RoRoute: Tools to Experiment with Routing Protocols in WMNs

Michele Segata, Nicolò Facchi, Leonardo Maccari, Renato Lo Cigno
Dept. of Information Engineering and Computer Science, University of Trento, Italy
{name.surname}@unitn.it

***Abstract*—Routing in wireless mesh networks is a well studied topic, but indeed, a deeper analysis in real world deployments shows that there are still many open issues, the real world is complex enough so that theoretic design, simulations and emulators can hardly encode it. This paper presents a set of tools and best practices we developed thanks to the WiSHFUL[1] experimental platform that enables the analysis of routing protocols robustness and resilience in face of failures and other impairments when the protocols run in real networks with all the complexity and non-linear interactions that characterize wireless meshes. We present results obtained with Optimized Link State Routing (OLSR) and possible extensions to this protocol that clearly document phenomena that are not visible or implementable in either simulation or emulation environments.**

I. Introduction

Wireless mesh and ad-hoc networks have been largely studied in the last two decades and today they found niche applications in which they are successful. Two examples at the opposite extremes are "Community Networks" [1], [2] (networks made of hundreds of nodes covering entire cities) and the small, high-throughput home meshes. In between there are hoards of other applications where mesh networks have been historically used successfully (military, security, industrial, etc.).

This success was made possible mostly thanks to the research that improved the PHY and MAC layers (IEEE 802.11 in particular) that was not directly targeted at mesh networks, but greatly improved their capacity. Instead, we observe that the research targeting the upper layers (IP in particular) did not produce much impact. Of the hundreds competing proposals for the routing layer produced in the literature, most real networks use just a few of them, with little degree of innovation from classical Internet-based protocols [3].

Now that off the shelf products have links with more than gigabit capacity for less than 100$, we need new ideas to make networks scale from tens to hundreds to thousands of nodes, for instance minimizing signaling overhead [4] or finding better routing metrics [5], [6], or novel applications [7]. We also need to ensure that these novel ideas do not remain on paper and in simulators, as we know that the path that leads from a proposal to code running on real devices is full of obstacles.

This work has been supported by the H2020 GA No. 645274 "Wireless Software and Hardware platforms for Flexible and Unified radio and network controL (WiSHFUL)" with the project "Pop-Routing On WiSHFUL (POPROW)" financed in Open Call 3.

[1]WiSHFUL is a European H2020 project supporting an experimental distributed testbed for wireless networks http://www.wishful-project.eu/

Starting from this observation, this paper documents and analyses the difficulties that can be encountered in real-world experimentation of mesh networks and the solutions that can be put into practice. We describe our methodology and we show that it can be effectively used to design, implement and test layer 3 research proposals in a time-efficient manner. The tool implementing our methodology, i.e., RoRoute, is released as Open Source software[2] to the scientific commnuity.

II. Background and Related Work

When proposing new routing strategies and protocols, we need to compare them to existing approaches to understand when they perform better or worse. As an example, in previous work we proposed an optimization of the Optimized Link State Routing (OLSR) protocol [8] that reduces routing convergence time when central nodes fail [9], and we have shown it can be efficiently implemented and emulated [10], but will the proposal stand the test of reality? Throughout this paper, we consider this example as use case to showcase the features of RoRoute and how it effectively helps in comparing standard OLSR against the proposed variant: Pop-Routing [9]. As the goal of Pop-Routing is increasing network resilience against failures, thus often in the paper we use "killing a node" as synonym of topology change.

As we use OLSR to exemplify the experiments that can be done with RoRoute, we recall some basic concepts. The reader can find more details in the RFCs that standardize the first [8] and the second version of OLSR [11] and in the relevant literature on its features as convergence speed, Multi-Point-Relays (MPRs) selection, or Fish Eye [4], [12], [13]. OLSR is, as the name states, a link-state routing protocol. OLSR nodes periodically send an `H` message to announce its presence to neighbors. `H` messages from node i contain the IP address of the 1-hop neighbors, so that at steady state each node has a full knowledge of its 2-hop neighborhood. MPRs can be elected by nodes to form a "signaling backbone" that keeps the protocol overhead to acceptable levels when the number of nodes increases. To simplify the tests, but without loss of generality (as Pop-Routing can be used also with MPRs) we assume that every node periodically generates a `TC` message containing the list of its destinations and re-broadcast the `TC` messages it receives from other nodes. `TC` messages distribute

[2]https://github.com/AdvancedNetworkingSystems/roroute

ISBN 978-3-903176-02-7 © 2018 IFIP

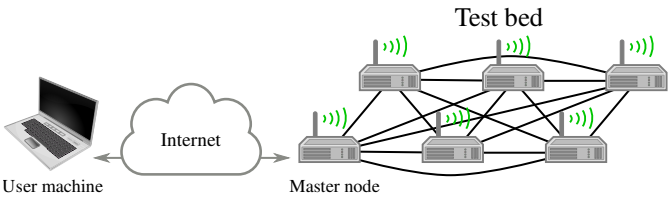


Figure 1: Experimental network setup and remote control.

the link state so that OLSR can compute the routing with Dijkstra algorithm.

What are instead fundamental to understand experimental results presented in Section IV are the H and TC timers: $t_{\mathtt{H}}$ and $t_{\mathtt{TC}}$ respectively. In the standard OLSR implementations these timers are constant and equal in all nodes, normally with $t_{\mathtt{H}} < t_{\mathtt{TC}}$. H and TC messages have a limited validity period, which is computed by multiplying their timer values by the $\mathtt{H}_{\mathtt{ml}}$ and $\mathtt{TC}_{\mathtt{ml}}$ multipliers, respectively. If a specific H or TC message is not received within this time frame, routes advertised through the missing node or link are invalidated, causing OLSR to change its perception of the topology and update the routing. As we have shown in [9] by proposing Pop-Routing, these timers can be optimized with different values in each node, depending on the node betweenness centrality, leaving the total overhead of the protocol constant, but significantly increasing the resilience of the networks in face of node failures.

Finding means to compare standard OLSR with Pop-Routing had been challenging. The first and quickest approach are network simulators like NS-3 or OMNeT++. They are powerful means but do not run in real time and the source code used for tests can not be reused as-is in real networks. The second approach is given by network emulators, on which one can test deployable code, since they set-up a virtual infrastructure that runs the same operating system and the same software of the target environment. Emulators like NEmu and Naxim, or ligtweight emulators like NePA TesT [14] (built on top of Mininet) make it possible to re-use the code in real devices, but do not allow to emulate large networks with realistic MAC and PHY levels.

The third approach is to use real testbeds like Community-lab and WiSHFUL[3], which allow access to remote network resources. Testbeds provide researchers with the highest degree of realism, as they run the real software on real hardware, on the other hand, it is difficult to control the test environment, making it harder to obtain reproducible results.

For this reason, this work proposes RoRoute, a set of tools that automatizes the majority of the steps required for evaluating the performance and the robustness of routing protocols in real testbeds.

III. RoRoute Workflow

The goal of RoRoute is enabling us to perform the experimental analysis of different OLSR versions in terms of network convergence time. To this goal, it must perform node and topology setup, run the experiment, as well as logging and retrieving the data. As the final and most important step, it must process the data and compare the different approaches, showing which one has the fastest convergence.

[3]https://community-lab.net/; http://www.wishful-project.eu/testbeds

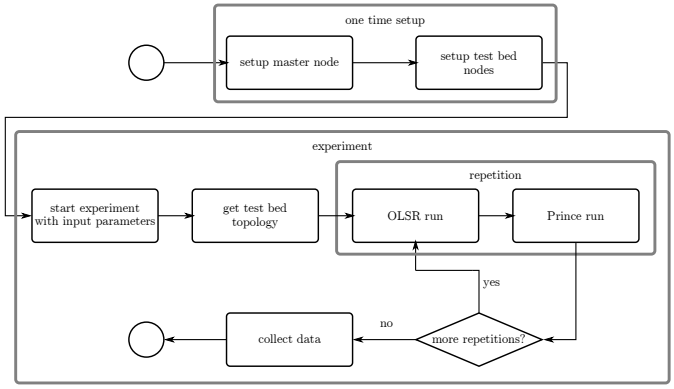


Figure 2: Flow chart of a RoRoute experiment with standard OLSR and Pop-Routing.

Fig. 1 shows the testbed network from a high-level perspective. Experiments are remotely controlled from the user machine, which delegates the job to one testbed node, called the master node. The user opens a single SSH connection to the master node which, in turn, connects to the nodes through the testbed Local Area Network (LAN). The master node is responsible for controlling the experiment by sending instructions to the nodes. For the setup phase (i.e., installing the required software) the master node exploits Ansible, a tool that enables the automation of tasks on clusters of nodes.

The control of the experiment, instead, is done through WiSHFUL, an open source framework permitting the management of radio and network devices with a set of Unified Programming Interfaces (UPIs). Through WiSHFUL, the master node (a.k.a. the controller) can automatically discover testbed nodes (a.k.a. the agents) and perform UPI calls on all or on a subset of them. For example, there exist UPIs for setting physical layer parameters (transmission power, bit rate), for managing routing tables, retrieve information from nodes, running code snippets on them, etc.

Fig. 2 shows the flow diagram of a single experiment. The first task is to realize a specific topology on top of the testbed one. To this aim RoRoute exploits OLSR to obtain the testbed topology and to realize the desired one via MAC layer filtering. In general, the *subgraph isomorphism problem* is $\mathcal{NP}$-complete [15], but solving it when the host graph (i.e., the testbed topology) is almost a full-mesh is trivial.

Once MAC filter rules are in place, RoRoute starts the OLSR daemon and waits for network convergence. To this purpose the master node periodically checks that the routing table and the topology computed by OLSR does not change for a predefined amount of time. Once convergence is reached, RoRoute starts the experiment.

Depending on the topology and the chosen strategy, which can be user-defined, the master computes the nodes that should be killed to test resilience and when, or it can be extended to change topology in other ways. The master defines and

ISBN 978-3-903176-02-7 © 2018 IFIP

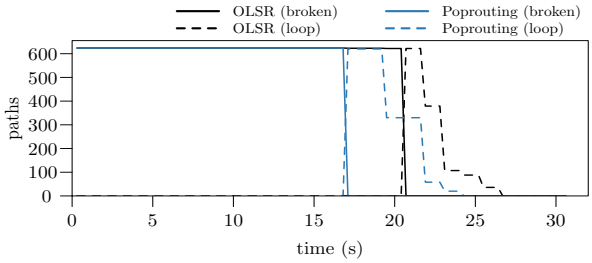


Figure 3: Typical RoRoute repetition outcome comparing standard OLSR and the Pop-Routing enhanced version.

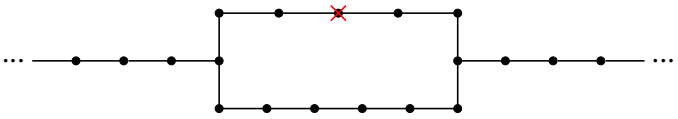


Figure 4: Line topology used for preliminary test experiments.

distributes the absolute experiment start time, all nodes are synchronized with Network Time Protocol (NTP), and this is key to the analysis phase.

RoRoute considers the experiment terminated after a predefined amount of time. This procedure is repeated for all variants of the protocol, and for the required number of repetitions. The final step is data collection. Each node compresses the data and sends its to the master node which, in turn, sends it to the user machine.

A. Analyzing Network Evolution

RoRoute takes care of the post-processing phase, which in our case consists in the analysis of the routing tables to count disrupted paths due to node failure. Disrupted routes include broken and looping paths. Broken paths are those where the routing table of one of the nodes along the route either points to the failed node or does not have an entry for the destination. Paths with loops, instead, are the ones where one node appears twice in the route, causing packets to bounce back and forth until the Time to Live (TTL) expires.

From the point of view of performance, it is important to measure how much time a protocol needs to recover broken paths. For each time sample, RoRoute computes the number of broken paths starting from the routing tables, which are stored in a dictionary $R_t^i[\cdot]$ where i is the node and t the time sample. For each timestamp h, RoRoute navigates the graph from every source n_j to every destination n_k recursively, using the routing tables of intermediate nodes. For each h, it counts the broken routes r_h (i.e., those that still include the failed node or that contain loops). This produces an array $\{(T_0, r_0), (T_1, r_1), \ldots\}$, where each entry associates a time instant to the number of failures.

The time/failures vector can be plotted for a qualitative analysis. Fig. 3 shows a typical RoRoute outcome comparing standard OLSR against its Pop-Routing enhanced version on the topology shown in Fig. 4. The topology is composed by two long chains connected by two paths, one being shorter than the other (and thus, preferable). The failing node is the one in the middle of the shortest path. Fig. 3 compares two single-shots only. To obtain a quantitative view, we can compute the integral of the time/failures vector for each OLSR version and average it over multiple repetitions for statistical confidence.

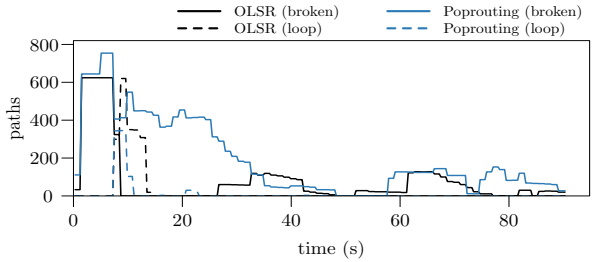


Figure 5: Non converging topology due to small $\texttt{H}_{\texttt{ml}}$ and $\texttt{TC}_{\texttt{ml}}$ values.

IV. Sample Results

The main goal of our experiments is to measure the gain in performance obtained by tuning the generation frequency of OLSR timers according to nodes' centrality. Before reaching this goal we needed to properly configure OLSR and RoRoute to obtain correct and meaningful results.

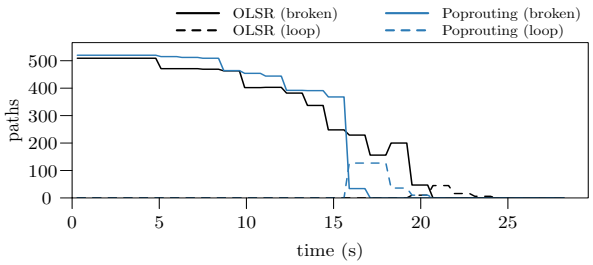
As a first step we need to choose values for the $\texttt{H}_{\texttt{ml}}$ and the $\texttt{TC}_{\texttt{ml}}$ parameters. At the same time, these parameters affect the reaction time to network changes, as well as the correct convergence of the protocol. Fig. 5 shows an example experiment outcome where $\texttt{H}_{\texttt{ml}}$ and $\texttt{TC}_{\texttt{ml}}$ are too small. In this experiment, we use the topology in Fig. 4 and set both $\texttt{H}_{\texttt{ml}}$ and $\texttt{TC}_{\texttt{ml}}$ to 3, the value recommended by the standard, but not the default in olsrd.

Too small values of $\texttt{H}_{\texttt{ml}}$ and $\texttt{TC}_{\texttt{ml}}$ combined with the high chance of loosing control packets in the long paths of the given topology, cause OLSR to invalidate routes by mistake. By observing Fig. 5 the problem was easy to diagnose and to fix.

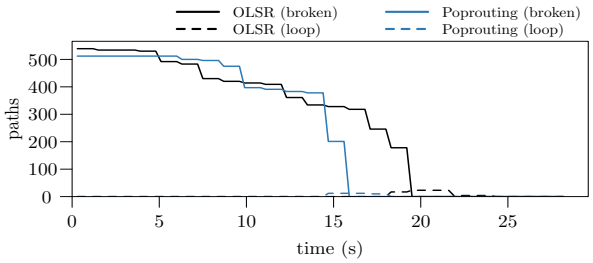
An additional problem we encountered was "routing randomness", which is only observable in a testbed or in real deployments. RoRoute correctly implements the experiment topology, but still there is no guarantee on which shortest paths the routing algorithm will compute. For example, in the case of multiple shortest paths between a pair of nodes, we cannot know which one will be chosen. This is a problem when we need to perform a comparison between different algorithm implementations, as the different experiment instances might compute different shortest paths. To minimize this problem, RoRoute can compute and assign random link quality multipliers to the edges, reducing the chances of having multiple shortest paths.

Fig. 6 shows this phenomenon. The figure shows the evolution of two repetitions. First, it can be seen that the two protocol versions being compared do not converge to the same topology, as killing the same nodes results in a different number of broken paths. Second, the behavior changes between different repetitions. In Fig. 6a, the node failure causes more broken paths in the Pop-Routing run, while in Fig. 6b happens exactly

ISBN 978-3-903176-02-7 © 2018 IFIP



(a) first repetition



(b) second repetition

Figure 6: Route disruption for different repetitions within one experiment.

the opposite. The random link quality multipliers generated by RoRoute, however, minimize the discrepancy, so that different protocol variants can still be compared by performing multiple repetitions and averaging the results. As in the case of the timers, RoRoute eased the identification of the problem.

To quantify the amount of variability we experience over a set of repetitions, we report here some statistics of a comparison experiment between OLSR and Pop-Routing using topologies generated with the well known "caveman" generator. In this experiment we run 7 repetitions for each OLSR variant, obtaining 14 topologies to compare. Table I shows the number of routes which differ between one repetition taken as a reference and all the others. More formally, let S_i be the set of all shortest paths of repetition i, with S_0 being the reference repetition. We define the set of shortest paths that differ between experiment i and experiment 0 as

$$d_i = \{\{u_i, \ldots, v_i\} \mid \{u_i, \ldots, v_i\} \in S_i \neq \{u_0, \ldots, v_0\} \in S_0 \wedge u_i = u_0 \wedge v_i = v_0 \wedge (|\{u_i, \ldots, v_i\}| > 2 \vee |\{u_0, \ldots, v_0\}| > 2)\}.$$

Table I shows the absolute count of differing paths $|d_i|$, as well as the relative percentage with respect to all the shortest paths, excluding self-loops and point-to-point links. Even considering the random link quality multipliers, we experience up to 18 % different shortest paths between different repetitions. On average, the difference is around 9.58 %, which shows how challenging it is to perform comparable repetitions in a real testbed, and thus how beneficial can be the use of RoRoute.

V. Conclusions

In this paper we presented RoRoute, a set of tools that automate the majority of the steps required for evaluating the performance of different OLSR implementations in terms convergence time. RoRoute eases resource discovery and reservation on Fed4FIRE compliant testbeds, it simplifies software deployment on network nodes, experiment execution, data collection, and processing. Although its use case is very specific, we believe it can still be a valuable tool for the community as we are currently planning to extend it to support generic routing protocols.

run	1	2	3	4	5	6	7	8	9	10	11	12	13
$\|d_i\|$	72	62	54	84	88	230	242	74	84	156	200	160	156
%	5	4	4	6	6	17	18	5	6	11	14	11	11

Table I: Shortest paths statistics computed over 14 repetitions, taking the first one as reference. The average difference is 9.58 %, while the total number of shortest paths is 1334.

References

[1] 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, Jan. 2015.

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

[3] J. Avonts, B. Braem, and C. Blondia, "A questionnaire based examination of community networks," in *2013 IEEE 9th Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct. 2013, pp. 8–15.

[4] L. Maccari and R. Lo Cigno, "How to Reduce and Stabilize MPR sets in OLSR networks," in *8th IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Barcelona, Spain, Oct. 2012, pp. 381–388.

[5] C. Barz, C. Fuchs, J. Kirchhoff, J. Niewiejska, and H. Rogge, "OLSRv2 for Community Networks: Using Directional Airtime Metric with external radios," *Computer Networks*, vol. 93, Part 2, pp. 324–341, Dec. 2015.

[6] J. Tremback and J. Kilpatrick, "Althea, an incentivized mesh network protocol," May 2017. [Online]. Available: http://altheamesh.com/

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

[8] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626, Internet Engineering Task Force, Oct. 2003.

[9] 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.

[10] 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.

[11] C. Dearlove and T. Clausen, "Routing Multipoint Relay Optimization for the Optimized Link State Routing Protocol Version 2 (OLSRv2)," RFC 7187, Internet Engineering Task Force, Apr. 2014.

[12] 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, March 2012.

[13] 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.

[14] L. Baldesi and L. Maccari, "NePA TesT: Network Protocol and Application Testing Toolchain for Community Networks," in *12th IEEE/IFIP Conf. on Wireless On demand Network Systems and Services (WONS)*, Jan. 2016, pp. 88–95.

[15] S. A. Cook, "The Complexity of Theorem-Proving Procedures," in *3rd ACM Symposium on Theory of Computing*, May 1971, pp. 151–158.