05-Aug-2018

Dear Prof. Maccari:

It is a pleasure to accept your manuscript entitled "Improving Routing Convergence with Centrality: Theory and Implementation of Pop-Routing" (TNET-2017-00225.R1), authored by Maccari, Leonardo; Lo Cigno, Renato, in its current form for publication in the IEEE/ACM Transactions on Networking. The comments of the reviewer(s) who reviewed your manuscript are included at the foot of this letter.

Thank you for your fine contribution. On behalf of the Editors of the IEEE/ACM Transactions on Networking, we look forward to your continued contributions to the Journal.

You have the option to make your article "open-access", i.e. available to non-subscribers from IEEE Xplore, for a fee of $1950 paid to the IEEE. If you wish for your article to be open access, please let the Publications Editor, Camille Ventura, know via email to c.ventura@ieee.org . Note that this fee is in addition to any page charges you may be assessed. More information about IEEE open access policy may be found at:

http://www.ieee.org/publications_standards/publications/authors/open_access.html

Published papers will be posted to Xplore and assigned permanent Digital Object Identifiers a few weeks after final files are accepted; volume, issue, and page numbers will be assigned at bimonthly intervals for onine issue publication to the papers in each issue. Reprints printed on paper will be available from the IEEE.

Please note that final submission uploads MUST be completed within 30 days of the date of this email. Please also note that by submitting your final files through ScholarOne Manuscripts, you are acknowledging and agreeing to any applicable page charges this paper may incur. If you have any questions about these charges, please contact the Journal Administrator prior to submitting your files.

Please find detailed instructions for final file submission below reviewers' comments.

Sincerely,
Prof. Song Chong
Associate Editor, IEEE/ACM Transactions on Networking
songchong@kaist.edu

Improving Routing Convergence with Centrality: Design and Implementation of Pop-Routing

Leonardo Maccari and Renato Lo Cigno
DISI – University of Trento, Italy – Email: leonardo.maccari@disi.unitn.it; locigno@disi.unitn.it

***Abstract*—One of the key features of a routing protocol is its ability to recover from link or node failures, recomputing routes efficiently without creating temporary loops. Indeed, in real conditions there is always a trade-off between the overhead due to the periodic generation of control messages and route convergence time. This work formalizes the problem of the choice of timers for control message generation as an optimization problem that minimizes the route convergence time, constrained to a constant signaling overhead. The solution requires the knowledge of nodes' centrality in the topology and can be obtained with a computational complexity low enough to allow on-line computation of the timers. Results on both synthetic and real topologies show a significant decrease of the transient duration with the consequent performance gain in terms of reduced number of unreachable destinations and routing loops. Our proposal is general and it can be applied to enhance any link-state routing protocol, albeit it is more suited for wireless networks. As a concrete example, we present the extension of OLSRv2 with our proposal, named Pop-Routing, and discuss its performance and the stability of centrality metrics in three large-scale real wireless mesh networks. This exhaustive analysis on traces of the topology evolution of real networks for one entire week show that Pop-Routing outperforms the non-enhanced protocol in every situation, even when it runs with sub-optimal timers due to centrality computation on stale information.**

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

I. Introduction

Recovery when a node or link fails is one of the key performance indicators of a routing algorithm, and link-state protocols have proven to perform better than distance-vector ones in this respect. Critical wired networks and high-bandwidth backbones use hardware redundancy, ad-hoc node and link failure detection, and pre-defined failover routes to minimize network disruption upon failure, often extending standard protocols to achieve the goal [2]. In many other cases, especially for fixed and mobile Wireless Mesh Network (WMN), this is not possible or too expensive, thus layer-3 control messages are used for: *i)* failure discovery; and *ii)* propagation of the new topology information. These two functions are implemented in all major link-state routing protocols through different periodic messages: `HELLO` (`H`) sent every $t_{\mathtt{H}}$ s, and `Link-State Advertisement` (`LSA`) sent every $t_{\mathtt{A}}$ s. `LSA` messages are called Topology Control messages in some protocols. `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.

Whenever a node fails, all the routes passing through it 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. Before the information is correctly propagated to the whole network, temporary routing loops can be created leading to further service disruption [3]. Fast convergence requires very small $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$; however, this not only implies a large overhead, but also triggers the risk of oscillations in case of temporary failures and frequent modifications of link costs (we describe the state of the art in this field in Sec. II). 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. So far this problem, albeit being at the core of any routing protocol, was not deeply investigated. In particular, no practical technique emerged to self-tune $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ *per-node* even if many protocols support differentiated per-node timers: Pop-Routing does exactly this.

We formalize the trade-off as an optimization problem: given a target overhead, for each n_i find the timers $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ that maximize the speed of route convergence (Secs. III to V). We derive a methodology that enables every node n_i of the network to locally solve it, and to find the exact optimal values of $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ that, based on the node *centrality* in the topology, maximize the performance (Sec. VI). Our approach is general, it can be applied to tune $t_{\mathtt{H}}(i)$ (which are at the base of any wireless routing protocol) or $t_{\mathtt{A}}(i)$ (for link-state protocols) in any routing protocol that makes it possible to compute betweenness centrality. We validate our approach on the open source implementation of the Optimized Link State Routing (OLSR) daemon and we show that route convergence of OLSRd in emulated networks with realistic topologies considerably improves when the routing daemon is configured with the optimal parameters derived with Pop-Routing (Secs. VII and VIII).

Pop-Routing requires every node to compute the betweenness centrality of every other node in the network. The complexity of this computation using the state-of-the-art algorithm is polynomial with the number of nodes [4], but still, it can

This work has been partially funded by the European Commission, H2020-ICT-2015 Programme, Grant Number 688768 'netCommons' (Network Infrastructure as Commons) and by the University of Trento with the grant for the project "*Wireless Community Networks: A Novel Techno-Legal Approach*". This paper extends the work"*Pop-Routing: Centrality-Based Tuning of Control Messages for Faster Route Convergence*" presented at Infocom 2016 [1]

be hardly performed in real-time in resource-limited devices such as wireless routers [5]. In the second part of the paper (Secs. IX and X) we use a data-set describing the behavior of three large-scale mesh networks to verify that re-computing $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ can be safely performed with an interval of tens of minutes incurring into a negligible sub-optimization. As a whole, the contribution of this paper covers the full spectrum of the subject, from the theoretical analysis down to the constraints for its implementation, which is currently undergoing and documented in Sec. IX.

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 decrease 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

Many works concerned with reducing network disruption after a failure have studied wireless networks, where the problem is more important and overhead more critical, thus we start with these works, and specially with those based on OLSR. Initially the values of $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$ have been studied to understand how they influence the delivery of packets [6], [7]. In [6] 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 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. A potential improvement strategy is to pre-calculate optimal values for the timers, which has been investigated in a series of works, the latest by Toutouh et al. [8] that uses optimization techniques and meta-heuristics. This approach assumes that there is an optimal static tuning of parameters for a large family of networks. Instead, we dynamically adjust the parameters based on the position of each node in the topology.

A few works try to autonomically tune the timers in mobile networks. A network cartography approach is used in [9], requiring the knowledge of the position of the nodes, while parameters are changed as a function of the network size in [10]. Protocol parameters have been studied for their obvious impact on the convergence times of routes and energy consumption in heterogeneous networks [11]. 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 [12] for a recent review on CDS, [13] for a survey on MPRs, and also recent works [14]–[16] exploring MPR and CDS nodes selection. PopR does not use a binary on/off flag for the generation of timers, but a continuous function which makes it possible to fine-tune the timers and achieve global optimality.

Fisheye routing [17] is a smart technique to reduce overhead. With Fisheye, LSA messages are sent with a constant timer, but with a variable time-to-live (TTL) field. Fisheye has important scalability properties [18] but also a serious risk: whenever two nodes in the network have a different view of the topology, they might take contrasting decisions and introduce routing loops [19]. 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. A survey on the issues related to convergence of OSPF and the proposed workarounds can be found in [2]. Among these, we mention IP fast re-routing or incremental update of link weights [3], [20]. These techniques also rely on failure detection and may be coupled with PopR like techniques, albeit extension to wired networks may require some further tuning.

In conclusions, PopR differs from all the known approaches because it does not define categories of nodes but increases or decreases $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$ using a continuous function computed locally by each node and it 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.

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 [21], but was not applied to communication networks until recently [22]. Centrality can be used to enhance network monitoring [23], intrusion detection and firewalling [24], [25], resources allocation [26] and topology control [27], [28].

PopR exploits *betweenness centrality*: the fraction of all possible shortest paths potentially 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 $\mathcal{O}(N^3)$. The fastest algorithm in literature is by Brandes and achieves exact computation of centrality in $\mathcal{O}(EN + N^2log(N))$ [29]. Recently Puzis et al. proposed an heuristic for centrality computation that introduces a speed-up in graphs with some specific topological features, that are common in real networks [4]. Puzis' algorithm pre-processes the topology and splits the problem in smaller domains. It computes the exact value of centrality: It is heuristic in the sense that it does not change the worst case complexity and it improves the performance only in graphs that can be split in several bi-connected components. For graphs that have only one giant component there is no time gain and some loss due to pre-processing.

We use state-of-the art algorithms to compute centrality, as it is not part of our proposal to improve the complexity

Table I: Main symbols used in the paper

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
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, its detection, information 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 information propagation time
d_i	degree of node n_i

of centrality computation. Indeed, leveraging Puzis' heuristic it is possible to show that, even using a very low-power device, the exact computation can be carried out for networks made of hundreds of nodes in seconds [5]. We exploit this result to further study how sensitive the betweenness metric is to topology changes in real networks. Topology analysis was carried out in several papers (see Vega et. al. and the reported bibliography [30]), but none of them, to the best of our knowledge, analyses the time variation of centrality metrics on a large time-window.

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 each edge corresponds to a link. We do not distinguish between the terms vertex/node and edge/link. Link endpoints correspond to logical interfaces at the IP level, thus in a wireless network router two or more links may share the same network 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 with TTL set to 1, so the packet is not re-broadcast by the neighbors. For simplicity, each edge has weight 1, so no quality metric is used to build the routing tables. Results can be directly extended 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 routing tables converge to an alternative path 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 is for the network. It is intuitive 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. The core of our proposal is to formalize this

[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 $|| \cdot ||$

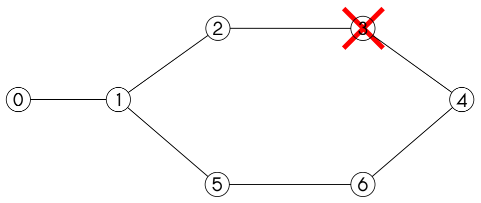

Figure 1: Sample topology used to exemplify PopR.

difference and to embed it in the protocol logic, in order to define differentiated timers $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ for every node n_i. We show that with a simple formulation that allows a compact solution, we can justify the goodness of our approach, not only, with some modifications we also allow to take into account more complex constraints, as we describe in Sec. VIII-C.

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 also implement PopR on OLSR to test it on a real protocol. We chose OLSR since it is a widely known and used protocol, with a stable open source implementation on which we can directly apply PopR.

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 to discover and maintain d_i neighbors. Each `H` message contains a *validity* field. A neighbor n_j of 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` generated by n_i contains the valid links $\{n_i, n_j\}$ for every neighbor n_j. `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_k does not receive a new `LSA` from n_i before the expiration of the validity timer, n_k 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 from 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. Since the transmission time in a wireless link is in the order of a few ms, the average number of hops in a network of hundreds of nodes is below 10 [31], and $t_{\mathtt{A}}(i)$ is in the order of seconds, again, this is a legitimate assumption. If intermediate nodes add a non-negligible delay before re-transmitting 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 large loss reduction.

Finally, note that failure detection via H must happen before information propagation via LSA, thus 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 lie in an acceptable region from the protocol point of view, all the tests we did resulted in timers within acceptable boundaries.

IV. Failure detection as an optimization problem

Referring 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:

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

b_k is a generic graph-based definition that is often used in the literature [21]. 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 $b_k \in [\frac{2}{N}, 1]$ as in the central node of a star topology[b].

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

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

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

[b]In some formulations 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.

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

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

Eq. (3) 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 $t_{\mathtt{H}}(k)$. 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.

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:

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

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

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

Note that we consider the overhead of a packet as proportional to the number of nodes that receive it, more details on this model can be found in Appendix A. 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 Eq. (3). Since the optimization is not influenced by the constants, we can safely remove them:

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

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

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 (a good flooding algorithm does not send the same message twice on the same link, so we can set $R = E$). The overhead due to LSA messages is:

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

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

To estimate the route disruption caused by delay in LSA messages we need more insight on 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 $V_{\texttt{A}} t_{\texttt{A}}(3)$ 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, and it recomputes its routing table excluding n_3.

The first way of discovery means that loops may exist at a node for a period $T_p - T_d = V_{\texttt{A}} t_{\texttt{A}}(i)$, where n_i is the failed node; $V_{\texttt{A}}$ is constant for all nodes, so it does not play any role in the optimization. In the second way of discovery the period $T_p - T_d$ does not depend on $t_{\texttt{A}}(i)$ when n_i fails, but it depends on $t_{\texttt{A}}(k)$ of some node n_k neighbor of n_i, since only neighbors can propagate a topology change before timer $V_{\texttt{A}} t_{\texttt{A}}(i)$ expires. In PopR we use the betweenness centrality of n_k to tune its timers and betweenness is correlated: neighbors of a node with high betweenness probably also have a high betweenness. This means that in this second case we can state that $T_p - T_d$ is proportional to some $t_{\texttt{A}}(k)$ of a neighbor of n_i, which is probably close to $t_{\texttt{A}}(i)$ of the failed node: $T_p - T_d \propto t_{\texttt{A}}(k) \simeq t_{\texttt{A}}(i)$. Constants do not influence the optimization, so we can safely state that also in this case minimizing the routes' disruption through $t_{\texttt{A}}(i)$ optimization is correct, albeit approximated. Thus, to solve the optimization problem, we simply consider $T_p - T_d \propto t_{\texttt{A}}(i)$.

A link-state protocol can also behave in a reactive way: It can anticipate the generation of an LSA when it senses the failure of a link, so in principle T_p could be very close to T_d. This feature is optional in OLSR and, in our experience, rarely used. The reason why is that while in a wired network link failure can be detected with link-level sensing and it is normally a long-lasting condition, in a wireless network failure detection can be due to temporary degradation and even to congestion conditions that leads to successive message collisions. If the protocol reacts too quickly to temporary conditions, then it can produce route flapping and consequent instability. Instead, quality link metrics with some hysteresis can be used to penalize a link that periodically faces congestion. Even if for simplicity we only deal with link failures, we note that PopR is effective also against other situations in which the reactive behavior simply can not be applied. If n_i does not fail, but for some reason the quality of its links decreases substantially (e.g., the node is subject to temporary shadowing), the effect is similar to a node failure (n_i is removed from many, and sometimes all, the shortest paths), but it is harder to detect since it is not an on/off situation. It can be detected and reacted upon using link quality thresholds, that introduce yet another tunable parameter in the protocol. Our model does not suffer from this complexity and behaves smoothly with the evolution of the network.

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

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

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 that does not create a loop, so 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 $t_{\texttt{A}}(i)$.

VI. Optimized Link-State Timers

The problem we defined for both $t_{\texttt{A}}(i)$ and $t_{\texttt{H}}(i)$ can be solved analytically; the full demonstration can be found in [1], here we report only the solution and its interpretation. The optimal values for $t_{\texttt{A}}(i)$ and $t_{\texttt{H}}(i)$ are given by:

$$t_{\texttt{H}}(i) = \frac{\sqrt{d_i}}{\sqrt{b_i}} \frac{1}{O_{\texttt{H}}} \sum_{j=1}^{N} \sqrt{b_j d_j} \tag{10}$$

$$t_{\texttt{A}}(i) = \frac{\sqrt{R}}{\sqrt{b_i}} \frac{1}{O_{\texttt{H}}} \sum_{j=1}^{N} \sqrt{b_j R} \tag{11}$$

and we can use them to compute the 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:

$$L_{\texttt{H}} = \frac{1}{O_{\texttt{H}}} \Big(\sum_{i=1}^{N} \sqrt{b_i d_i} \Big)^2 \tag{12}$$

$$L_{\texttt{LSA}} = \frac{1}{O_{\texttt{LSA}}} \Big(\sum_{i=1}^{N} \sqrt{b_i R} \Big)^2 \tag{13}$$

Eqs. (10) and (11) state that if n_i knows the betweenness and degree of the other nodes, it can easily compute the optimal value for $t_{\texttt{H}}(i)$. They 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 $t_{\texttt{H}}(i)$ and $t_{\texttt{A}}(i)$ in order to minimize the routes' disruption due to node failures while keeping the total overhead constant.

Note that we could further reduce the theoretical loss minimizing $L_{\texttt{H}} + L_{\texttt{LSA}}$ in a single minimization problem, instead of splitting the problem in two different ones. Since the space of the solutions would be larger we could find a distribution of timers that minimizes the loss even more. This has two drawbacks, that convinced us to split the problem in two. The first is that there is a functional relationship between `H` and `LSA`, that is, `LSA` propagate information that has been detected by using `H`. If we optimize both values together it may be that the best configuration for a certain n_i is given by $t_{\mathtt{A}}(i) < t_{\mathtt{H}}(i)$. This means that the process that performs link-sensing has a slower dynamic than the process that propagates link-sensing informations, which practically makes no sense. Splitting the optimization in two, we guarantee that the proportion between $t_{\mathtt{A}}(i)$ and $t_{\mathtt{H}}(i)$ is respected at least in their average value per node, and Sec. VIII shows that meaningful values are almost always computed. The second reason is that we want to outline that PopR is of general interest even for protocols that use only link-sensing based on `H`, possibly coupled with recent fully distributed techniques that make it possible to compute centrality without knowledge of the full topology [32]. For this reason we are interested in showing the benefits obtainable by tuning only `H` messages.

A. *Applicability of PopR*

The improvement achieved by PopR is perfectly compatible with any protocol that supports a differentiated timer for each node, and it can be used on top of any topology reduction strategy, like MPRs or CDS [33]. 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 select a new one.

In principle, the logic of PopR can be applied also to link-state protocols used in wired networks, such as OSPF. In practice, there are several reasons that make this impractical. Among therm, those protocols tend to be more reactive than protocols designed for wireless networks, for instance, 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. Moreover, high-end routers use link-level signalling to detect neighbor failures. Still, low-end routers that use the simple Bidirectional Forwarding Detection (BFD) protocol (which is based on `HELLO`s) can benefit from PopR to tune the frequency of `H` messages.

Finally, note that the array of betweenness values **B** changes with time, so periodically each node has to re-compute its own timers. However, with a minor change of **B** values PopR does not result in a service disruption, but just in a slightly sub-optimal generation of control messages. Sec. X is dedicated to the quantitative analysis of this mismatch.

VII. Evaluation Set-up

The rest of the paper presents three distinct sets of results that validate the theoretical approach of PopR, and its real world applicability. The first one is obtained applying directly the optimization derived in Sec. IV and Sec. V on syntetic graphs with controlled properties; the second set is obtained modifying the OLSRd code and running it on real topologies in an emulated environment; the third set is dedicated to study the behavior of running mesh networks to understand if PopR is compatible with real world constraints.

A. *Evaluation on synthetic and real graphs*

The first result set is the evaluation of the two loss formulas given by equations Eq. (12) and Eq. (13). Given a network graph $G(\mathcal{N}, \mathcal{E})$ we set $R = E$ (as typical for flood-based `LSA` distribution), $t_{\mathtt{H}} = 2\,\mathrm{s}$, $t_{\mathtt{A}} = 5\,\mathrm{s}$ (the OLSR default values) 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 with 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 loss computed with the optimal values of $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$. The absolute value of the performance loss is highly influenced by the topology, and also by the many constants that do not influence the optimal operation point. For this reason we use relative metrics of performance defined as

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

We use topologies generated following two popular models: *i)* The well known Barabási-Albert (BA) preferential attachment algorithm, that generates graphs with a power-law degree distribution; and *ii)* the model developed by Milic and Malek (MM) in [34]. 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 mesh networks made of 227, 143, and 126 nodes respectively, which are used daily by hundreds of people[c] [31].

B. *Evaluation using Mininet*

The second result set is produced using the Mininet network emulator[d]. Mininet enables 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

[c]For further nes and details on these mesh network visit http://www.funkfeuer.at/ and http://ninux.org/

[d]See http://mininet.org/ for a full description of the tool.

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 that we omitted in the analysis; and *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 packets 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 $R_t^i[\cdot]$ that associates a destination node n_k to the next hop n_j, so that $R_t^i[n_k] = n_j$;
2) Each $R_t^i[\cdot]$ is saved by every node every 300 ms together with the associated timestamp;
3) At time T_0 node n_k is forced to fail;
4) At time T_e, larger than the expected T_p for all nodes, when all $R_t^i[\cdot]$ are stabilized the emulation is stopped;
5) For each timestamp h navigate the routing tables from every source n_i to every destination n_j recursively, using the saved 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 $\{(T_0, r_0) \dots (T_e, r_e)\}$, each couple associates an instant after the node failure to the corresponding number of broken routes;
6) 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 show the number of broken paths due to both the detection phase (broken routes) and the propagation phase (paths with loops). $\tilde{L}$ 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 $\tilde{L}_R = \dfrac{\tilde{L}_A}{\tilde{L}_{olsr}}$.

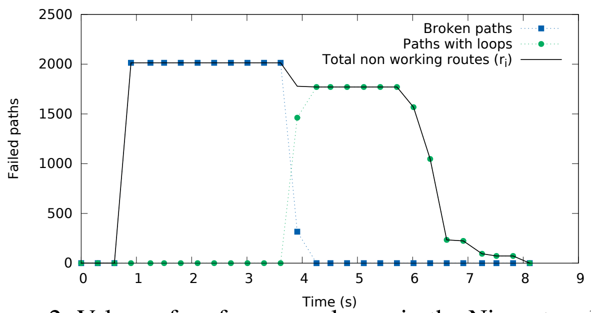

Figure 2: Values of r_h for a sample run in the Ninux topology. The three curves represent the number of broken paths (paths that pass through a failed node), the paths in which a loop is created, and the sum of the values.

To evaluate the average performance loss in a 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 that have an influence on the rest of the network. We are instead not interested in analyzing the convergence, for instance, when a leaf node fails, as only the traffic that was directed to that node will be affected. Thus in our emulations N_f is given by the number of nodes that, if removed from the network, will impact the other nodes' routes to some reachable node. Tab. II reports N_f for the three scenarios we emulated.

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 all needed to have an exhaustive evaluation of the theoretical and empirical performances of PopR:

$L_{\mathtt{H}}^R$, $L_{\mathtt{LSA}}^R$: 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 obtained emulating the failure of a generic node n_i;

$\tilde{L}_R$: relative empirical loss reduction obtained emulating the failure of a generic node n_i;

$\tilde{L}_g$: overall relative empirical loss reduction evaluated on all meaningful nodes' failures.

VIII. Experimental Results

Fig. 3 reports the relative loss reduction $L_{\mathtt{H}}^R$ and $L_{\mathtt{LSA}}^R$ on BA and MM synthetic topologies increasing the number of nodes. These metrics are computed averaging the losses on 30 randomly generated topologies. The performance of PopR improves as networks become larger, and results for $L_{\mathtt{LSA}}^R$ are

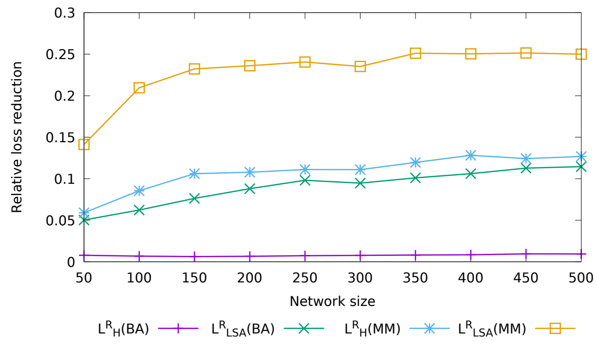

Figure 3: The relative theoretical loss reduction values due to PopR computed on Milic-Malek and Barabási-Albert graphs.

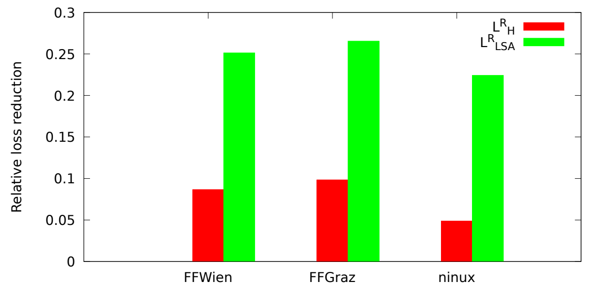

Figure 4: Relative loss reduction computed on the network topology of three running mesh networks made of 227, 143 and 126 nodes.

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 skewed centrality values, 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.

A. Tests on real topologies

Fig. 3 shows that the improvement given by PopR depends on the network topology and that a topology with skewed centrality values 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.

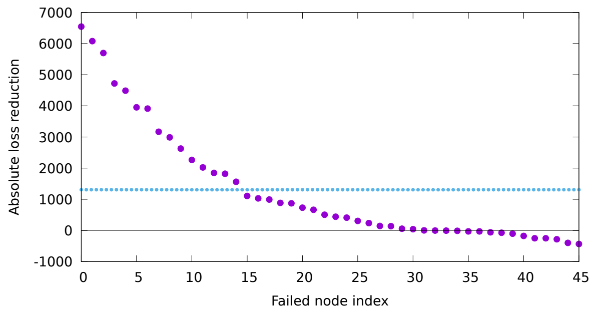

Figure 5: Absolute empirical loss reduction for each node computed on the Ninux network (126 nodes), with the average value reported as the dotted blue line.

B. Tests On Mininet

Fig. 5 reports $\tilde{L}_A$ for each node failure and its average in the Ninux topology. The results show that for the majority of the nodes there is a substantial absolute improvement in $\tilde{L}_A$, while there is only a slight performance loss when the least central nodes fail. For some of the failed nodes recovery time is slightly larger than what would be with standard timers, but on average (the dotted blue line) the gain is remarkable. $\tilde{L}_A$ results for FFWien and FFGraz are qualitatively equal to Ninux and we do not report them.

Table II reports the relative loss reduction $\tilde{L}_R$ computed on the ten nodes with the highest centrality for the three real topologies considered. These results are obtained implementing PopR in the OLSRd daemon and running it in Mininet emulations and show that on those nodes, which are the most critical ones for the whole network, PopR achieves up to 69% loss reduction. The figure also reports the average global loss reduction ($\tilde{L}_g$, computed on all the failures) that lies between 0.20 and 0.28, which confirms the overall efficacy of our approach implemented in real code.

$\tilde{L}_R$ is not monotonically increasing with centrality; this is due to two factors. The first one 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 H messages in Eq. (7) penalises the nodes that have many neighbors, because Eq. (10) depends linearly from $\sqrt{d_i}$. Frequently, central nodes also have many neighbors and their $t_{\mathtt{H}}(i)$ is limited by this factor. For these reasons the values of $t_{\mathtt{A}}(i)$ grow monotonically with the centrality while the values of $t_{\mathtt{H}}(i)$ don't show this trend. A possible modification could be to relax condition Eq. (7) replacing the term d_i with the average node degree. This would modify the overall number of control messages, but would not penalize nodes with high centrality, possibly producing even better results (see Appendix A for a detailed discussion on this choice).

The last row for each topology in Table II reports 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. Moreover, it reports the global loss reduction

Ninux (top ten nodes ranked by centrality, total nodes: 126)										
$\tilde{L}_R$	**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
$\tilde{L}_g$: **0.28**			Max $t_{\mathtt{H}}(i)$: 4.5s			Max $t_{\mathtt{A}}(i)$: 8.1s			N_f: 43	

FFGraz (top ten nodes ranked by centrality, total nodes: 143)										
$\tilde{L}_R$	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
$\tilde{L}_g$: **0.27**			Max $t_{\mathtt{H}}(i)$: 4.6s			Max $t_{\mathtt{A}}(i)$: 7.2s			N_f: 45	

FFWien (top ten nodes ranked by centrality, total nodes: 227)										
$\tilde{L}_R$	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	2.00
$\tilde{L}_g$: **0.20**			Max $t_{\mathtt{H}}(i)$: 4.2s			Max $t_{\mathtt{A}}(i)$: 7.4s			N_f: 165	

Table II: Normalized loss reduction $\tilde{L}_R$ and value of timers when each of the 10 most central nodes fail for the three real topologies considered; the last row for each topology reports the overall gain $\tilde{L}_g$, the maximum values of timers and N_f.

which means that assuming a uniform traffic matrix, the use of PopR would decrease the packet loss during recovery proportionally.

The key observation about this result is that this gain is obtained at no cost, i.e., *without increasing the protocol overhead*. The optimal equalization of the protocol timers alone reduces the convergence time of up to 28% in average and 69% as a peak when computed on real network topologies. Another key observation is that the relative gain metrics are ratios between loss equations defined in Eqs. (12) and (13). In the ratio, the terms $O_{\mathtt{H}}$ and $O_{\mathtt{LSA}}$ are cancelled, so that the gain compared to the standard configuration that PopR can achieve tuning H and LSA does not depend on the values chosen for $t_{\mathtt{H}}$ and $t_{\mathtt{A}}$. PopR exploits a specific property of the network graph (the distribution of the centrality values) and its gain does not depend on how aggressively the timers are set. Whatever configuration is used as a base comparison (as long as the timers are all the same) PopR will always improve. This is key to understand the value of our proposal, which is generic enough to be used in any similar context in which link-sensing is performed with H and/or link advertisement is performed by LSA.

C. *Further Improvements*

The results we showed confirm that PopR, leveraging a compact solution of an optimization problem can tangibly improve the recovery time upon node failure. Some modifications to PopR can be easily made to tailor its behaviour to different but related goals, while more work on the theoretical model would be needed to apply it to different cases. In this section we briefly review some ways to further improve PopR that will be the base for future works.

A first, minimal modification would be to allow an increase in the overhead to guarantee that no nodes incurs in a performance loss. In practice, one would set two upper bounds $\bar{t}_{\mathtt{H}}$ and $\bar{t}_{\mathtt{A}}$ for the timers that guarantee raising the values of the right tail of the curve in Fig. 5 to 0. The total overhead would be increased of a quantity that is graph-dependent. Another simple modification would be to protect some important traffic flows, removing some nodes from the optimization, and setting a fixed (low) timer for them. This would preserve some traffic flows that are known to be passing through those nodes and are considered particularly important. This would again produce an increase in the overhead, and possibly also a sub-optimal results for a given overhead, but it is straightforward to implement.

A more general approach is to impose upper bounds to the timers, complicating the minimization problem with inequalities of the kind $t_{\mathtt{H}}(i) < \bar{t}_{\mathtt{H}}$ and $t_{\mathtt{A}}(i) < \bar{t}_{\mathtt{A}}$. Similarly, if one wants to optimize both timers together while maintaining a specific difference between them, conditions of the kind $t_{\mathtt{H}}(i)k \leq t_{\mathtt{A}}(i)$ for some given k can be introduced. These modifications require a different solution as Lagrange Multipliers can not be used with inequalities. Some other technique (possibly the Karush-Kuhn-Tucker conditions) are needed to implement them, leading to a novel contribution.

Another interesting extension can be exploring the effects of imposing a finite set of choices for the timers, for example because an operator doesn't fully trust the algorithm to automatically set the timers, but wants to manually set them choosing from a set of pre-defined values. In this case PopR could simply suggest the best value to the operator, who can confirm or change the choice. This would move the problem in the discrete space and make it close to a classical knapsack problem, again, requiring a different solution technique. All these modifications to PopR are possible, and can be the base for future research.

IX. Implementing Pop-Routing: Prince

Results of Sec. VIII are obtained pre-configuring each OLSRd daemon with the optimal timers, and running a series of emulations. While pre-configured values can be used for testing purposes, this approach is hardly usable in a real application, since mesh networks are dynamic and frequent re-computation of the centrality values may be needed. Thus, we realized Prince, an open source daemon that implements PopR on top of the OLSRd daemon in *quasi* real time. The implementation of Prince incorporates real world constraints that opened new challenges for PopR, that we discuss and solve in the next sections. The next set of results are key to appreciate the value of PopR not only as a theoretical contribution but also as a technology ready to be adopted.

A. *Implementing Puzis' Heuristic*

The asymptotic complexity of centrality computation in a network graph is dominated by the computation of all the shortest paths, so it is polynomial with the number of nodes in the network. Brandes' algorithm achieves a complexity of $\mathcal{O}(NE + N^2log(N))$ and it is the fastest algorithm for weighted graphs. Puzis's heuristic [4] can reduce the computation time of Brandes' algorithm for networks with certain features. The first step to design Prince was to implement Puzis' heuristic and test its performance on real hardware. Preliminary results have shown that for a network made of

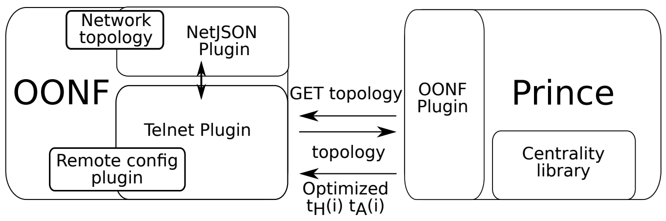

Figure 6: The interaction of Prince and OONF.

236 nodes, a low-cost wireless router (an Ubiquiti M5 wireless router, equipped with a MIPS 390MHz processor and 32M of RAM) requires 7 seconds to compute the centrality of each node using Puzis' heuristic, less than $\frac{1}{3}$ of the time required by Brandes' algorithm [5].

This outcome shows three facts: i) even low-power devices can compute centrality for networks made of hundreds of nodes; ii) centrality computation can not be performed in real time; iii) centrality computation can not be performed in the same process of the routing daemon since the routing daemon can not be frozen for several seconds. Centrality computation needs a separate process with low priority that will not interfere with the routing daemon, and thus may take several tens of seconds to update the centrality values on large networks. Sec. IX-B and Sec. X address two challenges for the real implementation of Pop-Routing: i) the interactions between the two processes, and ii) the correct timing for the re-computation of the centrality array **B**.

B. The Architecture of Prince

Prince is the open source implementation of the Pop-Routing principle for the successor of the OLSRd routing daemon. Recently, the standardization of the second version of OLSR was completed with a set of RFCs [35]–[38] that detail the messages, the metrics and the way this link state protocol works. The collection of these specifications takes the name OLSRv2. OLSRv2 maintains the basic functions we described of OLSRv1, so PopR can be applied to OLSRv2 as well. The OLSRd daemon was upgraded to OLSRv2 and in the process it was rebranded as "**o**lsrd.**o**rg **N**etwork **F**ramework", OONF [39]. OONF is under active development and supports plugins that help developers to add new features to the daemon. In particular three plugins are relevant, the NetJSON, the "remote config", and the Telnet plugin. NetJSON is a recently proposed format to describe network topologies[e] and OONF is one of the several implementation of routing protocols that can export the network topology using NetJSON. The remote config plugin allows changing the configuration parameters at run-time; finally the Telnet plugin can be used to access the other plugins remotely.

Prince is a separate daemon that communicates with OONF, it periodically polls OONF to receive the NetJSON topology, it computes the new timers for the node and pushes them to OONF via the Telnet plugin. The structure of Prince is described in Fig. 6. Prince is made of a main process in C language and a separate centrality library that implements Puzis' heuristic. The communication with OONF takes place via a dedicated plugin. This simple structure and the well-defined interface makes it possible to decouple centrality computation from the routing daemon and to extend Prince with new plug-ins for other link-state protocols. Prince is open source and freely available[f].

[e]See http://netjson.org

X. Re-computing centrality

In a network of N nodes every n_i receives roughly $\frac{N-1}{t_A}$ LSA per second. Every LSA potentially carries the information related to a topology variation, and the substantial modification of even only one link quality can modify the shortest paths between many couples of nodes. Thus, in principle, for every received LSA the array of centrality should be re-computed together with the values of $t_H(i)$ and $t_A(i)$. As we have commented in Sec. II-A the computation of betweenness centrality is polynomial with the number of nodes, but we have showed in [5] that computing the centrality in real networks with the elaboration power of an embedded device (a wireless router) may take up to tens of seconds, which in realistic conditions (more processes running on the router) may increase even more. It is clear that centrality can not be re-computed in real time every time a new LSA is received, but must be periodically updated. Using a large update interval has a small impact on the router CPU, but it also implies that for a long time $t_H(i)$ and $t_A(i)$ will be set according to an optimization done for a topology that possibly changed in the meantime, and thus far from the optimal ones. In the worst case, they could behave worse than the default values. We have to decide a re-computation timer Δ that is large enough to allow the computation even on low-power devices but short enough to follow the evolution of the network. From now on we will refer to a small, medium and large Δ when it falls below 5 minutes, from 5 to 20 minutes and from 20 to 60 minutes respectively.

The only way to estimate a suitable value of Δ is to analyze data extracted from real networks and verify the trend of variation of the value of centrality per each node. We analyze the three middle-size networks that we introduced in Sec. VII for a period of 7 days. For each network we downloaded the snapshots of the topology exported by OLSRd: One snapshot every 5 minutes for FFW and ninux, one every 10 minutes for FFG, corresponding to roughly 2000 and 1000 snapshots respectively. For each snapshot and for each node i we compute $t_H(i)$ and $t_A(i)$ as per Eqs. (12) and (13) and we analyze their trend in time.

A. Timers' Stability

Fig. 7 reports the average and standard deviation of $t_H(i)$ and $t_A(i)$ computed on all the snapshots, for every node i in each of the three networks (for readability the plots do not contain the values relative to nodes that remain leaf nodes in all the samples, since their value never changes). The plot shows that the coefficient of variation is sufficiently small (the

[f]The source code can be found at https://github.com/AdvancedNetworkingSystems/poprouting

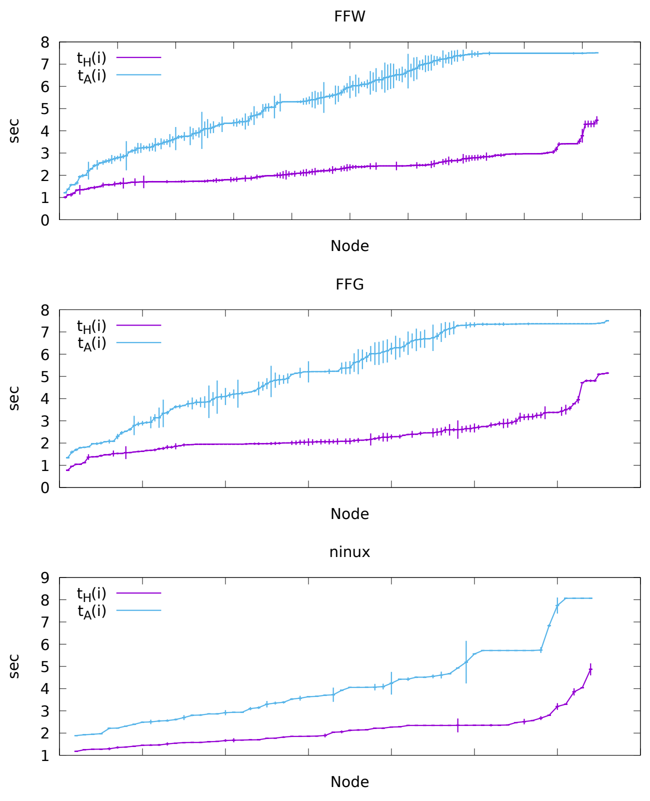

Figure 7: The average and standard deviation of $t_{\text{H}}(i)$ and $t_{\text{A}}(i)$ for the three networks under consideration. Each curve is ordered for increasing average value.

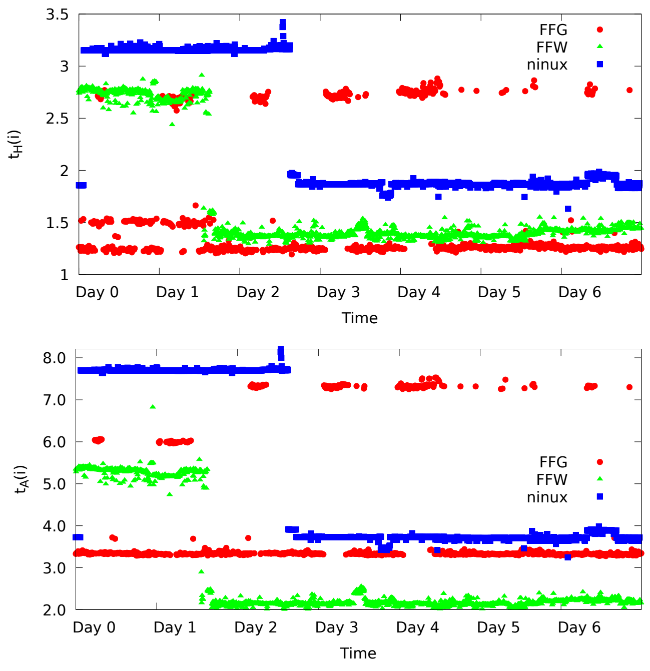

Figure 8: The samples of $t_{\text{H}}(i)$ (upper graph) and $t_{\text{A}}(i)$ (lower graph) for the three nodes with the highest coefficient of variation (FFG: red circles, FFW: green triangles, ninux:blue squares).

average lays below 8% for both $t_{\text{H}}(i)$ and $t_{\text{A}}(i)$) even in a week sampling time. At the right extreme of $t_{\text{A}}(i)$ curves there is a set of nodes with a very stable timer. These nodes are not leaf nodes (they are not in the plot) but behave as such. They have one good link that they primarily use, plus other bad links whose cost is high enough that they are never used to route traffic. Since betweenness is computed on the weighted graph, their betweenness takes the minimal value (exactly as a leaf node) and changes only marginally. The same effect does not appear in the graph of $t_{\text{H}}(i)$ since d_i is present in Eq. (10) so nodes with different degree take different values of $t_{\text{H}}(i)$, and when the number of neighbors of node i changes, $t_{\text{H}}(i)$ changes too. In Appendix A we give a more detailed interpretation of the value of d_i.

Fig. 7 suggests that the values taken by timers have a small interval of variation, but does not help understanding the time correlation of the timers, which is what we are mostly interested into. Before generalizing, we analyze the nodes with the extreme behaviour: we choose the nodes that have the largest coefficient of variation for $t_{\text{H}}(i)$ and $t_{\text{A}}(i)$, and report the values of the timers in Fig. 8. In ninux the high variation depends on a sharp transition from one state to another; before and after the transition, the values of $t_{\text{H}}(i)$ and $t_{\text{A}}(i)$ are pretty stable. This change reflects a topology modification that directly impacts the centrality of the node. In FFW the trend is similar to ninux, but with a higher deviation in the stable states, indicating that FFW topology changes are more frequent that in ninux. Remember that Eqs. (10) and (11) include the centrality of all the nodes, so a single modification in the network topology may cause small changes in the timers of *all* the nodes. In practice this oscillation is negligible, but from one snapshot to the next the timers are likely to fluctuate lightly even in stable conditions. In FFG, instead, timers oscillates between 2 or 3 different states, with one that is more likely (the one with the lowest value). This behaviour is probably related to some flapping link, a faulty router that periodically reboots, or temporary congestion.

This analysis shows that in general the optimal timers are stable, and even the nodes that display the highest variability have a stable operation, but for some sharp transitions due to topology changes. This indicates that using a large re-computation interval Δ, should not lead to unacceptable performance loss, except, maybe when a state transition for a node occurs, and this require further investigation.

B. Stability Impact on Performance

As we have seen, $t_{\text{H}}(i)$ and $t_{\text{A}}(i)$ are subject to variations with topology changes, some of them are small, noise-like, and should not affect the optimization; others are larger and may lead to loss of performance. Thus, we present a sensitivity analysis on the re-computation interval Δ: How much it influences the potential loss that derives from a node failure?

Let P_S be the period used to take snapshots of the network (five or ten minutes in our case). k indexes the snapshots $G[k]$, and $\delta = \left\lfloor \frac{\Delta}{P_S} \right\rfloor$ measures how many snapshots pass between the re-computation of the optimal timers done every Δ s. With this notation we extend the loss metrics L_{H} and L_{LSA} to take into account time evolution. Starting from Eqs. (12) and (13) we derive $L_{\text{H}}[k,\delta]$ and $L_{\text{LSA}}[k,\delta]$ that are the theoretical loss of PopR computed on snapshot k of the network, with the timers that were computed to optimize convergence in snapshot $k-\delta$. For the sake of clarity we separate the three cases:

- $\delta = 0$ in which we optimize at every step. Clearly this is the optimal case as the metric is updated continuously and $L_{\text{H}}[k,0] = L^{\star}_{\text{H}}$ and $L_{\text{LSA}}[k,0] = L^{\star}_{\text{LSA}}$;
- $1 < \delta < k-1$ that is a generic sub-optimized version of PopR;
- $\delta = \infty$: This is the limit case in which we optimize the timers for $k = 0$ and never again, and we refer to loss values in this case as $L_{\text{H}}[k,\infty]$ and $L_{\text{LSA}}[k,\infty]$ to highlight the fact that there is no update of the timers during normal network operation.

We can now introduce the corresponding relative performance metrics:

$$L^{R}_{\text{H}}[k,\delta] = 1 - \frac{L_{\text{H}}[k,\delta]}{L^{\star}_{\text{H}}}; \quad L^{R}_{\text{LSA}}[k,\delta] = 1 - \frac{L_{\text{LSA}}[k,\delta]}{L^{\star}_{\text{LSA}}} \quad (14)$$

$$L^{\star R}_{\text{H}}[k] = 1 - \frac{L_{\text{H}}[k,\infty]}{L^{\star}_{\text{H}}}; \quad L^{\star R}_{\text{LSA}}[k] = 1 - \frac{L_{\text{LSA}}[k,\infty]}{L^{\star}_{\text{LSA}}} \quad (15)$$

$$L^{R}_{\text{H}}[k] = 1 - \frac{L_{\text{H}}[k,\infty]}{L_{\text{H}}}; \quad L^{R}_{\text{LSA}}[k] = 1 - \frac{L_{\text{LSA}}[k,\infty]}{L_{\text{LSA}}} \quad (16)$$

These metrics express the relative gain of a strategy against another, computed on loss measures. The metrics in Eq. (14) measure the relative performance of PopR with $\delta > 0$ against PopR with $\delta = 0$. We expect these two metrics to be negative, as for $\delta > 0$ the timers are sub-optimal and the performance loss should be larger. Those in Eq. (15) compare the extreme case when timers are optimized only once at network start-up against the case with times continuously optimized. These metrics should degrade (become more negative) as k becomes larger as we expect topologies and centrality to slowly change in time, but it is difficult to predict the trend of this degradation. The metrics in Eq. (16), finally, compare the performance when timers are set at network start-up against the performance of standard OLSR. In this case the metric should be positive, as we expect in any case that tuning the timers to the topology properties leads to better performance than no tuning at all, but again it is very difficult to have quantitative predictions.

Since there are several plots to discuss, we present only the results for FFW, which in the previous analysis was the less stable network. Results for ninux and FFG are in general slightly better and confirm the discussion.

Fig. 9 reports $L^{R}_{\text{LSA}}[k,\delta]$ for the first 5 hours of network evolution and $\delta = 1,4,9$ and helps to have a qualitative understanding of the metrics before we show plots based on a larger dataset. The curves return to zero with an interval equal to $\delta+1$, when timers are recomputed; the relative loss never exceeds -2% compared to the optimal. The graph also shows that in some cases the values of $t_{\text{H}}(i)$ and $t_{\text{A}}(i)$ computed for snapshot $k-\delta$ perform even better than the ones computed for snapshot k. This behaviour is counter-intuitive, but it can be explained easily. Recall that $t_{\text{H}}(i)$ and $t_{\text{A}}(i)$ are derived from the optimization of the convergence time constrained to a constant overhead. In some cases the old timers perform better than the new ones because the topology is changed and they generate more overhead compared to the optimal solution. For instance, when some nodes are removed from the network, the constant O_H should be recomputed to rescale the values of all timers. If this is not done the timers may remain more aggressive and lead to a slightly better performance.

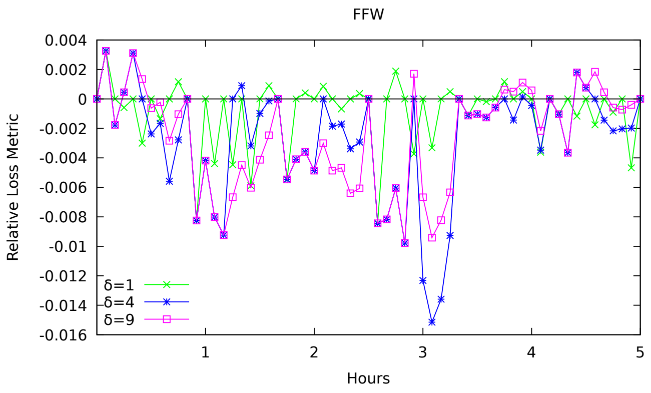

Figure 9: The values of samples of $L^{R}_{\text{LSA}}[k,\delta]$ for $\delta = 1,4,9$ in FFW (one snapshot every 5 minutes).

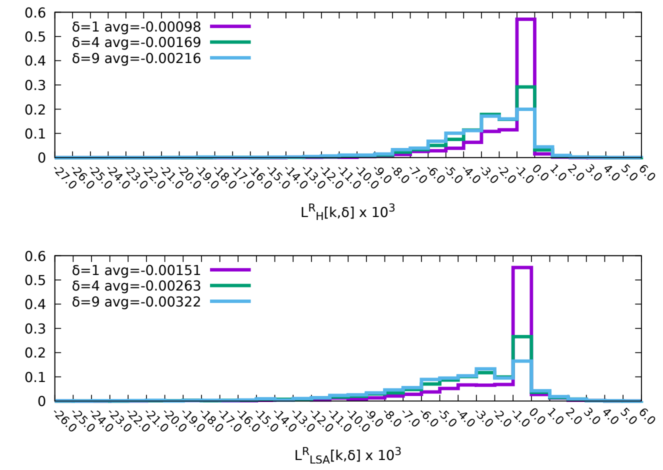

Figure 10: Binned values of $L^{R}_{\text{H}}[k,\delta]$ for $\delta = 1,4,9$ (upper plot) and $L^{R}_{\text{LSA}}[k,\delta]$ for $\delta = 1,4,9$ (lower plot) in FFW.

Fig. 10 generalizes these results and shows the binned distribution of the relative loss metrics for the same three values of δ. The bin size is 1.5×10^{-3} and the statistic is computed on the samples of the whole week. As expected, the average loss slightly increases with the increase of δ and the mass of the value is mostly confined in the $[-0.02, 0]$ interval, while in the keys the average value is also reported. In practice, we can say that sub-optimization has a negligible impact on the performance of PopR.

Finally, Figs. 11 and 12 further extend the results and corroborates the idea that Δ can be large. The colored filled dots are the values of $L^{R}_{\text{H}}[k,\delta]$ and $L^{R}_{\text{LSA}}[k,\delta]$ for δ ranging from 1 to 19. For every possible value of k on the k axis there

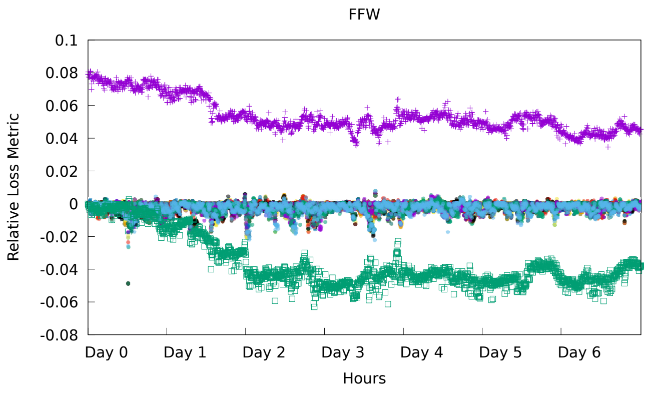

Figure 11: Value of $L^{R}_{\mathtt{H}}[k,\delta]$ for $\delta = 1\dots19$ (coloured dots), $L^{\star R}_{\mathtt{H}}[k]$ (empty green boxes), $L^{R}_{\mathtt{H}}[k]$ (purple crosses).

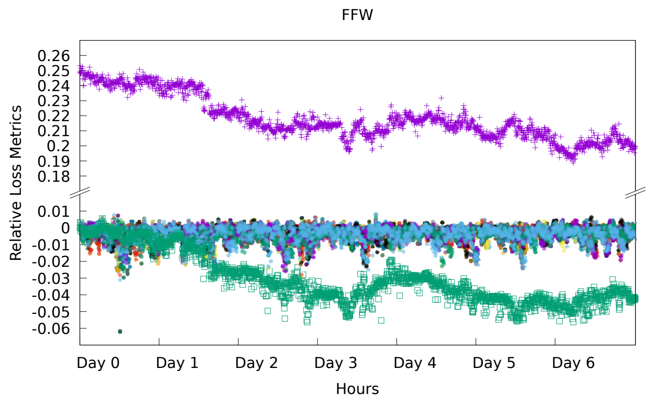

Figure 12: Value of $L^{R}_{\mathtt{LSA}}[k,\delta]$ for $\delta = 1\dots19$ (coloured dots), $L^{\star R}_{\mathtt{LSA}}[k]$ (empty green boxes), $L^{R}_{\mathtt{LSA}}[k]$ (purple crosses).

are 19 separate dots, one per value of δ. We use a distinct color for each value of δ, but the goal of the plot is not to discriminate between the dots corresponding to distinct values of δ: it is to show that even if we hold the values of $t_{\mathtt{H}}(i)$ and $t_{\mathtt{A}}(i)$ for 20 intervals (corresponding to $\Delta = 100$ minutes) the sub-optimization is limited to less than 6.3%. The green empty boxes are the values of $L^{\star R}_{\mathtt{H}}[k]$ and $L^{\star R}_{\mathtt{LSA}}[k]$, and the purple crosses are the values of $L^{R}_{\mathtt{H}}[k]$ and $L^{R}_{\mathtt{LSA}}[k]$. These series show that even if we never re-compute the timers for a whole week, the performance of pop-routing deteriorates (with some oscillations) but still remains close to the optimal value and always outperforms standard OLSR, and that for `LSA` the gain remains fairly high.

The results (consistent with the results for FFG and ninux) confirm that in a real network we can safely recompute the timers using a large Δ with a very small sub-optimization. In general, it is not worth to track temporary and limited topology changes: Even if they have a non-negligible local impact the global behaviour of the network is not strongly affected.

XI. Conclusions

Tuning the generation of control messages 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 use, experience with, and research on link-state protocols, there is not an automatic, let alone optimal, procedure to tune the timers for control message generation.

This work formalized the problem of route convergence after a node failure as an optimization problem in the space of the timers for the generation of control messages (`HELLO` and `Link-State Advertisement`), subject to the constraint that the total overhead in terms of messages per second remains constant for each category of control messages. The solution of the problem is computationally efficient, and it makes it possible for every node in the network (even on low-power devices) to auto-tune its own timers. Our results on emulated networks using the OLSR routing protocol show that the reduction of the convergence time after a node fails can reach 69% in the best case, and stays stably above 20% in the average case when tested on real network topologies.

From the initial tests performed on real hardware we observed that the performance of centrality computation does not allow to re-compute centrality in real-time. Therefore, we analyzed the behavior of three real mesh networks to identify the sensitivity of the optimization to the changes in the network topology. We observed that running networks are pretty stable and the impact of topology modifications on the optimization is marginal if we update the centrality computation with an interval of tens of minutes.

Acknowledgements

We would like to thank our former students Quynh P.X. Nguyen, Gabriele Gemmi and Matteo Bridarolli for the support in the implementation and testing of Prince and of the centrality library.

References

[1] L. Maccari and R. Lo Cigno, “Pop-Routing: Centrality-Based Tuning of Control Messages for Faster Route Convergence,” in *IEEE Int. Conf. on Computer Communications (INFOCOM)*, Apr 2016, pp. 1–9.
[2] 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.
[3] 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 Trans. on Networking*, vol. 22, no. 1, pp. 300–312, 2014.
[4] R. Puzis, P. Zilberman, Y. Elovici, S. Dolev, and U. Brandes, “Heuristics for Speeding Up Betweenness Centrality Computation,” in *ASE/IEEE Int. Conf. on Social Computing, Int. Conf. on Privacy, Security, Risk and Trust*, Sep 2012, pp. 302–311.
[5] L. Maccari, Q. Nguyen, and R. Lo Cigno, “On the Computation of Centrality Metrics for Network Security in Mesh Networks,” in *IEEE Global Communications Conference (GLOBECOM)*, Dec 2016, pp. 1–6.
[6] 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)*, Jun 2005, pp. 16–21.
[7] Y. Huang, S. N. Bhatti, and D. Parker, “Tuning OLSR,” in *IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications, (PIMRC)*, Sept 2006, pp. 1–5.
[8] J. Toutouh, J. Garcia-Nieto, and E. Alba, “Intelligent OLSR Routing Protocol Optimization for VANETs,” *IEEE Trans. on Vehicular Technology*, vol. 61, no. 4, pp. 1884–1894, 2012.
[9] A. Belghith and M. Abid, “Autonomic Self Tunable Proactive Routing in Mobile Ad Hoc Networks,” in *IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications, (WIMOB)*, Oct 2009, pp. 276–281.

[10] 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, 2010.

[11] 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, 2015.

[12] 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, 2013.

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

[14] L. Maccari and R. Lo Cigno, "How to reduce and stabilize MPR sets in OLSR networks," in *IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WIMOB)*, Oct 2012, pp. 373–380.

[15] 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.

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

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

[18] C. Adjih, E. Baccelli, T. Clausen, P. Jacquet, and G. Rodolakis, "Fish eye olsr scaling properties," *Journal of Communications and Networks*, vol. 6, no. 4, pp. 343–351, Dec 2004.

[19] Y. Faheem and J. L. Rougier, "Loop avoidance for Fish-Eye OLSR in sparse wireless mesh networks," in *IEEE Int. Conf. on Wireless On-Demand Network Systems and Services (WONS)*, Feb 2009, pp. 231–234.

[20] P. Francois, M. Shand, and O. Bonaventure, "Disruption Free Topology Reconfiguration in OSPF Networks," in *IEEE Int. Conf. on Computer Communications (INFOCOM)*, May 2007, pp. 89–97.

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

[22] 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.

[23] 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.

[24] 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, 2011.

[25] L. Maccari and R. Lo Cigno, "Waterwall: a cooperative, distributed firewall for wireless mesh networks," *Jou. on Wireless Communications and Networking*, vol. 2013, no. 225, pp. 1–12, 2013.

[26] 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.

[27] A. VÃązquez-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.

[28] L. Baldesi, L. Maccari, and R. Lo Cigno, "On the Use of Eigenvector Centrality for Cooperative Streaming," *IEEE Communications Letters*, vol. 21, pp. 1953–1956, 2017.

[29] U. Brandes, "A Faster Algorithm for Betweenness Centrality," *Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[30] D. Vega, L. Cerda-Alabern, L. Navarro, and R. Meseguer, "Topology patterns of a community network: Guifi.net," in *IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2012, pp. 612–619.

[31] 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.

[32] 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 *IEEE International Conference on Computer Communications (INFOCOM)*, Honolulu (USA), Apr 2018.

[33] T. H. Clausen, P. Jacquet, D.-Q. Nguyen, and E. Baccelli, "RFC 5449: OSPF Multipoint Relay (MPR) Extension for Ad Hoc Networks." [Online]. Available: https://tools.ietf.org/html/rfc5449

[34] B. Milic and M. Malek, "NPART - Node Placement Algorithm for Realistic Topologies in Wireless Multihop Network Simulation," in *Int. Conf. on Simulation Tools and Techniques (SIMUTOOLS)*, Mar 2009, pp. 9:1–9:10.

[35] U. Herberg, C. Dearlove, and T. Clausen, "Integrity protection for the neighborhood discovery protocol (nhdp) and optimized link state routing protocol version 2 (olsrv2)," RFC 7183 (Proposed Standard), Internet Engineering Task Force, Tech. Rep. 7183, Apr 2014.

[36] U. Herberg, R. Cole, and T. Clausen, "Definition of managed objects for the optimized link state routing protocol version 2," RFC 7184 (Proposed Standard), Internet Engineering Task Force, Tech. Rep. 7184, Apr 2014.

[37] C. Dearlove, T. Clausen, and P. Jacquet, "Link metrics for the mobile ad hoc network (manet) routing protocol olsrv2 - rationale," RFC 7185 (Informational), Internet Engineering Task Force, Tech. Rep. 7185, Apr 2014.

[38] C. Dearlove and T. Clausen, "Routing multipoint relay optimization for the optimized link state routing protocol version 2 (olsrv2)," RFC 7187 (Proposed Standard), Internet Engineering Task Force, Tech. Rep. 7187, Apr 2014.

[39] C. Barz, J. Niewiejskai, and H. Rogge, "NHDP and OLSRv2 for community networks," in *9th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Lyon, (France), Oct 2013.

Appendix A

In this section we better justify how we model d_i and how this impacts our results. The number of links of a node in a wired network is easy to define, while in wireless mesh networks this is not the case, and specific protocols are needed to cover the many possible cases [39]. A mesh protocol running on n_i generally maintains a set of nodes $\mathcal{L}_i$ with which it can potentially create links, and a set of neighbors $\mathcal{M}_i$. The former is the set of nodes that n_i received H messages from, the latter is the set of nodes that n_i has a bi-directional link with, and could be used as next hop in the routing table[g]. Some nodes that appear in $\mathcal{L}_i$ may not have a quality decent enough to become neighbors, so $\mathcal{M}_i \subseteq \mathcal{L}_i$. Node n_i will also be in the set $\mathcal{L}_j$ of some other node n_j. We call $\mathcal{R}l_i$ the reverse link set of n_i, that is, the set of nodes that can receive H packets from n_i: $\mathcal{R}l_i = \{n_j | n_i \in \mathcal{L}_j\}$.

We are interested in defining the resources occupation of a H packet. As a first approximation, we consider H all of the same size (we could easily add a dependency on the neighbor size in our formulation, but we actually believe that it is more important to model the number of packets instead of the their size, to keep the model simple). In principle, we could simply say that a H packet always occupies the same airtime, so the model could simply preserve the number of H packets generated per second in the whole network. Actually, we go further than that, and we consider that a H packet generated by n_i uses network resources that are proportional to $||\mathcal{R}l_i||$, because it actually keeps occupied a number of radios given by $||\mathcal{R}l_i||$. As links may be asymmetric, n_i does not know $\mathcal{R}l_i$, and we approximate it with $\mathcal{M}_i$. A better choice in the real world would be to use $\mathcal{L}_i$, but in Mininet we don't have this information and thus we set $d_i = ||\mathcal{M}_i||$. Note that if we chose the much simpler optimization of preserving the total number of H per second, we could just set $d_i = 1$ (or to some other constant) and achieve a higher gain.

[g]Actually, depending on the protocol implementation, these lists may refer to nodes, or IP addresses of network interface, or any other suitable identifier, but we use the generic term 'node' for simplicity.