resources allocation [22] and topology control [23].

PopR exploits betweenness centrality: the fraction of all shortest paths routed by a node. Given a graph with N nodes and E links, the computation of the shortest path rooted at a node with Dijkstra's algorithm scales as $\mathcal{O}(E + Nlog(N))$. Betweenness computation with a straightforward application of Dijkstra's algorithm scales as $(O)(N^3)$. Other approaches take advantage of the sparsity of real networks, that have a small number of links compared to a full mesh, and achieve exact computation of centrality in $\mathcal{O}(EN + N^2log(N))$ [24]. According to this work, betweenness centrality for all the nodes in a network with 500 nodes can be computed in few seconds on commodity hardware. [24]. There also exists extremely efficient approximations and heuristics particularly efficient in real networks [25], [26], that make it possible to compute contrality on-line with thousands of nodes.

III. Formulation of the problem

Consider a network graph $G(\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ is the set of vertexes and $\mathcal{E}$ is the set of edges with $||\mathcal{N}|| = N$ and $||\mathcal{E}|| = E$. Tab. I reports the main notation and symbols we use in the math analysis of the problem [a]. The graph represents a multi-hop network, where each vertex corresponds to a node and

[a]From now on we will use the calligraphic style to refer to sets, as in $\mathcal{N}$ and the bold style to refer to arrays, as in **B** and we refer to the size of a set with $||\cdot||$

Symbol	Description
n_i	node i
$\mathcal{N}$, $\mathcal{E}$	set of edges and vertexes of the graph
N, E	size of $\mathcal{E}$ and $\mathcal{N}$
$t_{\mathtt{H}}(i), t_{\mathtt{A}}(i), t_{\mathtt{H}}, t_{\mathtt{A}}$	timers for H and LSA messages and default values
$V_{\mathtt{H}}$, $V_{\mathtt{A}}$	threshold of lost H and LSA messages to detect failures
R	number of messages for network-wide flooding
b_i, **B**	betweenness of n_i, array of all b_i
$L(k)$, $L_{\mathtt{H}}$, $L_{\mathtt{LSA}}$	theoretical loss due to: n_k failure, detection, propagation
$\tilde{L}_A$, $\tilde{L}_R$, $\tilde{L}_g$	absolute, relative and global empirical loss reduction
$O_{\mathtt{H}}$, $O_{\mathtt{LSA}}$	total overhead when $t_{\mathtt{H}}(i) = t_{\mathtt{H}}$, $t_{\mathtt{A}}(i) = t_{\mathtt{A}}$ resp.
$p_{i,j,=}\{n_i \dots n_j\}$	sequence of nodes in a shortest path $n_i \rightarrow n_j$
T_d, T_p	failure detection and propagation time

Table I
MAIN SYMBOLS USED IN THE PAPER

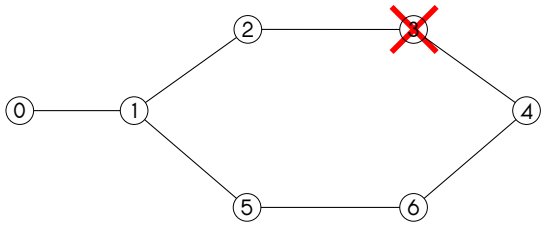

Figure 1. Sample topology used to exemplify PopR.

each edge corresponds to a link. We do not distinguish between the terms vertex/node and edge/link, we call the generic node n_i. Links correspond to logical interfaces at the IP level, thus in wireless networks two or more links may share the same physical resources (radio interface).

When we refer to *1-hop broadcast*, we mean that the node sends the packet to the IP broadcast address on every logical interface TTL set to 1, so the packet is not re-broadcast by the neighbours. For simplicity, each edge has weight 1, so no quality metric is used to build the routing tables. Results can be directly applied to link-quality routing.

Refer to the sample network in Fig. 1. Suppose the routing table of n_1 uses n_2 as a next-hop to reach n_4, so the shortest path from n_0 to n_4 will be $p_{0,4} = \{n_0, n_1, n_2, n_3, n_4\}$. If n_3 fails, before the network reconfigures itself every route that includes n_3 will fail too. The position of n_3 in the network is important to understand how critical its failure for the network is. It is intuitive to understand that the failure of n_3 impacts a number of routes, while the failure of n_0 impacts only traffic to/from n_0 itself. Formalizing this difference and embed it in the logic to define differentiated timers $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ for every node n_i is the core of our proposal.

PopR can be applied to a variety of link-state protocols, for this reason we do not target a specific one but we describe a generic protocol model that includes features of many link-state routing protocols. Since our interest is primarily directed to large wireless mesh networks we will then implement PopR on OLSR, and we will use default values from the OLSR RFC.

A. Link-state protocol model

Let's consider a generic proactive, link-state routing protocol. Every node n_i sends H messages every time interval $t_{\mathtt{H}}(i)$. H messages use 1-hop broadcast and discover and maintain n_i neighbors. Each H message contains a *validity* field. A neighbor n_j on n_i sets a timer to the validity time at the reception of any H from n_i, if a new H is not received before its expiration, n_j considers link $\{n_i, n_j\}$ broken. The validity is generally set to a multiple of $t_{\mathtt{H}}(i)$, so that validity is defined as $V_{\mathtt{H}} t_{\mathtt{H}}(i)$ with $V_{\mathtt{H}}$ an appropriate constant. Every node n_i also sends LSA messages every time interval $t_{\mathtt{A}}(i)$ (generally with $t_{\mathtt{A}}(i) > t_{\mathtt{H}}(i)$). An LSA contains all the valid $\{n_i, n_j\}$ links. LSA messages are flooded and reach every node in the network so every node n_k is aware of the full topology and can compute the shortest path to any destination and build its routing table. Similarly to what happens with H messages, LSA messages include a validity timer so that when n_j fails to receive a new LSA from n_i before the expiration of the validity timer, n_j will recompute its routing table removing the links that were included in the expired LSA message. Again, we express validity as $V_{\mathtt{A}} t_{\mathtt{A}}(i)$.

We also introduce two simplifying assumptions, that do not influence the results but ease the theoretical analysis. Link-state protocols have a protocol-internal logic that ensures that every LSA is received by all the nodes passing through a minimal number of links (we call this number R). Our conclusions are independent on the minimization strategy used, the only assumption we do is that R does not depend on the source of the LSA, which is perfectly plausible. The second assumption is that $t_{\mathtt{A}}(i)$ dominates both propagation delays and transmission delays. Again, a legitimate assumption with almost any modern network. If intermediate nodes add a non-negligible delay before retransmitting an LSA packet to perform message aggregation, our theory is still valid if the total information diffusion time is still dominated by $t_{\mathtt{A}}(i)$. To validate this assumption we test PopR on the OLSRd routing daemon, which introduces an aggregation delay, and we show that also in this case PopR achieves a loss reduction.

Finally, it is worth noticing that failure detection via H must happen before information propagation via LSA, so that it is reasonable to optimize separately $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$. Else, the optimization could yield mathematically valid but non-realistic values for the two timers (such as $t_{\mathtt{H}}(i) >> t_{\mathtt{A}}(i)$). Albeit separate optimization does not guarantee that the optimized timers lies in an acceptable region from the protocol point of view, all the test we did yielded results we within acceptable boundaries.

IV. Failure detection as an optimization problem

Referring again to Fig. 1, after n_3 fails at time T_0, nodes n_2 and n_4 will sense the event after the timer set to $V_{\mathtt{H}} t_{\mathtt{H}}(i)$ expires and recompute their routing tables to use an alternative path. Considering the worst case scenario in which n_3 fails right after generating the H, the detection time is $T_d = T_0 + V_{\mathtt{H}} t_{\mathtt{H}}(3)$.

Given all the shortest paths $p_{i,j} = \{n_i, \ldots, n_j\}$ in the network, we call b_k the *shortest path betweenness* of n_k defined as:

$$b_k = \frac{1}{N(N-1)} \sum_{i,j \in N;\, i \neq j} \frac{||\{p_{i,j} | n_k \in \{n_i, \ldots, n_j\}\}||}{||\{p_{i,j}\}||} \quad (1)$$

this is a generic graph-based definition that is often used in the literature. When the graph represents an IP network, at each instant there is only one shortest path from n_i to n_j so that $||\{p_{i,j}\}|| = 1$. In a directed connected graph without self loops the sum in Eq. (1) ranges from a minimum of $2(N-1)$ paths that start or terminate at n_k, to a maximum corresponding to the total number of paths given by $N(N-1)$ implying $b_k \in [\frac{2}{N-1}, 1]$[b].

[b]Typically b_k does not include the endpoints in the computation so $b_k \in [0, 1]$; we instead use a variant that includes also the paths that have one endpoint in n_k, so that b_k is never 0 and singularities are avoided when b_k is at denominator of a fraction.

We define the potential loss due to the failure of n_k as:

$$L(k) = V_{\mathtt{H}} t_{\mathtt{H}}(k)\, N(N-1) b_k \tag{2}$$

$L(k)$ is the number of broken paths due to the failure of n_k multiplied by the time these paths will stay broken.

If we assume that the traffic matrix is uniform, then $L(k)$ also estimates the total amount of traffic lost due to the failure of n_k. In case we have precise information on the amount of traffic per link (which is plausible if such information is conveyed in LSA messages) then the definition of b_k can be modified to use a weighted graph, where each link is weighted by the carried traffic so that b_k measures the importance of n_k as a function of the traffic it routes. This can be particularly useful when the network is connected to a gateway node, which may be topologically peripheral, but may be routing a large amount of traffic.

Finally, the average loss due to the failure of any node in the network is given by:

$$L = \frac{1}{N} \sum_{k=1}^{N} L(k) = V_{\mathtt{H}}(N-1) \sum_{k=1}^{N} t_{\mathtt{H}}(k) b_k \tag{3}$$

Sec. IV formalizes something that is intuitively easy to understand. Since the time needed to reconstruct a broken route is linear with the interval between each H, the average packet loss due to the breakage of a route grows with the values $t_{\mathtt{H}}(k)$. Moreover, the failure of nodes with high centrality (that are traversed by many routes) generates a higher loss compared to the failure of peripheral nodes.

To have a faster route convergence upon a failure we can thus decrease $t_{\mathtt{H}}(k)$, which in turn will increase the overhead due to control traffic. The overhead generated by node n_i with H messages is given by the number of H messages per second per link, multiplied by the size of the H messages. Our strategy does not modify the size of H and LSA messages, so from now on we refer to the number of control messages when using the term overhead.

Each H is sent on all the links exiting n_i, so the number of H messages per second is simply:

$$O_i = \frac{d_i}{t_{\mathtt{H}}(i)} \tag{4}$$

Thus, the total overhead generated per second on the network is given by:

$$O = \sum_{i=1}^{N} \frac{d_i}{t_{\mathtt{H}}(i)} \tag{5}$$

Setting $t_{\mathtt{H}}(i) = t_{\mathtt{H}}$ for all nodes, we obtain the overhead of the unmodified protocol: $O_H = \sum_{i=1}^{N} \frac{d_i}{t_{\mathtt{H}}}$.

We can now formalize the problem of failure detection as an optimization problem defined by Eq. (5) and Sec. IV. Since the optimization is not influenced by the constants, we can safely remove them:

$$\text{minimize} \quad L_{\mathtt{H}} = \sum_{i=1}^{N} t_{\mathtt{H}}(i) b_i \tag{6}$$

$$\text{subject to} \quad O_{\mathtt{H}} = \sum_{i=1}^{N} \frac{d_i}{t_{\mathtt{H}}(i)} \tag{7}$$

Eq. (6) minimizes the loss in the network, while Eq. (7) sets the total overhead to be constant. The solution technique we use ensures that all $t_{\mathtt{H}}(i)$ have the same sign, so it is easy to select all $t_{\mathtt{H}}(i)$ positive.

V. Information Propagation: Optimizing $t_{\mathtt{A}}(i)$

Every node n_i sends LSA messages every $t_{\mathtt{A}}(i)$, and each LSA is forwarded R times in the network for flooding. The overhead due to LSA messages is:

$$O = \sum_{i=1}^{N} \frac{R}{t_{\mathtt{A}}(i)} \tag{8}$$

and $O_{\mathtt{LSA}} = \sum_{i=1}^{N} \frac{R}{t_{\mathtt{A}}}$ is the total overhead in a network configured to have $t_{\mathtt{A}}(i) = t_{\mathtt{A}}$.

Te estimate the route disruption caused by delay in LSA messages we need more insight into the protocol. Refer again to the failure of n_3 in the network in Fig. 1. After the detection time T_d, n_2 knows that link $\{n_2, n_3\}$ is not active anymore; it computes a new path to reach n_4, which is given by $p_{2,4} = \{n_2, n_1, n_5, n_6, n_4\}$. n_1, instead, still doesn't know of the breakage, so a temporary loop is created between n_1 and n_2, which is typical of link-state protocols. The loop will be solved at time T_p when n_1 detects the change in the topology, which can happen in two different ways: *i)* after the timer $V_{\mathtt{A}} t_{\mathtt{A}}(3)$ expires, so that n_1 assumes n_3 is dead, removes n_3 and its outgoing links from the network graph, and recomputes the correct path $p_{2,4} = \{n_1, n_5, n_6, n_4\}$; or *ii)* n_1 receives a LSA from n_2 (or n_4) which does not include n_3, so that n_1 knows that link $\{n_2, n_3\}$ (or link $\{n_4, n_3\}$) does not exist anymore, do it will recompute its routing table excluding n_3.

Since loops may be created also farther away from the failure, and they may also include more than two nodes, the complete description of $T_p - T_d$ is a complex stochastic problem, which is outside the scope of this paper. To solve the optimization problem, we refer to the worst case in the the first detection case above, i.e., $T_p - T_d = V_{\mathtt{A}} t_{\mathtt{A}}(3)$. Note however, that also in the second discovery way $T_p - T_d$ does not change significantly. The centrality b_k of n_k depends on the position in the network of n_k and betweenness is correlated: neighbors of a node with high betweenness probably also have a high betweenness and viceversa, so and the approximation above remains reasonable. Consider also that a node failure is not the only case that lead to route breakage and topology modification, and in wireless networks it is surely not the the most usual one. If n_k does not fail, but for some reason the quality of its links decreases substantially (for instance the node is subject to temporary shading), the effect is similar to

a node failure: n_3 is removed from many, and sometimes all, the shortest paths.

With the analysis above, the total average potential loss due LSA messages due to a node failure is proportional to

$$L_{\mathtt{LSA}} = \sum_{i=1}^{N} t_{\mathtt{A}}(i) b_i \tag{9}$$

having removed any constant that do not enter in the optimization procedure. The minimization of Eq. (9) subject to the constraint expressed by Eq. (8) is structurally the same optimization problem formulated by Eq. (6) and Eq. (7), so the same kind of solution can be applied to both problems.

Finally note that a loop is not deterministically created when a node fails, since its neighbors may recompute an alternative route which does not create a loop, so again Eq. (9) is a worst case, and the network performance after a failure may be better than this. It must be considered, though, that a loop not only breaks some routes, it generates a flood of packets in the interested link which makes it (almost) unusable for other routes. In some cases loops may persist for tens of seconds, bringing havoc to the entire network. This justifies to use the worst case scenario to tune $t_{\mathtt{A}}(i)$, with the reasonable assumption that the relative gain obtained is maintained in all cases.

VI. Optimization Solution

One approach to solve some combinatorial problems is the use of Lagrange multipliers. Lagrange multipliers can be used to find the necessary conditions for the extremes of a function subject to a constraint and can be used to solve our optimization problem. If $\mathbf{x} = [x_1, \ldots, x_m]$ is an array of variables, $f(\mathbf{x})$ is the function to be minimized/maximised, and the constraint can be expressed with a function g such as $g(\mathbf{x}) = 0$, then *critical points* are defined as the points that satisfy $\nabla f(\mathbf{x}) = -\lambda \nabla g(\mathbf{x})$ for some constant value λ. Critical points include all the maximum and minimum values of the constrained system.

Consider first H message timers, in this case $\mathbf{x} = [t_{\mathtt{H}}(1), t_{\mathtt{H}}(2), \ldots, t_{\mathtt{H}}(N)]$, $f(\mathbf{x})$ is given by Eq. (6), and $g(\mathbf{x})$ by Eq. (7), so that the gradients are:

$$\nabla f(\mathbf{x}) = [b_1, \ldots, b_N] \tag{10}$$

$$\nabla g(\mathbf{x}) = [-\frac{1}{t_{\mathtt{H}}(1)^2} d_1, \ldots, -\frac{1}{t_{\mathtt{H}}(N)^2} d_N] \tag{11}$$

The Lagrange Multiplier generates the set of N independent equations given by:

$$b_i = \frac{\lambda}{t_{\mathtt{H}}(i)^2} d_i; \quad i = 1, \ldots, N$$

and, given we are interested only in positive values of $t_{\mathtt{H}}(i)$:

$$t_{\mathtt{H}}(i) = \sqrt{\frac{d_i \lambda}{b_i}} \tag{12}$$

enforcing the constraint (7) yields:

$$\lambda = \Big(\frac{1}{O_{\mathtt{H}}} \sum_{i=1}^{N} \sqrt{b_i d_i}\Big)^2 \tag{13}$$

Substituting λ into Eq. (12), we finally obtain $t_{\mathtt{H}}(i)$ as a function of b_i and d_i only:

$$t_{\mathtt{H}}(i) = \frac{\sqrt{d_i}}{\sqrt{b_i}} \frac{1}{O_{\mathtt{H}}} \sum_{j=1}^{N} \sqrt{b_j d_j} \tag{14}$$

Given the optimal set of timers $t_{\mathtt{H}}(i)$, we can use Eq. (6) to compute average performance loss, i.e., the expectation of the product of the number of disrupted routes times the disruption duration if nodes failure probability is uniform:

$$L_{\mathtt{H}} = \sqrt{\lambda} \sum_{i=1}^{N} b_i \sqrt{\frac{d_i}{b_i}} = \frac{1}{O_{\mathtt{H}}} \sum_{i=1}^{N} \sqrt{b_i d_i} \sum_{i=1}^{N} \sqrt{d_i b_i} = \frac{1}{O_{\mathtt{H}}} \Big(\sum_{i=1}^{N} \sqrt{b_i d_i} \Big)^2 \tag{15}$$

Eq. (14) states that a if n_i knows the betweenness and degree of the other nodes, it can easily compute the optimal value for $t_{\mathtt{H}}(i)$. If all the nodes in the network apply the same formula, then the expectation of the traffic disruption upon a node failure is minimize maintaining the overhead constant and equals $O_{\mathtt{H}}$.

With the same procedure we can derive the optimal solution for Eq. (9) subject to Eq. (8):

$$t_{\mathtt{A}}(i) = \frac{\sqrt{R}}{\sqrt{b_i}} \frac{\sum_{j=1}^{N} \sqrt{b_j R}}{O_{\mathtt{H}}} \tag{16}$$

$$L_{\mathtt{LSA}} = \frac{1}{O_{\mathtt{H}}} \Big(\sum_{i=1}^{N} \sqrt{b_i R} \Big)^2 \tag{17}$$

A. Interpretation of the solution

Eq. (14) and Eq. (17) give a fundamental insight: once the network topology is known to every node, which is an intrinsic property of link-state protocols, each node has enough information to compute the optimal values for $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ in order to minimize the routes' disruption due to node failures while keeping the total overhead constant.

This improvement is perfectly compatible with any protocol that support a differentiated timer for each node, and it can be used on top of any topology reduction strategy, like MPRs or Connected Dominated Sets (CDS) [**?**]. Indeed, our approach supersedes those strategies. In fact, the basic idea of topology reduction is to apply a binary label to each node that enables or disables the generation of LSA messages depending on some properties that are locally computed (for instance the betweenness computed on the 2-hop neighborhood, for MPRs in OLSR). Our approach, instead, uses a continuous function to fine tune every timer, with two advantages: first and foremost, PopR reaches optimality, second, PopR does not need any negotiation to select MPR or CDS nodes. Thus, there are no transitory phases in which the state of the network is logically disconnected. This happens instead any time a CDS node, a cluster head, or an MPR fails and the neighbors have to agree to select a new one.

So far we took as an example of topology modification the failure of a node, but routing protocols for wireless networks

use link-quality estimation which make topological changes more frequent. The loss of H messages increase the link cost and may trigger route recomputation even if the link does not effectively break. Indeed, those dynamics still depend on the H and LSA mechanisms, therefore PopR will be beneficial also in those cases (as our emulations on OLSRd show in Sec. VIII).

Even in wired networks using OSPF the optimization of the H timers is an open problem, with some differences compared to the wireless domain. First, the requirements of a network with multi-gigabit point-to-point links carrying real-time traffic are very high. Detection and recovery of broken links should be in the order of the tens or hundreds of milliseconds. On the other hand the available bandwidth per link is generally much higher, so the waste of resources due to control messages is marginal compared to a wireless shared medium. Therefore, one may think that it is safe to arbitrarily reduce $t_{\mathtt{H}}(i)$. A key difference is that a node using OSPF sends LSA messages periodically with a timer set to tens of minutes (in order to remove stale entries) but also generates LSA messages asynchronously when its neighborhood changes. Since temporary link congestion produces correlated packet loss if $t_{\mathtt{H}}(i)$ is very small a short congestion can cause the loss of $V_{\mathtt{H}}$messages and trigger the generation of a LSA message, and thus, a global reconfiguration of the routing tables. When the congestion terminates the topology will change again, and this will produce random routes' fluctuations. It is thus of paramount importance to reduce such risk in nodes that are not critical for the network topology and concentrate it only on those that are central, as PopR does.

Finally, note that betweenness is a feature that slowly changes with modifications in the network. For the centrality of nodes to change significantly, a large number of shortest paths need to be redirected. This happens when a new node that dramatically reduces the average shortest path is added to the network, and this is a rare event. We believe **B** can be recomputed in a static mesh network with a period in the order of tens of minutes. Even if **B** is a little stale and approximate the true betweenness of nodes, PopR does not incur in service disruption, just in a slightly sub-optimal generation of control messages.

VII. Evaluation Set-up

Sec. VIII presents two distinct set of results. The first one is obtained applying directly the optimization derived in Sec. IV and Sec. V on syntetic graphs with controlled properties; the second one is obtained modifying state-of-the art OLSRd code and running it on real topologies in an emulated environment.

The first result set is the evaluation of the two loss formulas given by equations Eq. (15) and Eq. (17). Given a network graph $G(\mathcal{N},\mathcal{E})$ we set $R = E$, $t_{\mathtt{H}} = 2\,\mathrm{s}$, $t_{\mathtt{A}} = 5\,\mathrm{s}$ and we compute the optimal values of $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$.

Let $L_{\mathtt{H}}$ and $L_{\mathtt{LSA}}$ be the value of performance loss (routes' disruption) obtained by the standard version of the protocol, i.e., with all timers equal to $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$, and $L^{\star}_{\mathtt{H}}$ and $L^{\star}_{\mathtt{LSA}}$ the optimal values computed with the optimal values of $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$. The absolute value of the performance loss is however highly influenced by the topology, and also by the many constant that do not influence the optimal operation point. For this reason we use relative metrics of performance defined as

$$L^{R}_{\mathtt{H}} = 1 - \frac{L_{\mathtt{H}}}{L^{\star}_{\mathtt{H}}}; \quad L^{R}_{\mathtt{LSA}} = 1 - \frac{L_{\mathtt{LSA}}}{L^{\star}_{\mathtt{LSA}}}$$

We use topologies generated following two popular models: *i)* The Barabási-Albert (BA) preferential attachment algorithm, that generates graphs with a power-law degree distribution that are very common in real networks; and *ii)* the model developed by Milic and Malek (MM) in [27]. This is a mixed geometrical-statistical model that has been created from the observation of large existing German wireless mesh networks. To further confirm the results, we also test the performance reduction on the topology of three real networks: the wireless community network of Wien (FunkFeuer Wien, abbreviated FFWien), the community network of Graz (FFGraz) and the community network of Rome (the Ninux network). These are three large networks made of 227, 143, and 126 nodes respectively that can be considered in-production networks daily used by hundreds of people[c].

The second result set is thus produced using the Mininet network emulator[d]. Mininet allows the emulation of entire networks with custom topologies, and it is the perfect instrument to experiment with real implementations of routing daemons in large topologies made of hundreds of nodes that can not be recreated in a lab. This second set of results validates the model of link-state protocols we used in the optimization answering three key issues. *i)* How much the approximations we did in the theoretical formulation affect the results. *ii)* What is the effect on PopR of heuristic "improvements" used by real protocols, such as link-quality metrics and message aggregation, which we can not capture in our theory. *iii)* What is the global effect due to the application of PopR to both H and LSA messages, since it is clear that the loss reduction due to the two effects can not be just summed, but it blends in ways difficult to predict.

We tested PopR on the OLSRd daemon, and evaluated how fast the network reacts to the failure of a node. Typically, such evaluation in real scenarios is done measuring lost packet at the application layer on a subset of the nodes in the network. Since we control all the nodes in the emulation we can instead use a more comprehensive metric, computed as follows:

1) Run OLSRd on every emulated node in mininet with a given topology G. At steady state each instance of OLSRd has a routing table with valid next-hops to any destination. The routing table for n_i at time t is stored as a dictionary $R^i_t[\cdot]$ that associates a destination node n_k to the next hop n_j, so that $R^i_t[n_k] = n_j$;
2) Each $R^i_t[\cdot]$ is saved by every node once every 300 ms together with the associated timestamp;
3) At time T_0 node n_i is forced to fail;
4) At time T_e, larger than the expected T_p for all nodes, when all $R^i_t[\cdot]$ are stabilized the emulation is stopped;

[c]http://www.funkfeuer.at/; http://ninux.org/
[d]http://mininet.org/

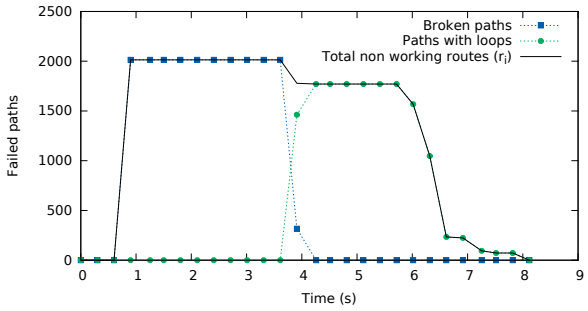

Figure 2. Values of r_h for a sample run in the Ninux topology. The three curves represent the number of broken paths (paths that pass through a failed node) the paths in which a loop is created and the sum of the values.

5) Post-processing all $R_t^i[\cdot]$ yields the exact estimation of the expected routes' disruption due to n_i failure;
6) Iteration over all nodes n_i measures the overall performance loss.

Once an emulation is over, the processing the dictionaries $R_t^i[\cdot]$ at point 5) above is as follows.

A) For each timestamp h navigate the graph from every source n_j to every destination n_k recursively, using the routing tables of intermediate nodes. For each h count the broken routes r_h (i.e., those that still include the failed node or that contain loops). This produces an array $\{(T_0, r_0) \dots (T_h, r_h) \dots (T_e, r_e)\}$ where each position associates an instant after the node failure to number of failures.
B) Define the *combined empirical loss reduction* ($\tilde{L}$) as the integral of the step function stored in the array

$$\tilde{L} = \sum_{h=1}^{e} r_h * (T_h - T_{h-1})$$

$\tilde{L}$ gives the exact measure of the number of broken routes multiplied by the time they remain broken, which is an effective measure for routing protocol convergence and its performance loss due to a failure. $\tilde{L}$ combines the effect of the optimization of both $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ and is the empirical equivalent of the theoretical loss we computed on synthetic graphs.

Fig. 2 reports a sample run emulating the failure of a node in the Ninux network. The curves shows the number of broken paths due both to the detection phase (broken routes) and to the propagation phase (routes in loop). $\tilde{L}$ is the area subtended by the envelope of the broken and looped paths.

We repeat each scenario with standard OLSR and with PopR optimization, obtaining two values of $\tilde{L}$: $\tilde{L}_{olsr}$ and $\tilde{L}_{pop}$ respectively. The absolute performance loss reduction is $\tilde{L}_A = \tilde{L}_{olsr} - \tilde{L}_{pop}$, and the normalized one is $\tilde{L}_R = 1 - \frac{\tilde{L}_{pop}}{\tilde{L}_{olsr}}$.

For each graph G we perform N_f emulations, in each one a different node fails. Clearly, it is interesting to study the effect of the failure of nodes with high centrality, since their failure produces a large reconfiguration and a high stress for the routing protocol (conversely, if a leaf node fails, only the traffic that was directed to that node will be affected, and the application generating that traffic will probably detect the failure way before the routing protocol does, so route convergence to leaf nodes is not really interesting). N_f is the number of nodes for which $b_k > \frac{2}{n-2}$, so their failure also interests traffic that is not directed or generated by them. If a node failure partitions the network, we compute the convergence only on the largest component of the network that remains connected.

Once all the emulations have been run we need another metric that gives a measure of the average impact of PopR on the topology, we thus define the *global loss reduction*:

$$\tilde{L}_g = 1 - \frac{\sum_{i=1}^{N_f} \tilde{L}_{pop}(i)}{\sum_{i=1}^{N_f} \tilde{L}_{olsr}(i)}$$

where i is the index of the failed node. $\tilde{L}_g$ is the average routes' failure reduction due to the failure of any node in the network that potentially carries traffic generated by other nodes.

Summing up, we compute four metrics that, albeit increasing the complexity of the analysis, are needed to have an exhaustive evaluation of the theoretical and empirical performances of PopR:

$L^R_{\mathtt{H}}$, $L^R_{\mathtt{LSA}}$: relative theoretical loss reduction computed on a network graph, due to PopR applied to `H` and `LSA` messages respectively in the abstract model of a link-state protocol;
$\tilde{L}_A$: absolute empirical loss reduction computed emulating the failure of a generic node n_i;
$\tilde{L}_R$: relative empirical loss reduction computed emulating the failure of a generic node n_i;
$\tilde{L}_g$: overall relative empirical loss reduction evaluated on all meaningful nodes' failures.

VIII. Results

Fig. 3 reports the results of the computation of the loss reduction $L^R_{\mathtt{H}}$ and $L^R_{\mathtt{LSA}}$ on the two types of synthetic topologies with an increasing number of nodes in the network. The performance of PopR increases as networks become larger, and results for $L^R_{\mathtt{LSA}}$ are those yielding the most advantage, with routes' disruptions that are reduced by 25% in case of MM networks. Recall that PopR does not increase the overall number (and size) of control messages, so this gain is, in some sense, "for free". The difference between BA and MM networks can be explained by the absence/presence of leaf nodes. A BA graph has no leaf nodes by construction, while MM graphs do have leaf nodes. Since leaf nodes have the minimal betweenness, **B** is less skewed if there are no leaf nodes and there is less room for optimization. As a clarifying example, consider a ring network, each node has the same centrality value and PopR will produce $t_{\mathtt{H}}(i) = t_{\mathtt{H}}$. Real networks do have leaf nodes, so we expect that PopR in real networks will behave as well as in MM networks. For the same reason, the loss reduction $L^R_{\mathtt{H}}$ is practically negligible with BA networks while it oscillates between 5% and 10% in MM networks.

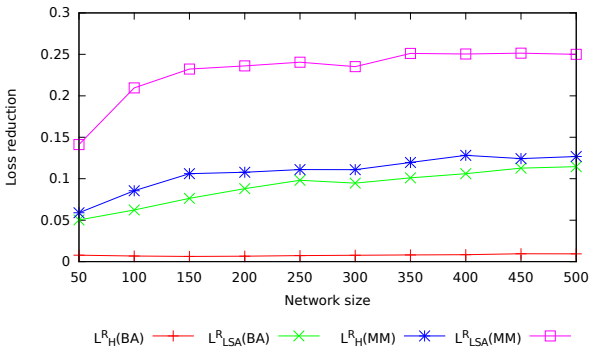

Figure 3. The theoretical loss reduction values due to PopR computed on Milic-Malek and Barabási-Albert graphs.

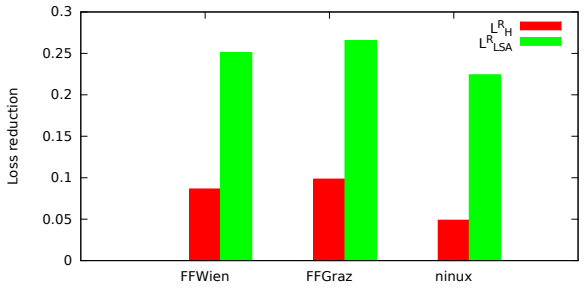

Figure 4. Loss reduction computed on the network topology of three in-production mesh networks made of 227, 143 and 126 nodes.

Fig. 3 shows that the improvement given by PopR depends on the network topology and that a topology with leaves has more room for improvement. Fig. 4 reports $L^R_{\mathtt{H}}$ and $L^R_{\mathtt{LSA}}$ for the three real topologies we consider, and it confirms that in a real topology that has a balanced ratio between leaf nodes and core nodes, $L^R_{\mathtt{LSA}}$ is around 25% and $L^R_{\mathtt{H}}$ ranges from 5% to 10%, aligned with the results obtained using MM graphs.

A. Tests on real topologies

We run the emulation on the three mentioned real topologies, Fig. 5 reports the curve of the $\tilde{L}_A$ for each run on the Ninux topology, due to lack of space we report only aggregate values for the other two networks in Table II. The results show that for the majority of the nodes there is a substantial absolute improvement in $\tilde{L}_A$. The global loss reduction value $\tilde{L}_g$ reaches 0.28, which means that in the general case in which we assume an uniform traffic matrix, the use of PopR will decrease the total packet loss of a value proportional to $\tilde{L}_g$ in average. We believe this is an extremely relevant result since we stress again this reduction is obtained without increasing the overall number of generated control messages.

Table II reports the relative loss reduction $\tilde{L}_R$ computed on the ten nodes with the highest centrality and shows that on those nodes, which are the most critical ones for the whole network we can achieve as much as 69% loss reduction. Note that a higher centrality does not always reflect in a higher $\tilde{L}_R$. This is due to two factors, the first is that in some cases the failure of a very central node partitions the network, some nodes remain isolated and we have to reduce the overall number of considered routes. The second is that strictly imposing the equivalence in the total amount of $\mathtt{H}$ messages in equation 7 penalises the nodes that have many neighbors, because equation 12 depends linearly from $\sqrt{d_i}$. Oftentimes central nodes also have many neighbors and their $t_{\mathtt{H}}(i)$ is limited by this factor. Notice in fact that the values of $t_{\mathtt{A}}(i)$ grows monotonically with the centrality while the value of $t_{\mathtt{H}}(i)$ doesn't. A further improvement could be to relax condition equation 7 replacing the term d_i with the average node degree. This would modify the overall number of control messages but would not penalise nodes with high centrality, thus producing even better results.

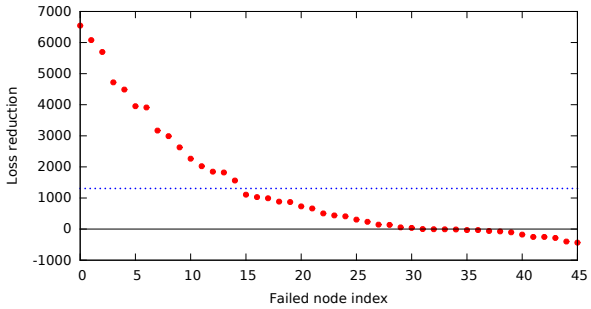

Figure 5. Empirical loss reduction computed on the Ninux network, with average value reported as the dotted line

Ninux (top ten nodes by centrality)										
$\tilde{L}_R$	0.50	0.36	0.43	0.43	0.25	0.38	0.25	0.41	0.38	0.32
$t_{\mathtt{H}}(i)$	1.42	1.33	1.48	1.36	1.06	0.89	1.83	2.17	1.96	2.20
$t_{\mathtt{A}}(i)$	1.81	1.81	2.01	2.18	2.20	2.28	2.32	2.47	2.49	2.50
Ninux $\tilde{L}_g$: **0.28**			Ninux max $t_{\mathtt{H}}(i)$: 4.51s				Ninux max $t_{\mathtt{A}}(i)$: 8.1s			
FFGraz (top ten nodes by centrality)										
$\tilde{L}_R$	0.31	0.60	0.40	0.30	0.49	0.31	0.48	0.26	0.36	0.16
$t_{\mathtt{H}}(i)$	1.34	1.40	1.49	1.68	1.38	1.53	1.09	1.53	0.74	1.87
$t_{\mathtt{A}}(i)$	1.38	1.44	1.66	1.72	1.77	1.86	1.87	1.97	2.03	2.08
FFGraz $\tilde{L}_g$: **0.27**			FFGraz max $t_{\mathtt{H}}(i)$: 4.6s				FFGraz max $t_{\mathtt{A}}(i)$: 7.23s			
FFWien (top ten nodes by centrality)										
$\tilde{L}_R$	0.44	0.29	0.40	0.69	0.34	0.30	0.24	0.20	0.37	0.26
$t_{\mathtt{H}}(i)$	1.25	1.36	1.43	1.04	1.65	1.35	1.51	1.60	1.22	1.61
$t_{\mathtt{A}}(i)$	1.15	1.47	1.50	1.59	1.63	1.76	1.88	1.99	1.99	2.0
FFWien $\tilde{L}_g$: **0.20**			FFWien max $t_{\mathtt{H}}(i)$: 4.24s				FFWien max $t_{\mathtt{A}}(i)$: 7.45s			

Table II
SUMMARY OF RESULTS FOR THE THREE TOPOLOGIES

Results for the two other networks are reported in Table II and confirm that results are extremely significant in all the topologies. We also report the maximum value for $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ to show that the optimization problem is well conditioned so that the timers do not diverge to unusable values.

IX. CONCLUSIONS

Fine tuning the generation of signaling message is of the utmost importance for the performance of routing protocols, link-state protocols in particular. One of the key performance

indexes is the capability of fast recovery after a node (or link) failure. Yet, after decades of link-state protocol use, experience with, and research on, there is not an automatic, let alone optimal, procedure to set the generation times of signaling messages.

This work has formalized the problem of minimal disruption after a node failure as an optimization problem in the space of signaling messages (`HELLO` and `Link-state Advertisement`) generation timers, subject to the constraint that the total overhead in terms of messages per second remains constant for each category of signaling messages. The solution of the problem, with some modeling assumptions that allows the use of the Lagrange Multipliers technique, turns out to be computationally very efficient, so that it can be implemented on-line even on low-power devices and for networks of hundreds of nodes.

The theoretical analysis on topologies with power-law characteristics shows that the network disruption, measured in number of broken or looped routes multiplied by the time they are mis-functioning, can be reduced by 30% and more. Furthermore, measures on the topology of real wireless mesh networks obtained extending the OLSR protocol to include the proposed technique show an equivalent improvement, validating the approach and showing that the modeling assumptions are correct.

The technique can also be extended to distance-vector protocols, such as BABEL, BATMAN or BMX6 that are largely used in wireless mesh networks. These protocols use `H` messages for link detection, but they do not distribute information on the whole topology, so centrality metrics can be only approximated. The exploration of PopR performance applied to these protocols will be part of future works, together with the application of PopR to other timers, such as the timers used to hold message to perform aggregation on intermediate routers.

REFERENCES

[1] F. Clad, P. Merindol, J.-J. Pansiot, P. Francois, and O. Bonaventure, "Graceful Convergence in Link-State IP Networks: A Lightweight Algorithm Ensuring Minimal Operational Impact," *IEEE/ACM Transactions on Networking*, vol. 22, no. 1, pp. 300–312, Feb. 2014.

[2] C. Gomez, D. Garcia, and J. Paradells, "Improving performance of a real ad-hoc network by tuning OLSR parameters," in *10th IEEE Symposium on Computers and Communications, (ISCC)*, 2005.

[3] Y. Huang, S. N. Bhatti, and D. Parker, "Tuning olsr," in *IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications, (PIMRC)*, 2006.

[4] J. Toutouh, J. Garcia-Nieto, and E. Alba, "Intelligent OLSR routing protocol optimization for VANETs," *IEEE Transactions on Vehicular Technology*, vol. 61, no. 4, pp. 1884–1894, May 2012.

[5] A. Belghith and M. Abid, "Autonomic self tunable proactive routing in mobile ad hoc networks," in *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, (WIMOB)*, 2009.

[6] L. Guardalben, L. J. G. Villalba, F. Buiati, J. B. M. Sobral, and E. Camponogara, "Self-configuration and self-optimization process in heterogeneous wireless networks," *Sensors*, vol. 11, no. 1, pp. 425–454, Dec. 2010.

[7] D. F. H. Sadok, T. G. Rodrigues, R. D. M. Amorim, and J. Kelner, "On the performance of heterogeneous MANETs," *Wireless Networks*, vol. 21, no. 1, pp. 139–160, Jan. 2015.

[8] J. Yu, N. Wang, G. Wang, and D. Yu, "Connected dominating sets in wireless ad hoc and sensor networks a comprehensive survey," *Computer Communications*, vol. 36, no. 2, pp. 121–134, Jan. 2013.

[9] O. Liang, Y. A. Sekercioglu, and N. Mani, "A survey of multipoint relay based broadcast schemes in wireless ad hoc networks." *IEEE Communications Surveys and Tutorials*, vol. 8, no. 1-4, pp. 30–46, 2006.

[10] L. Maccari and R. Lo Cigno, "How to reduce and stabilize MPR sets in OLSR networks," in *IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WIMOB)*, 2012.

[11] J. H. Ahn and T.-J. Lee, "Multipoint relay selection for robust broadcast in ad hoc networks," *Ad Hoc Networks*, vol. 17, pp. 82–97, Jun. 2014.

[12] L. Maccari and R. L. Cigno, "Betweenness estimation in OLSR-based multi-hop networks for distributed filtering," *Journal of Computer and System Sciences*, vol. 80, no. 3, pp. 670 – 685, 2014.

[13] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen, "Scalable routing strategies for ad hoc wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 8, pp. 1369–1379, Aug. 1999.

[14] Y. Faheem and J. L. Rougier, "Loop avoidance for fish-eye OLSR in sparse wireless mesh networks," in *IEEE International Conference on Wireless On-Demand Network Systems and Services (WONS)*, 2009.

[15] M. Goyal, M. Soperi, E. Baccelli, G. Choudhury, A. Shaikh, H. Hosseini, and K. Trivedi, "Improving Convergence Speed and Scalability in OSPF: A Survey," *IEEE Communications Surveys Tutorials*, vol. 14, no. 2, pp. 443–463, 2012.

[16] P. Francois, M. Shand, and O. Bonaventure, "Disruption free topology reconfiguration in OSPF networks," in *IEEE International Conference on Computer Communications (INFOCOM) (Best Paper award)*, 2007.

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

[18] 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.

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

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

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

[22] 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.

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

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

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

[26] R. Puzis, P. Zilberman, Y. Elovici, S. Dolev, and U. Brandes, "Heuristics for Speeding Up Betweenness Centrality Computation," in *ASE/IEEE International Conference on Social Computing, International Conference on Privacy, Security, Risk and Trust*, 2012, pp. 302–311.

[27] B. Milic and M. Malek, "NPART - Node Placement Algorithm for Realistic Topologies in Wireless Multihop Network Simulation," in *International Conference on Simulation Tools and Techniques SIMUTOOLS*, 2009.