Evaluation of mesh routing protocols for wireless community networks

Axel Neumann[a], Ester López[b], Leandro Navarro[b]

[a]*Routek SL*
[b]*Universitat Politècnica de Catalunya*

Abstract

In recent years, we have witnessed the exponential growth of wireless community networks as a response to the clear necessity of Internet access for participation in society. For wireless mesh networks that can scale to up to thousands of nodes, which are owned and managed in a decentralized way, it is imperative for their survival to provide the network with self-management mechanisms that reduce the requirements of human intervention and technological knowledge in the operation of a community network. In this paper, we focus on one important self-management mechanism, routing, and we study the scalability, performance, and stability of three proactive mesh routing protocols: BMX6, OLSR, and Babel. We study different metrics on an emulation framework and on the W-ILab.T testbed at iMinds, making the most of the two worlds. Emulation allows us to have more control over the topology and more systematically repeat the experiments, whereas a testbed provides a realistic wireless medium and more reliable measurements, especially in terms of interference and CPU consumption. Results show the relative merits, costs, and limitations of the three protocols.

Keywords: mesh routing, wireless community networks, BMX6, OSLR, Babel

1. Introduction

Wireless community networks present an alternative ownership model for IP-networks, where every piece of equipment is managed and owned in a decentralized fashion by members of the community, and traffic is routed cooperatively. These were motivated to overcome the gap between Internet access and the connectivity offered by traditional ISPs where coverage is far from ideal, especially in rural areas. As time has gone by and as Internet access has become increasingly important in individual and collective participation in society, wireless

Email addresses: `axel@routek.net` (Axel Neumann), `esterl@ac.upc.edu` (Ester López), `leandro@ac.upc.edu` (Leandro Navarro)

community networks have been growing exponentially in many countries, some of them reaching thousands of nodes (e.g., Guifi.net [1] and Athens Wireless Metropolitan Network (AWMN) [2]).

From a scalability point of view, this exponential growth must come with the proper set of tools and mechanisms that render the network as autonomous as possible, so that the management and control actions required are reduced to a minimum. However, not only is minimizing human effort needed, but also making sure that those tools, mechanisms, and software that bind the network together are adequately scalable and can support network growth.

This paper is an extension of our previous work presented in [3], [4], and [5], which focused on the scalability and stability of proactive routing protocols, one of the key building blocks for operating such networks. In this paper we expand previous knowledge by analyzing the performance of OLSR[6], BMX6[7] and Babel[8], three common routing protocols in wireless community networks, under different networks' sizes and characteristics, using both an emulation framework and the W-ILab.T testbed. These routing protocols have been characterized by means of studying their control overhead, convergence delay, CPU and memory consumption and stability.

The rest of the paper is structured as follows. Section 2 details similar work that has been previously done and Section 3 explains in detail the three routing protocols that are compared. Section 4 discusses the different metrics that characterize the routing protocols and then we present our emulation- and testbed-based experiments and results in Sections 5 and 6. We discuss the results as a whole in Section 7, and we conclude in Section 8.

2. Related Work

Ad hoc network routing protocols have been extensively studied in the literature; however, most of the work done focuses on mobile ad hoc networks. The performance of routing protocols is expected to be different in wireless mesh networks, where the backbone mesh nodes are static and do not have energy constraints.

Among the three protocols considered, OLSR has received more attention: its protocol overhead, route convergence time, delivery ratio, end-to-end delay, and throughput have been compared in simulations with those of AODV, HWMP, DSR, TORA and DSDV ([9, 10, 11, 12, 13, 14]). There are also some real testbed experiments that compare OLSR with BATMAN([15, 16]) or AODV([17, 18, 19]) in networks that range up to 49 nodes and some provide additional information such as CPU and memory consumption.

The results in these studies demonstrate that compared to on-demand routing protocols, OLSR has comparable results in terms of delivery ratio, while the end-to-end delay is lower, and the overhead is lower in dense networks with many flows and higher in the contrary situation. The OLSR protocol has been also compared with other proactive routing protocols, such as Babel and BATMAN in [20] and [15], showing that distance-vector protocols have lower overhead but do not necessarily achieve higher throughput.

The authors of [19] experimentally quantified and discussed the performance of OLSR, Babel, and AODV implementations based on a small (7-nodes) indoors testbed. Results are ranked using Kiviat diagrams to balance between the studied measures of overhead, energy consumption, packet reordering, delay, and loss. Although the obtained standard deviations for each measure indicate a high statistical validity and allow a clear and non-overlapping ranking between the different protocols, the relevance of equivalently compared but obviously correlated measures of different importance seems misleading. Another shortcoming is the lack of plausible explanations for certain results, such as why the overhead of proactive routing protocols in an unchanging environment should depend on the distance (hops) between two communicating nodes or how application-traffic delay and loss measured between two nodes with only a single possible single-hop path between them could be affected by the routing protocol.

Regarding self-healing and convergence performance, while a tremendous amount of work has been employed in the last years on simulation-based research on the performance of mobile ad-hoc networking and mesh routing protocols regarding their performance [21, 22, 23, 24] and tuning [25, 26], much less insight hs been published based on experimental analysis using emulation [27, 20, 3] or testbeds [28, 29, 19], leaving the experimental performance evaluation of changing topology aspects on mesh routing protocols surprisingly unattended.

Despite this gap of profound experimental evaluation, pioneering projects have started to work on concrete solutions targeted for low-budget IEEE802.11 enabled devices, such as the Serval project [30], promising a communication anytime and anywhere even in the absence of phone towers and other supporting infrastructure. Another project, however, which does not focus on mobile routers, is the Village Telco project [31]. It is interesting to note that both of these projects have changed the underlying routing protocols since their existence due to critical performance issues found in live deployments.

In addition, it is worth mentioning the Battlemesh event [32]], which is an annual gathering of mesh-routing protocol developers with the objective to compare their protocols in different challenging environments and scenarios, including mobile scenarios. Unfortunately, due to the dynamics of these events, reproducible results obtained via systematically performed measurements have not yet been documented. However, measurement snapshots from past years [33], capturing the performance in semi-static and mobile scenarios suggest a huge gap between simulation-based and experimentation-based measurements.

This work presents an extension of our work in [3], [5], and [4], which compared with other studies, such as [20, 15, 19], evaluates the effect of network dynamics (e.g. link failures) and topology changes. Another relevant difference of our work is that results are based on the combination of various scenarios and topologies and are thus not tightly coupled to a particular deployment, which allows us to generalize results and derive expectations for deployments with different topological structures, sizes, or node densities.

3. Mesh Routing Protocols

Routing is a critical function in wireless mesh networks, since it decides the path any packet must follow to reach its destination. In a community network that grows organically, with several hops from the source to the gateway and where network management is not done by a single entity, but by many members of the community in a decentralized way, it is imperative that a routing protocol is able to continuously adapt to network changes. Routing protocols are usually classified as proactive or reactive, based on whether they learn routing paths proactively or just when needed (reactively). In [34], it can be seen that the vast majority of community networks use proactive routing protocols, since nodes do not have energy constraints. Additionally, proactive routing protocols are more efficient in terms of packet delay and outperform reactive protocols when the number of flows in the network increases [35].

One of the goals of our evaluation is to understand the consequences of choosing either a distance-vector or a link-state paradigm and how it affects scalability. Distance-vector routing protocols follow the Bellman-Ford algorithm, sharing only aggregated information about the path metrics, whereas link-state protocols share the whole view of the network, and the metric of every single link is known by every node. On the representation of link-state routing protocols, the choice is easy; OLSR is the most studied and used link-stated routing protocol. On the distance-vector side, we have chosen BMX6 and Babel, which have been extensively compared with other mesh routing protocols in Battlemesh [32] workshops. Babel has been chosen because it is a clear implementation of a distance-vector protocol and BMX6 because of its recent popularity in existing community network projects (e.g., Guifi.net/qMp [36] and Libre-mesh [37]). In addition, BMX6 uses the Secure Hash Algorithm (SHA) hashes instead of IP addresses as node identifiers and implements a number of features to reduce the protocol overhead while keeping the protocol as reactive as possible. The other famous Layer-2 BATMAN [38] has not been included in our evaluation because of the difficulty of comparing it with the Layer-3 protocols.

In the following subsections, we explain how these three routing protocols work by describing the mechanisms used for neighbor discovery (How does a node know other mesh nodes in range?) and topology dissemination (How does a node learn about routes to nodes that are not directly reachable?).

3.1. Babel

Babel is a proactive, distance-vector routing protocol based on the Bellman-Ford protocol [8]. Its main concern is to limit routing pathologies as routing loops or black holes, which it achieves using a proper feasibility condition and attaching a sequence number to routing updates.

Babel's feasibility condition determines which of the received routing updates should be considered and which should not; a routing update for a route is feasible only if its metric is smaller than any of the routing updates for the previously advertised route.

The sequence number attached to a routing update is generated by the destination node it announces and determines to which other routing updates the metric can be compared. Only information with the same sequence number is comparable.

Neighbor discovery. Babel nodes discover its neighborhood by exchanging two types of messages.

- *Hello* messages are sent to a multicast address by every Babel interface with a sequence number that is increased locally every time a new *Hello* message is sent. By listening to

Hello messages, a node not only discovers its neighboring nodes, but it also estimates the reception cost (*rxcost*) of that link. By default Babel sends a *Hello* message every four seconds.

- *I heard you (IHU)* messages are used to determine the bidirectionality of a link and share the *rxcost* with the neighboring node. The *IHU* messages are conceptually unicast; however, they are sent to a multicast address to avoid address resolution protocol (ARP) exchanges and to aggregate multiple messages in a single packet. They are also sent periodically, but usually not as often as a *Hello* messages; by default they are sent every 12 seconds.

Topology dissemination. In Babel, nodes discover far away nodes by sharing their routing table in route update messages.

- A *route update* message announces a route and its associated cost, and every Babel node sends a periodic update for every node it can reach to a multicast address. Additionally, when there is a significant change in the network topology, such as a route retraction or a significant change in the metric, an unscheduled route update is sent, so that periodic updates do not need to happen as often (by default every 16 seconds).

When a node receives an update, first of all it checks its feasibility, and if feasible, it computes the accumulated metric by combining the metric on the update message plus the cost of the link from where the update is received.

3.2. BMX6

The BMX6 protocol is also a proactive, destination-sequenced distance vector protocol whose main goal is to reduce the size of periodic messages to achieve low routing overhead while attaining high reactivity to network changes. The key concepts behind this are (i) using a stateful-compressed communication between neighbors and (ii) the context-specific propagation of local versus global and static versus dynamic information.

In a mesh network with flat addressing, reducing overhead using stateful communication translates largely to the use of compact (16 bit) local identifiers to refer to other nodes, since addresses (specially with IPv6) are very long. Therefore, every message sent by a node will use its own local identifiers instead of a global one, which has been previously shared.

On the categorization for information, static information refers to such addresses and other details about a node that are unlikely to change; those attributes are gathered together into the node's *description.* On the other hand, dynamic information refers mainly to link and path costs estimations. The global versus local separation determines which information is kept within the neighborhood and which is flooded through the network; local identifiers and link costs are kept locally, while path costs and node descriptions are shared globally.

Neighbor discovery. Neighbors are discovered in a similar fashion to Babel.

- *Hello* messages are sent to a multicast address periodically with a sequence number that is locally increased every time a new *Hello* message is generated. By default, *Hello* messages are sent every 0.5 seconds.

- *Report (RP)* messages are sent with the same interval as *Hello* messages and report the number of *Hello* messages received. By counting the number of *Hello* messages received from a node and knowing the number of *Hello* messages that a node has received, a node can compute both the transmission and reception costs of a link.

Topology dissemination. Routes to other nodes in the network in BMX6 are obtained as a result of the flooding of originator messages.

- *OriGinator Messages (OGM)* are sent periodically by every node (originator) to announce its presence and then re-sent if appropriate by any node that receives it. An *OGM* contains the sender's local identifier of the originator, a sequence number, and a metric that measures the cost of reaching it from the sender's perspective. When a node receives an *OGM*, it computes the cost of reaching the originator by combining the metric announced in the *OGM* with the cost of the sender's link; if this cost is smaller than the cost via any other neighbor, then the node will re-multicast the *OGM*, after updating it with its local identifier and the metric computed. By default, a node generates an *OGM* every five seconds.

Additionally, static information is shared on demand. When a node receives an *OGM* or a *Hello* message with an unknown local identifier, it will ask the sender for the node description's hash. This hash allows the node to determine whether the local identifier refers to any of the known nodes, and if it is not the case, then it will request the node's description and update its knowledge conccerning the network.

3.3. OLSR

In contrast, OLSR, as its name points out, is an optimized link-state routing protocol. The optimized part comes from the optimization on the flooding mechanism; only nodes selected as multi-point relays (MPR) retransmit the node's messages.

Topology dissemination. As any link-state routing protocol, OLSR provides every node in the network with a (partial) view of the whole topology by flooding the network with *Topology Control (TC)* messages.

- A *TC* message describes all the nodes that are reachable from the message creator, as well as the quality of the involved links in both directions. The TC messages are generated periodically by every node in the network and are then retransmitted unchanged throughout the network. By default, the implementation used in our experiments transmits a TC message every five seconds.

Neighbor discovery Neighborhood sensing is performed in OLSR by periodically sending *Hello* messages.

- A *Hello* message consists of a locally increased sequence number and the list of known links to the sender's neighbors as well as their quality and the quality from the neighbor's perspective. Link quality is computed as a function of the number of received *Hello* messages from that neighbor, while the quality from the neighbor's perspective is simply the quality reported in its *Hello* messages. *Hello* messages are sent to a multicast address periodically, by default every two seconds.

In this evaluation, we have used the popular implementation by olsrd.org, which implements three fundamental changes to the original RFC. First, the MPR optimization itself, originally designed for wired scenarios with loss-free links, is modified to require not only one but seven nodes to reach every two hop neighbor. Otherwise, when considering even the weakest detected link as a reliable resource for the dissemination of topology information, massive network instabilities must be expected. Second, to compensate for the overhead introduced by the increased MPR redundancy, the fish-eye extension has been introduced [39] where TC updates are exchanged less often between far away nodes than between nearby nodes. This is achieved by letting the originator of each TC message use different time to live (TTL) values with the consequence that only every second TC message propagates beyond the first hop. Third, the path metric used by the olsrd.org implementation is based on the expected transmit count (ETX) metric [40] which, compared to the originally proposed hop count metric, provides a better reflection of the real path cost for transmitting a packet via wireless links. Although controversy on the best parameterization of this protocol exists (such as the findings published by Johnson and Hancke in [41] on the performance of ETX and the hysteresis-based hop count metric), we decided to base our experiments on the defaults of this implementation because their current selection still represents a common ground that reflects the experience from its usage in several community networks over many years.

3.4. Summary

In essence, what differentiates these routing protocols is their topology dissemination mechanisms and how they solve and position themselves in the trade-off between convergence delay and overhead.

- In Babel, nodes only interact with their neighbors, sharing all the relevant information between them periodically. Routing updates are bigger because they contain the complete routing table, but are only shared locally. Overhead is reduced by retaining long intervals between periodic updates, while reactiveness is increased by sending unscheduled updates when the network changes considerably.
- Additionally, BMX6 floods small OGM messages through the network, but principally the information shared is the same as on Babel, except messages are split and triggered differently. Overhead is reduced by compacting periodic messages as much as possible using stateful communication between neighbors, whereas convergence delay is minimized by having a very frequent exchange of messages.
- Further, OLSR is a link-state protocol; therefore, during topology dissemination, information concerning every link is shared, instead of path-aggregated information. Overhead is reduced using the fish-eye extension; updates are shared more frequently with nearby nodes than with far away nodes, but to be adequately reactive, the time interval between updates is kept small.

Table 1: Periodic Messages

	Message	Size (B)	Interval	Messages/node
Babel	Hello	8	4 s	1
	IHU	16	12 s	$1 \times n$
	Update	$[\ 12, 28\ ]$	16 s	$1 \times route$
BMX6	Hello	$[\ 4, 6\ ]$	0.5 s	1
	RP	1	0.5 s	$1 \times n$
	OGM	4	5 s	1
OLSR	Hello	$[\ 28, 32\ ] + 20 \times n$	2 s	1
	TC	$28 + 20 \times n$	5 s	1

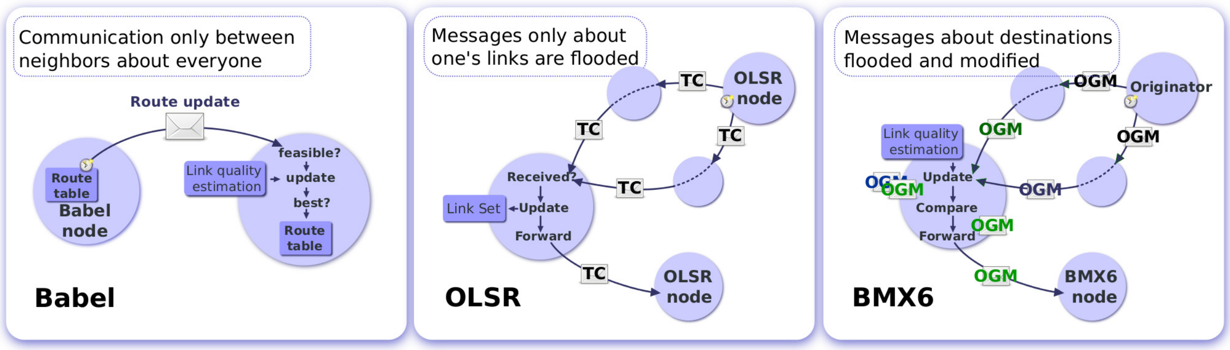


Figure 1: Topology dissemination mechanisms

Figure 1 summarizes the topology dissemination mechanism for each routing protocol. Table 1 lists the messages exchanged by each routing protocol, explaining its size and how many of them are exchanged depending on the number of neighbors (n) that a node has. The size given for each message type does not take into account the length of the headers.

4. Evaluation Metrics

There are several metrics to consider when evaluating the performance and overhead of a wireless mesh routing protocol.

The most common one is to measure its *network efficiency*, that is: how much routing traffic is necessary to be able to establish a connected network. Network efficiency is usually measured in terms of bytes/second or packets/second.

Given the relatively dynamic properties of wireless community networks we are interested in measuring how fast the routing protocol can adapt to these network changes or how stable the IP network is, given that the physical network is not stable. In terms of *stability*, we can measure the percentage of time the network is connected by pinging from some nodes to others or, similarly, measuring the longest time the network is not connected. From the perspective of reactivity, we can measure how long it takes to learn a new or better path, which is what we call *convergence time*. We are also interested in the cost in terms of the resource consumption of processing (CPU) and memory for each routing protocol. We must ensure the network nodes have the capabilities to run such protocols.

We measure the sensitivity of these metrics to the scale of the network, according to the variation of the size of the node neighborhood, variation of the total network size, variation of the length and number of hops of the network paths, and variation of the availability of links or the rate of changes in the network.

These measurements are performed and evaluated in different scenarios using container-based emulation and testbed-based experimentation. Both of them have different degrees of realism and flexibility. Emulation allows total control over environmental conditions, such as topology, availability, and changes but under limited realism. For instance, details taken from diverse real wireless community networks regarding the structure and events during a temporal period can be reproduced in a series of experiments and even be subjected to variations. In contrast, testbed experimentation provides a real environment, under stable environmental conditions, with a given set of nodes, radios, and locations, and control over a few aspects, such as transmit power and choice of nodes to use in an experiment among those available in the testbed.

5. Emulation Experiments

Using emulation, we can easily measure network overhead and convergence time. It is also possible to measure the cost in terms of memory, but the CPU cost is not reliable. Measuring the quality of the path is also complicated because the channels are not perfectly modeled. These metrics, therefore, are better studied in a testbed or real world deployment.

The emulation experiments presented in this paper represent a summary of the results obtained during several experiments using the same emulation system. Table 2 shows the network characteristics used for each figure. The Barcelonès area corresponds to a large portion of the city and metropolitan area of Barcelona, and its topology was retrieved from Guifi.net Community Network Mark Up Language (CNML). *Generator* refers to topologies obtained using the generator from [42] with parameters from the Osona county, a representative semi-rural area where Guifi.net started. Each specific experiment with a given set of parameter values was repeated at least 20 times.

Our emulation system is based on mesh Linux containers (MLC) [43], which is a set of scripts based on Linux containers (LXC) and Linux networking tools, such as *ip* or *tc*. The MLC runs an LXC container for each node in the network and establishes the desired connections between them with a given link quality using *tc*, so that packets are randomly dropped and delayed with probabilities as configured. The system does not allow applying complex Wi-Fi models or reflecting the impact of interference between links of any kind.

5.1. Network Overhead

In our first set of experiments, we studied the effect of network size on the network overhead of each routing protocol. The topology emulated in this case corresponds with a

Table 2: Network characteristics of the emulation experiments

	Topology	Number of nodes	Number of links
Figs. 3a, 4a	Barcelonès	{10,20, 30, 40, 50, 60, 66}	{9,21,31,43,54,65,72}
Figs. 3b, 4b	Barcelonès	50	54
Figs. 3c, 4c	Barcelonès	69	{75,100,150,250,500,750,1000}
Fig. 5a	Barcelonès	66	72
Figs. 5b, 5c	Generator	50	75
Fig. 6a	Generator	49	74
Fig. 6b	Generator	{16,25,49,64,81,100}	{24,38,74,96,122,150}

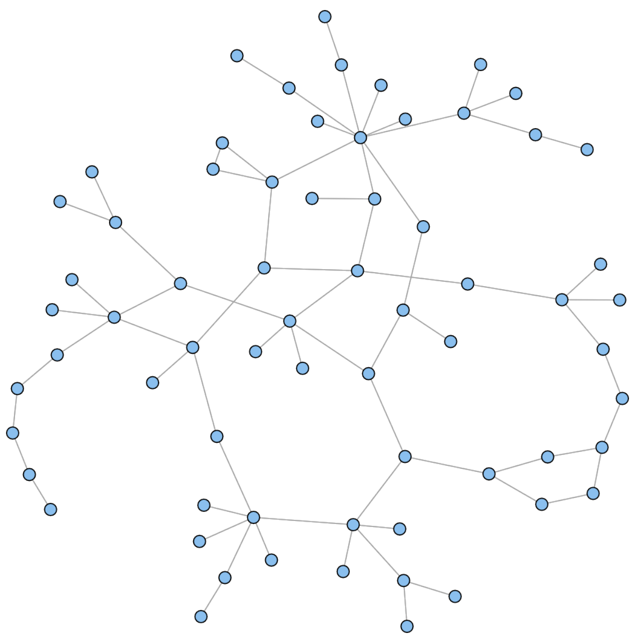
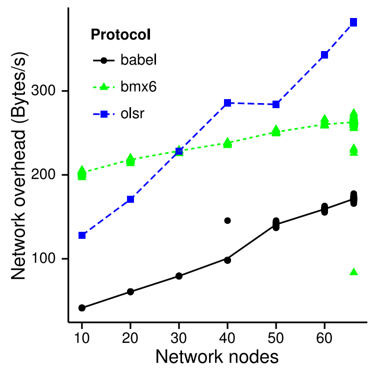


Figure 2: The topology of the Barcelonès network

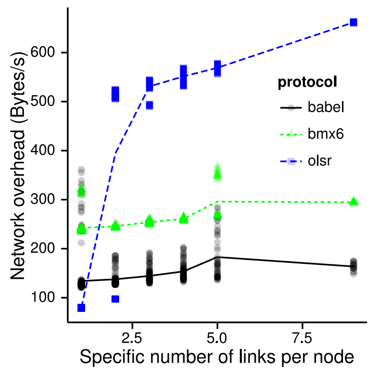
representation of the Barcelonès area of Guifi.net (shown in Figure 2 and more details in Table 2), where each link quality was determined by averaging the measurements of one hour [5].

Figures 3a and 4a illustrate the network overhead in bytes and packets of each routing protocol on networks with different numbers of nodes. To obtain those networks, the original network was randomly sampled. As we can see, the number of bytes increases with the number of nodes, and OLSR seems to be the more heavily influenced protocol, Babel always has lower overhead but seems to increase at a faster rate than BMX6. Regarding the number of packets, all routing protocols seem quite stable except Babel, which has a step increase on 50 nodes.

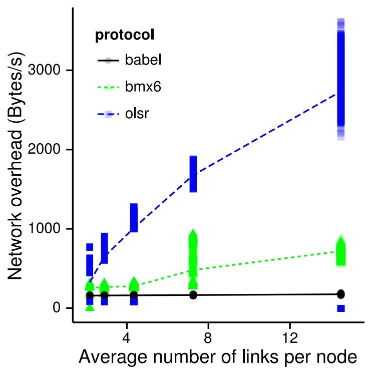
Then for Figures 3b and 4b, we run the experiment using the Barcelonès network without any modifications, and it shows the overhead of each node depending on the number of neighbors each node has. As we can see, OLSR is more heavily affected by the number of



(a) Bytes vs. nodes

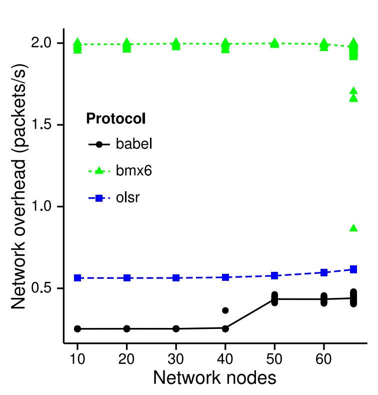


(b) Bytes vs. specific links

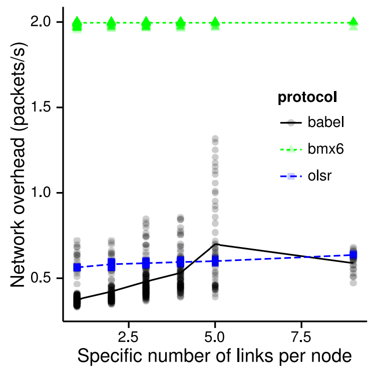


(c) Bytes vs. average links

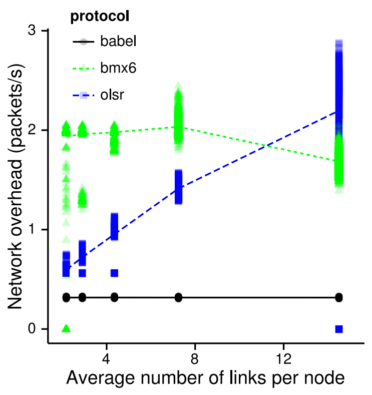
Figure 3: Network overhead in bytes



(a) Packets vs. nodes



(b) Packets vs. specific links



(c) Packets vs. average links

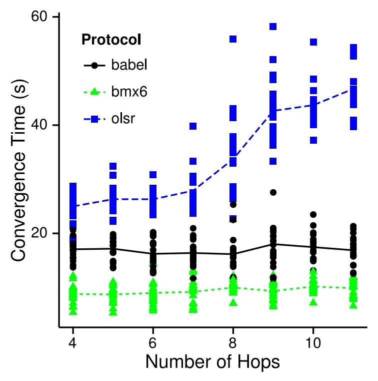
Figure 4: Network overhead in number of packets

neighbors, whereas Babel and BMX6's overhead in bytes only increases slightly with the number of neighbors. The number of packets looks stable in every case, except for a peak in Babel when there are five neighbors.

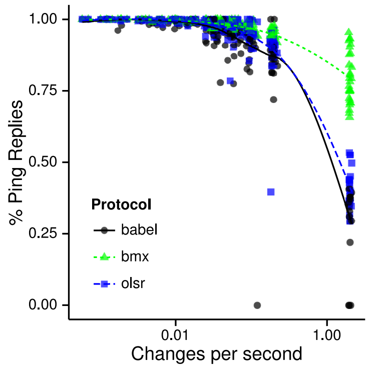
Finally, Figures 3c and 4c show the results when there is a fixed number of 69 nodes in the network, but the number of links is variable. As before, OLSR is the protocol that is more heavily affected by the number of links, whereas Babel remains stable in every case. Further, BMX6 increases slightly as the number of links increases. Because every point represents the overhead for a node, nodes with higher numbers of links within the same network will have higher overhead, and this difference is greater for OLSR than for BMX6 and Babel. The results for the number of packets are similar.

5.2. Stability and Reactivity

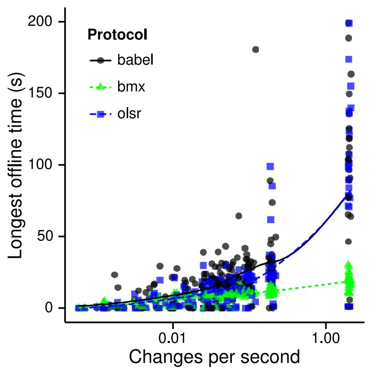
Our second set of experiments attempts to characterize the stability and reactivity of the three routing protocols.



(a) Convergence vs. hops



(b) Ping success vs. changes



(c) Offline period vs. changes

Figure 5: Network stability and reactivity

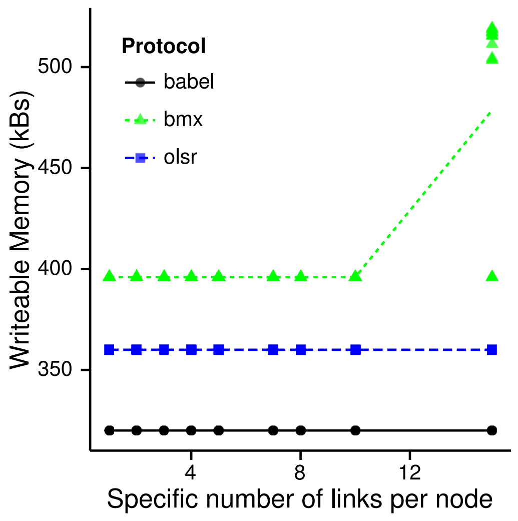
To measure the convergence time, we have looked for the longest path on the Barcelonès network. Then, we have measured how long it takes to discover new nodes that are attached to each of the nodes in the path from one of the endpoints. Figure 5a depicts our experiments' results. The OLSR obtains the worst performance, and we can clearly see a step function. This is due to the fish-eye extension [39], which sends link updates more frequently to nearby nodes, and not that frequently to far away nodes. Both BMX6 and Babel have flat responses, with BMX6 outperforming Babel.

In reference to measuring the stability, we have generated random network topologies using the community network generator presented in [42] with the parameters from the Osona zone in the Guifi.net [1] community network (degree=2.99, shape $\alpha = 0.2521602$, and rate $\beta = 0.01147359$). Then links of the network were turned on and off with different times between changes but always ensuring that the network remained connected. Figure 5b illustrates the success rate of a ping between two randomly selected nodes of the network, depending on the number of changes per second. As expected, more changes imply that the routing protocol cannot keep up, and connectivity is lost sometimes. In this case, BMX6 outperforms both Babel and OLSR, which present similar results. Figure 5c presents the same results regarding which routing protocol has longer offline periods.

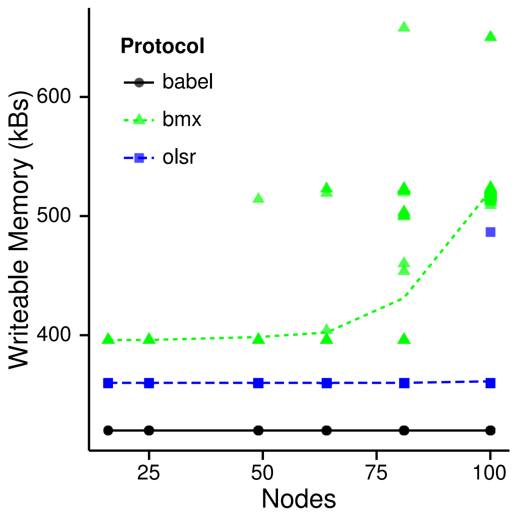
5.3. Memory Usage

The final metric studied through emulation is the cost in terms of memory. To measure the memory consumption, we have run the experiments on a computer with an Intel Core2 Duo E8400 processor running at 3.00 GHz with 4 GB of memory. We have retrieved statistics regarding memory usage using the *pmap* utility.

Our first experiment considers a network of 49 nodes and 74 links, and studies the memory consumed by each node based on the number of neighbors it has (Figure 6a). In this scenario, both OLSR and Babel show constant memory usage, independently of the number of neighbors; however, BMX6 requires more memory when the number of neighbors reaches 15. The results are similar in our next experiment (Figure 6b), which increases



(a) Memory vs. number of links



(b) Memory vs. number of nodes

Figure 6: Memory consumption

the size of the network (e.g., network with 16 nodes and 24 links, then 25 nodes and 38 links, etc.) and measures the average memory used by each node. Babel memory usage is stable, BMX6 increases with size and for OLSR, we observe an increase in one case for the biggest network. We believe that, for bigger networks, we would also see an increase in the memory use for OLSR and Babel, but this is not seen because memory is assigned in chunks, and, in contrast to BMX6, the OLSR and Babel protocols do not yet need the full memory provided by the current chunk. The assignment of memory in chunks also explains why memory usage of BMX6 seems non-linear. The measured usage remains unchanged until the last allocated memory chunk is exhausted and the probability for allocating the next chunk rises quickly and is non-linear in sections. The measured BMX6 memory usage in Figure 6a for 10 and 15 neighbors indicates such a section. In Figure 6b, the exponential growth in memory requirements for BMX6 is also caused by an increasing number of links that come with an increasing number of nodes.

6. Testbed Experiments

6.1. Experiment Setup

Protocol performance measurements based on real hardware have been performed in the W-ILab.T wireless testbed [44]. The facility consists of approximately 60 stationary and 15 mobile experimentation devices deployed in a 60 x 20 meter indoor location at iMinds. The grid-like deployment structure of stationary devices is principally given by six rows with 10 devices (columns) each and an inter row space of 3.6 meters and inter column space of six meters. The devices consist of of-the-shelf computer hardware, each equipped with two 802.11abgn WLAN cards, which can be freely programmed by the experimenters.

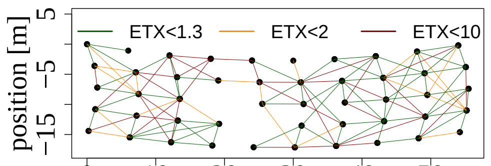
Further characteristics and configurations are summarized in Table 3. The devices, also called nodes, were configured to run a Linux operating system (OS) based on OpenWRT, a Linux distribution optimized for embedded wireless devices, and several convenient OS and measurement tools to control and collect measurement data. For the experiment, 50 nodes (the upper five node rows) have been used with one radio each configured in IEEE 802.11a ad-hoc mode. All protocols were configured to run on IPv6 and to announce only their primary interface addresses. Given the physically dense node deployment with an average neighbor distance of less than five meters, a number of preliminary measurements were performed to understand the wireless characteristics of the testbed and to avoid a fully connected mesh where the broadcast-based link detection mechanisms of the routing protocols detect links to nearly all other nodes (note that even the two most distant nodes are less than 63 meters away from each other) and achieve the establishment of true multi-hop topologies. The result of this exercise can be seen in Figure 7, showing topology snapshots (as detected by the OLSR protocol) resulting from different transmit power configurations with and without background traffic.

The average number of links per node is shown in Figure 8 for the number of nodes, transmit (TX) power, and TX rate grouped by their link quality and whether the links were captured with or without interference caused by TCP background traffic. It can be seen that the presence of TCP user traffic significantly decreases the perception of high-quality links (here those with an ETX rate of less than 1.3), while the total number of detected links is much less affected. The figure also shows that, in our testbed scenario, the average number of links per node scales linearly with the selected transmit power or rate. Figure 9a demonstrates the path length established by each routing protocol to route from the leftmost node in the second upper row (node Id 0) to the rightmost node in the same row (node Id 9). As expected, with a low transmit power (or high TX rate) and consequently small transmit range the selected end-to-end path relies on many intermediate hops, while less relaying nodes are required with an increased TX power (or a more robust but lower rate). Figures 9b and 9c give an impression on the end-to-end throughput achievable with each routing protocol when varying the node density by increasing power or rate.

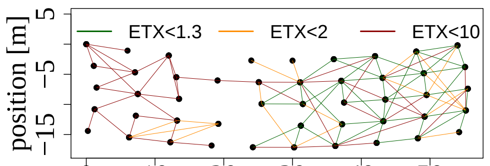
Based on these findings, our following experiments will be configured to use the parameters shown in Table 4. Primarily, we use 3 dBm of transmit power and disabled transmit rates below 36 Mbps to enforce the establishment of topologies with more than seven hops.

6.2. Measurements

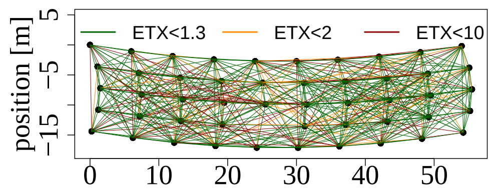
In order to measure the impact of neighbor size (density) and network size on cost and performance, all protocols have been sequentially exposed to a variety of testbed configurations. These testbed configurations were given by the range of parameterization values for each studied parameter and the default value for all other parameters. Each exposure (experiment) consists of the following standard procedure. First, all currently active protocols were disabled on all nodes. Then, the interfaces, IPv6 addresses, wireless settings (channel, mode, TX power, and enabled rates), and the currently probed protocol (only one at a time) were configured and activated only on those nodes relevant for this test. A stabilization period of 100 seconds was applied before continuing the actual measurement



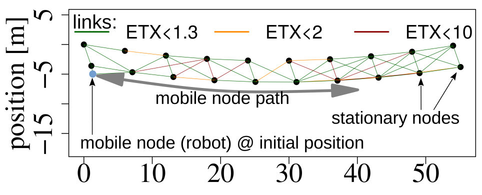
(a) 3dBm,36Mbps



(b) 3dBm,36Mbps with TCP user traffic

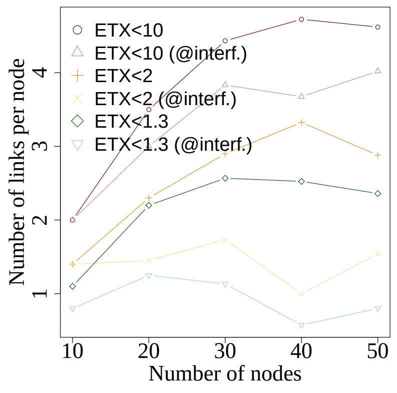


(c) 8dBm,36Mbps

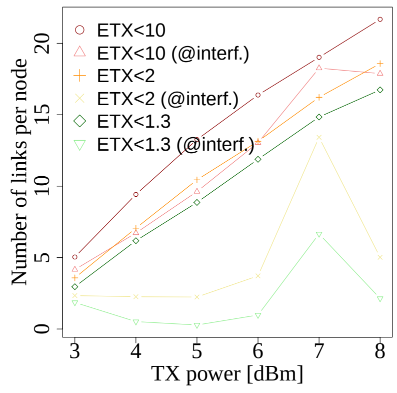


(d) mobile setup @4dBm,36Mbps

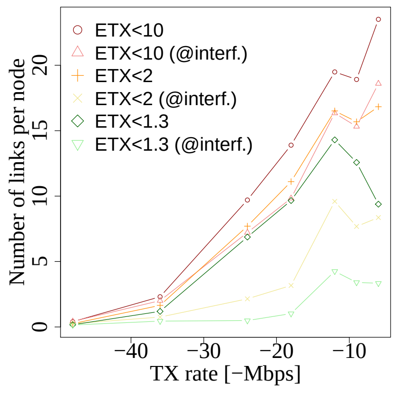
Figure 7: Topologies in different scenarios



(a) Links depending on number of nodes (3dBm,36Mbit)

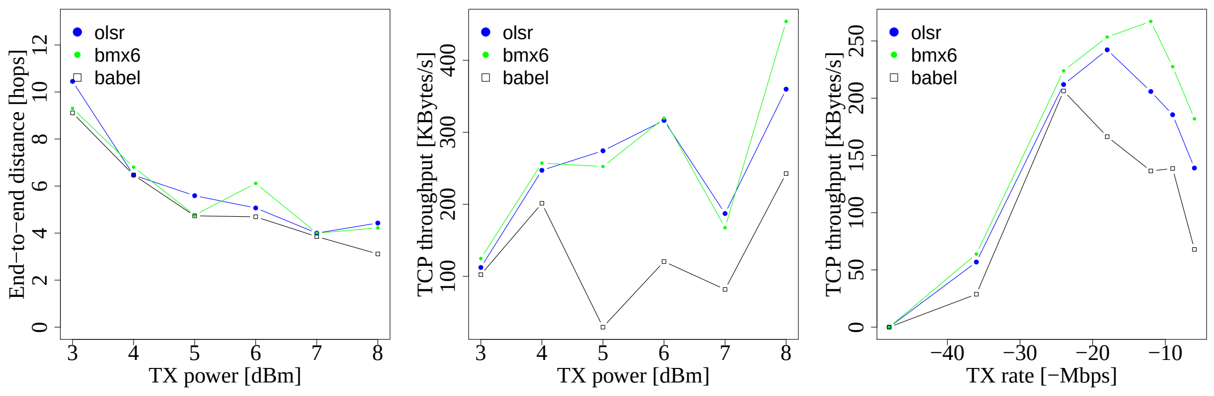


(b) Links depending on TX power (36Mbit)



(c) Links depending on TX rate (3dBm)

Figure 8: Network densities and link qualities for different testbed configurations



(a) Path length vs. TX power (36 Mbps) (b) Throughput vs. TX power (c) Throughput vs. TX rate

Figure 9: End-to-end path length and TCP throughput depending on network density varying power and rate

to avoid capturing of atypical bootstrapping effects. Each following measurement lasted 60 seconds and relied on common Linux tools (such as ping6, iperf, top, and tcpdump) for active probing of end-to-end path characteristics and monitoring and capturing CPU, memory, and traffic overhead.

After each experiment, the measurement data were offloaded and, once all experiments of a particular scenario were executed, post processed into graphs illustrating the dependency of one characteristic depending on a particular testbed parameter. Each measurement point in any of the graphs shown in Figures 8 to 15 represents the averaged relevant measurement data captured during a single experiment run.

The very limited exclusive testbed usage slots that allow interference-free experiments in the often overbooked W-ILab.T infrastructure do not allow the systematic repetition of scenarios required for a statistical validity analysis. Instead, the goal has been covering a wide range of selected parameters. However, atypical measurements have been selectively

Table 3: General W-ILab.T testbed and protocol characteristics and configuration

Characteristic	Configuration
Environment	Laboratory 16x60 meter
Deployment	Regular 5x10 nodes grid (see Fig. 7a)
Operating system	Linux/OpenWRT BarrierBreaker rev41558
Protocol impl.	babeld v1.5.0, bmx6 rev8b0585e8, olsrd v0.6.6.2
Hardware	ZOTAC NM10-ITX
CPU model	Intel(R) Atom(TM) CPU D525 @ 1.80GHz
Memory	903460 kB
Wireless	Atheros - AR928X 802.11a/b/g/n
Wireless mode	80211a, ad-hoc, channel 36 (5.18GHz)

repeated to avoid the consideration of exceptional outliers. This way, the resulting and purposely un-smoothed plots also include strongly varying behavior that is a typical characteristic of any real wireless network. Still, a number of protocol-typical characteristics and tendencies can be identified and generally acknowledge the findings made via our previous emulation-based measurements.

The results in Figures 11a, 12a, 13a, and 14a illustrate the impact of network size on overhead, CPU, and memory consumption and are based on the experiments of a single scenario where nodes, grouped in rows with 10 nodes each, were successively added to the total number of participating nodes, starting with the second row (which also contained source and destination node used for end-to-end path probing and optional transmission of TCP user traffic), then adding the first, third, fourth, and fifth rows until the final size of 50 nodes was reached, eventually yielding a topology as illustrated in Figure 7a. During this scenario, the power and minimum rate was fixed to 3 dBm and 36 Mbps.

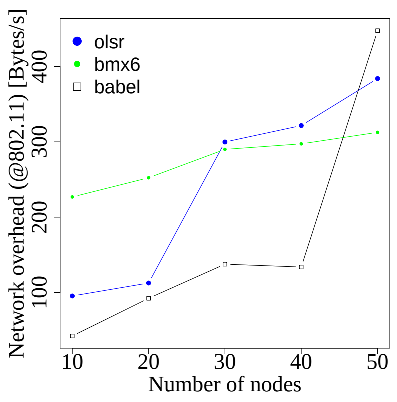
The impact of node density has been studied by either varying the transmit power of each node (with results shown in Figures 11b, 12b, 13b, and 14b) or by varying the minimal allowed transmit rate per node (see Figures 11c, 12c, 13c, and 14c). For these scenarios, all 50 nodes were used from the beginning, but transmit power or rate was successively changed in each experiment round.

The repetition of the above scenarios without background user traffic revealed that CPU and memory consumption do not significantly differ in both cases: thus, only the resulting impact on protocol overhead is shown in Figure 10.

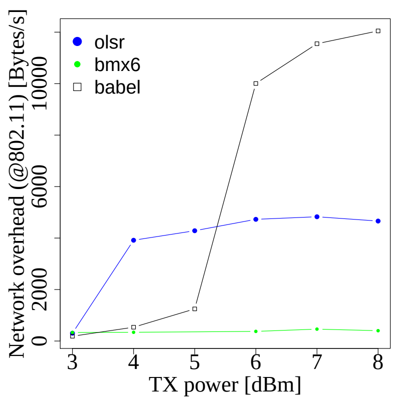
In order to obtain an experimentation-based picture of the self-healing capabilities of each protocol, we probed the end-to-end path between two nodes. This was realized using a 20-node subset (given by the upper two rows) of the original 50-node topology and an additional "mobile" node installed on a robot that was programmed to move with different speeds just below the second row of nodes in the W-ILab.T deployment from near the leftmost node column to the eighth column and back. Figure 7d illustrates this setup. The robot turning points were located 42 meters (or seven inter-column spaces) apart from each other. Path health to this moving node was probed using the ping6 command from the fourth node of the second row, located in the middle between the turning points. Figure 15a shows, based on a TX power and rate setting of 3 dBm and 36 Mbps, that for each protocol the average ping success rate to this moving node depending on its velocity, each reflects a scenario with differently fast changing links. It can be seen how the success rate, around 90% at velocities

Table 4: Experimented parameters, defaults, and ranges

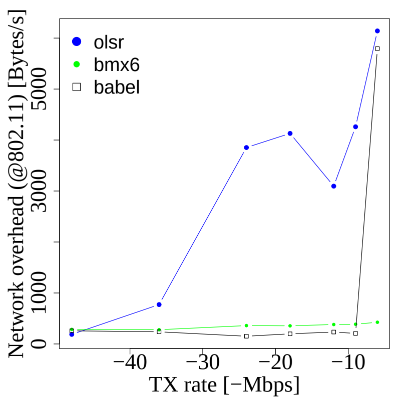
Parameter	Default	Range
Transmit power [dBm]	3	3, 4, 5, 6, 7, 8
Transmit rate [Mbit]	36	6, 9, 12, 18, 24, 36, 48, 54
Used nodes	50	10, 20, 30, 40, 50
Network Load	active	none versus active (single TCP stream between most distant nodes of a row)



(a) Impact of Network size

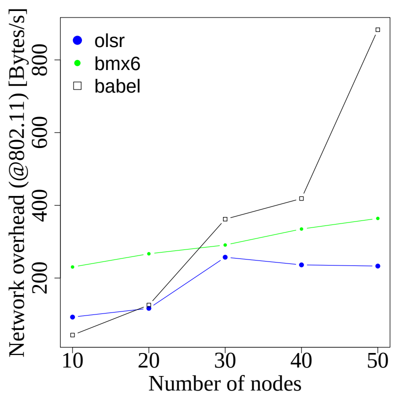


(b) Impact of TX power

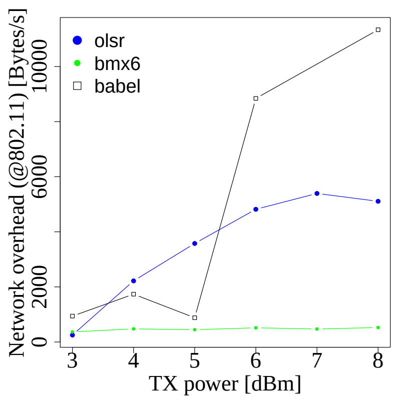


(c) Impact of TX rate

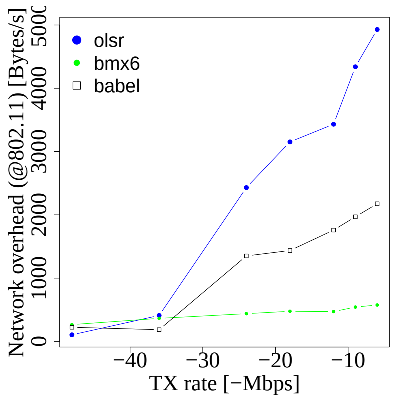
Figure 10: Data overhead (bytes/s) depending on network size and density (power, rate), **no TCP user traffic**



(a) Impact of Network size



(b) Impact of TX power



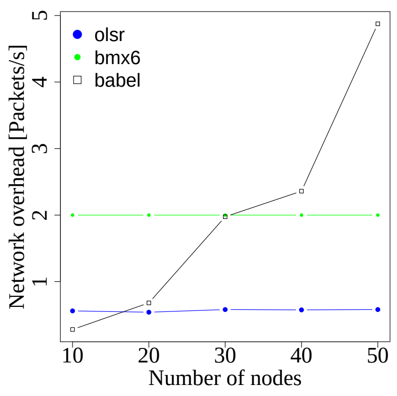
(c) Impact of TX rate

Figure 11: Data overhead (bytes/s) depending on network size and density (power, rate)

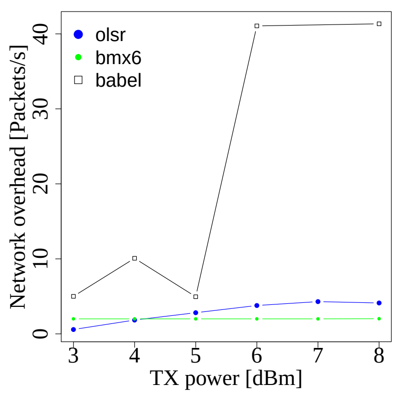
of 5 cm per second for all protocols, decreases with increasing destination velocities down to around 60% for BMX6 and below 40% for OLSR and Babel. The repetition of this scenario with slightly increased power settings at 4 and 5 dBm (see Figure 15b and 15c) lead to a broader continuous coverage of the moving node, giving routing protocols more time to adapt to weakening links and narrow the performance gap between the three protocols.

7. Discussion and Summary of Results

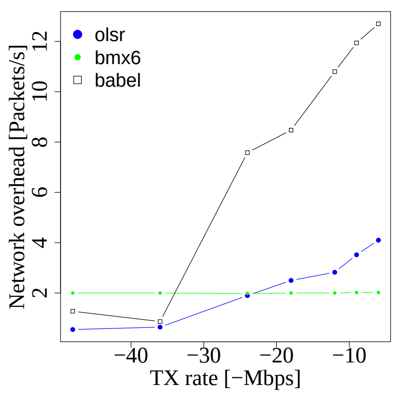
In this section, we take a closer look at the measurement results obtained via emulation and experimentation with the objective to gain a general understanding of how the characteristics of a network influence the performance of various routing protocols. We also discuss potential discrepancies of the two different methodologies. Table 5 provides a high-level summary of the identified protocol-specific behaviors by grouping the studied performance characteristics (in terms of overhead and self-healing capability) and dependencies



(a) Impact of Network size

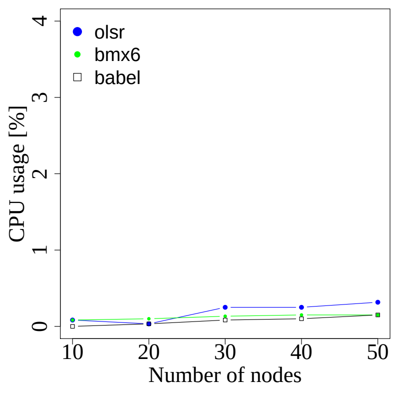


(b) Impact of TX power

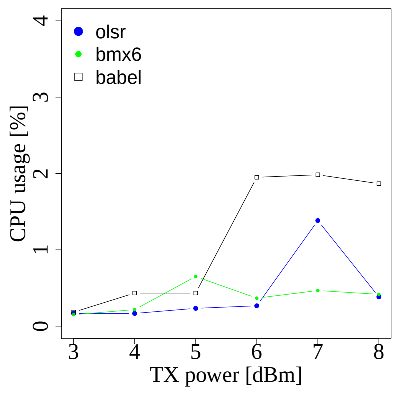


(c) Impact of TX rate

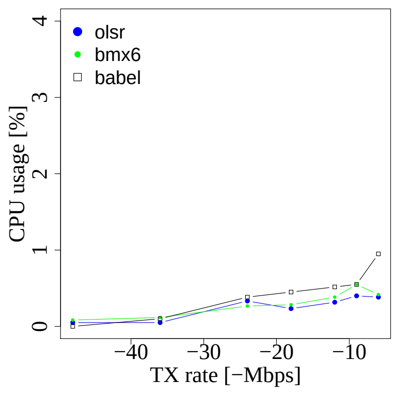
Figure 12: Data overhead (packets/s) depending on network size and density (power, rate)



(a) Impact of Network size

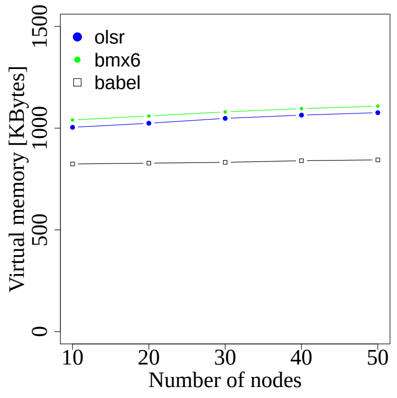


(b) Impact of TX power

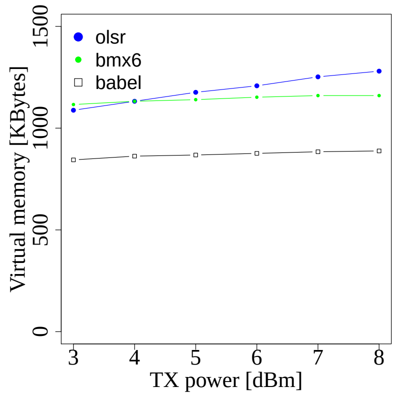


(c) Impact of TX rate

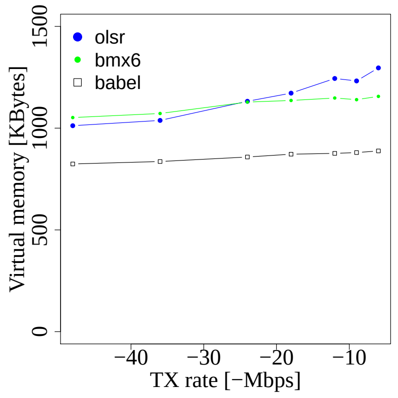
Figure 13: CPU consumption depending on network size and density (power, rate)



(a) Impact of Network size

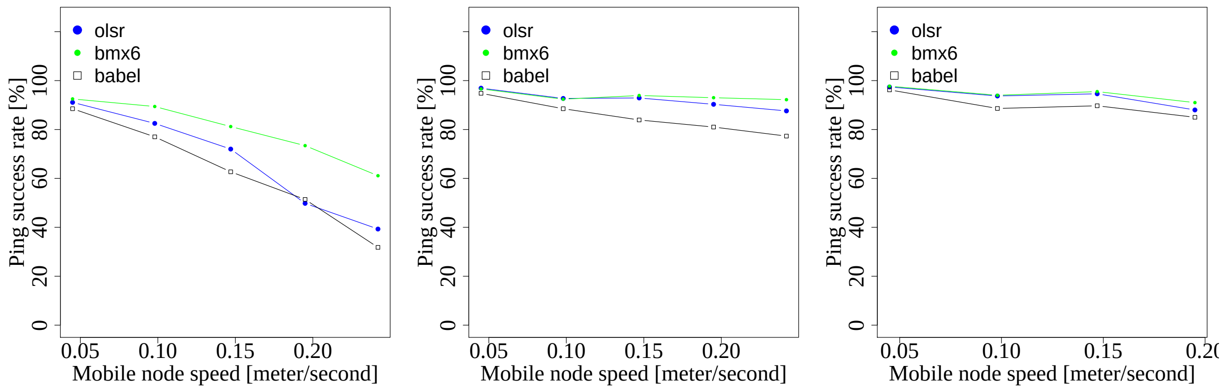


(b) Impact of TX power



(c) Impact of TX rate

Figure 14: Memory consumption depending on network size and density (power, rate)



(a) Poor link redundancy (TX power 3dBm)

(b) Medium link redundancy (TX power 4dBm)

(c) High link redundancy (TX power 5dBm)

Figure 15: End-to-end delivery success depending on topology dynamics and link redundancy

(in terms of density, size and topology dynamics) into rows with a few comparative words for each protocol.

7.1. Protocol Data Overhead

Protocol data overhead has been measured in bytes and packets per second and depending on network size, density, and dynamics.

Regarding **byte overhead depending on network size**, both, emulation- and experimentation-based measurements (Figures 10a and 3a) show (apart from one exception in experimental Babel measurements, which we will discuss later) consistent results of an essentially linear increase with a protocol specific slope and base load. In addition, BMX6 shows the highest base load but lowest slope, while Babel shows the lowest base load and a slightly greater slope and OLSR shows the greatest slope which, given the emulation-based results, raises up to 400 bps, about 120% more than Babel and 50% more than BMX6 for a network of 70 nodes. The different absolute numbers between experimentation- and measurement-based results can be explained by the greater average number of links (neighbors) per node in the different scenarios, with three versus up to five (compare Figure 8a).

A different picture arises when experimentally comparing the byte **overhead in the presence of TCP user traffic** as shown in Figure 11a. Then, transmissions naturally cause interference and thereby affect the perception of link qualities between neighbors (Figure 8a) as well as the propagation of routing information. In this scenario, we see that the overhead of Babel increases dramatically because Babel reacts to topology changes by sending unscheduled route updates. On the other hand, BMX6 and OLSR propagate routing updates periodically, independently of topology changes; therefore, this results in little effect on the overhead.

The slight increase in BMX6 can be explained by the requirement of acknowledgements when exchanging routing updates, which will cause overhead due to the retransmissions caused by traffic collisions. The opposite is the case for OLSR where the collision of link-information (TC messages) containing packets, if not successfully received via alternative links, are not further propagated and eventually result in an decreased overall overhead, an effect particularly likely in sparse networks with low link redundancy.

Another critical factor affecting protocol overhead is given by the **network density** where emulation- and experimentation-based measurements show quite different results. Looking at the former (Figure 3c), the observed shape of all protocols well matches with what one could expect from each protocol-dissemination algorithms. The highest, quite linear, slope for OLSR represents that of a non-optimized link-state protocol where information about all links in the network are propagated to all nodes for calculating a local view of the total topology. For this purpose, every node contributes to the propagation of this information by re-broadcasting new link state information once (via TC messages) and thus causing respectively increasing transmission overhead by each node. This non-optimized link-state behavior could be explained by the non-standard behavior of the OLSR implementation using a default MPR selector set of seven (instead of one, see also Section 3.3). However, compared to the experimentation-based results in Figure 11b, in the beginning the greatly increasing OLSR overhead quickly flattens for densities of around 10 or more links per node (corresponding to a TX power of 5dBm according Figure 8b), an effect which indicates that the OLSR-implementation specific MPR selector set value of seven still yields significant optimization in very dense networks.

The moderate overhead slope for BMX6 and the constant seeming slope for Babel from the emulation-based results correspond with the typical characteristics of distance-vector routing protocols, where link-quality information is only exchanged between neighboring nodes and thus affects each nodes' overhead only by a few additional link-probing related messages and as far as the size of its local neighborhood increases but not beyond. Here the aggregation of many messages into much less eventually broadcasted packets help to even flatten the observable overhead on layer 2.

Interestingly, a significantly different picture could be observed for the Babel overhead when considering the measurement-based experiments where the amount of transmitted data increases by a factor of 10 when the node density (at a power level of 6 dBm) exceeds 15 links per node. We attribute this behavior to the high susceptibility of the Babel protocol to topology dynamics, which was already observed for the impact of interfering TCP traffic in Figure 11a. In this case, such topology dynamics are caused by natural interference and consequent collisions due to the dense wireless deployments, a factor not existing in the emulation-based analysis. In fact, protocol performance instabilities, which are likely related to similar topology dynamics, were measured repeatedly in different scenarios. The exceptional experimental Babel measurements points mentioned earlier for byte overhead depending on network size are one example.

The measurement results for network **overhead in terms of packets** in Figures 4 and 12 illustrate on the one hand the dominance of protocol-specific link-probing and update intervals causing a constant and minimal packet transmission rate even in the simplest possible deployment. On the other hand, once protocol-stress factors (such as network size, density, or dynamics) cause a protocol to disseminate more data than could be aggregated into the packets sent at a minimal transmission rate, they show how packet overhead first increases in steps before scaling linearly with the byte overhead discussed earlier. In this sense, the high base rate of BMX6, at two packets/second compared to 0.5 packets/second for OLSR and even less for Babel, should only be considered relevant for deployments of rather stable and sparse networks. For more complex deployments the initial lowest packet rate of Babel can easily turn into a rate several times higher rate than that of OLSR and BMX6.

7.2. CPU and Memory Consumption

Experimentation-based measurements show (see Figures 13 and 14) that network size and density only have a very limited and generally non-critical impact on the CPU and memory consumption caused by the respective routing-protocol process. Given the embedded hardware and studied parameter space of up to 50 nodes and densities above 20 links per node, protocol-specific CPU usage always remained below 2% of the total CPU processing capacity, and virtual memory consumption (including all shared library objects mapped into the process) remained significantly below 1.5 MByte while showing only a very low increase over the studied ranges. Given these experimentation-based measurements, the memory consumption of Babel is the lowest and least increasing, while OLSR and BMX6 both show a very similar low linear increase depending on network size. Regarding density, BMX6 demonstrates a similar low slope as Babel, which matches what can be expected from any distance-vector protocol that only has to maintain the next hop towards any distant node. However, the link-state based OLSR protocol, which must keep track of all relevant links in the overall network topology, shows only a slightly greater increase of memory usage depending on density.

In contrast, the emulation-based measurements in Figure 6 depict the writable memory requirements (instead of virtual) of OLSR and Babel as completely unaffected by network size and density and below those of BMX6, which also shows a more spread requirement of memory for networks with more than 50 nodes or densities with more than 14 links per node. Nonetheless, the total memory requirements of all protocols remain non-critically low, given memory provisioning, even of resource-constrained embedded devices.

7.3. Self-healing Performance

A number of cases have been studied based on emulation or experimentation to characterize the capabilities of the different protocols to react to topology changes in different scenarios.

Emulation-based results studying the average time needed by each protocol to fix an end-to-end path depending on its length (Figure 5a) show that the two distance-vector based protocols outperform the link-state based OLSR, particularly in end-to-end scenarios with many intermediate hops.

While the poor performance of OLSR could be explained by the implementation of fish-eye optimisation, the better convergence time of BMX6 compared to Babel is surprising given the reactive nature of the Babel algorithm that should encounter spontaneously detected topology changes on demand instead of delaying the propagation of corresponding routing updates for the next update period. However, the better performance of BMX6 regarding topology dynamics is consistently confirmed in all further measurements (emulation- or experimentation-based, such as shown in Figures 5b, 5c, and 15) and must be attributed to the prevailing of other protocol characteristics. One reason is certainly given by the different link-probing and route update intervals (see Table 1) used by each protocol that allow BMX6 (using a Hello interval of only 0.5 seconds) to detect and react to local topology changes much faster than Babel and OLSR. On the other hand, to enhance the self-healing performance of a protocol, such intervals cannot be decreased without introducing additional protocol overhead, which is already significantly higher for the latter two protocols in large and dense networks and which, if further increased, would also lead to further protocol instabilities due to self-caused collisions and interference.

Table 5: Summary of observed and interpreted protocol performance characteristics from Section 7

Characteristic	OLSR	BMX6	Babel
Increase of protocol overhead depending on size, density, and dynamics			
Size:	high linear	low linear	moderate linear
Density (low density):	high linear	low linear	lowest
Density (high wireless density):	logarithmic	low linear	in non-linear steps
Topology dynamics due to interference from TCP user traffic:	negative	unimpaired	highly susceptible with typical strong growth
Increase of memory usage depending on size and density			
Size:	low linear	low linear	lowest, unaffected
Density:	linear, acceptable	low linear	low, unaffected
Comment: Non-critical given the studied range of size, density, and dynamics			
Increase of CPU usage depending on size and density			
Size:	low, total max $< 0.5\%$	low linear, total max $< 0.2\%$	low linear, total max $< 0.2\%$
Density:	varying, total max $< 1.5\%$	varying, total max $< 1\%$	varying, total max $< 2\%$
Comment: Non-critical given the studied range of size, density, and dynamics			
End-to-end path-healing performance due to topology changes			
Off time versus path length:	high, increasing slope with jump between 7 and 9 hops, $avg \sim 35s$	low, unaffected, $avg \sim 8s$	medium, unaffected, $avg \sim 16s$
Outage versus changing rate:	BMX6 shows least outage in highly changing environments All protocols equally good at low changing rates		

8. Conclusion

This paper presents an evaluation of mesh routing protocols for wireless community networks. We study the scalability, performance, and stability of BMX6, OLSR and Babel, three proactive routing protocols commonly used in these networks, through emulation and experimentation.

Our emulation and testbed-based experiments with various network conditions at different scales have provided several detailed results that compare the three protocols. In summary, we can say that Babel is the most lightweight protocol with the least memory, CPU, and control-traffic requirements as long as it is used in networks with stable links and low node densities.

However, if the protocol is used in large or dense wireless deployments with frequent link changes due to dynamic interference or nodes leaving or joining the network, then its reactive mechanisms to encounter topology changes by sending additional routing updates and route request messages turn into massive control-traffic and processing overhead. In such scenarios, OLSR and BMX6, with their strictly constant rate for sending topology and routing update messages, outperform Babel in terms of overhead, stability, and even self-healing capabilities.

The OLSR protocol significantly benefits from the MPR mechanism that (despite the highly redundant parametrization used within our experiments) achieves only a logarithmically increasing overhead depending on network density.

The BMX6 protocol benefits from its generally low control overhead due to the usage of compact local identifiers and the hiding of local state (e.g. link qualities) from globally propagated information. It differentiates from OLSR with higher memory requirements but lower control overhead and a better reaction on dynamic link changes.

Data sets and other details are at: `http://dsg.ac.upc.edu/eval-mesh-routing-wcn`.

Acknowledgement

This work was carried out with the support of the European Commission through the 7th ICT Framework Programme in the Fed4FIRE project ("Federation for FIRE", 318389), the CONFINE project ("Community Networks Testbed for the Future Internet", 288535), and by the Spanish government under contract TIN2013-47245-C2-1-R.

References

[1] Guifi.net, `http://guifi.net/`.

[2] Athens wireless metropolitan network, `http://www.athenswireless.net/`.

[3] A. Neumann, E. Lopez, L. Navarro, An evaluation of BMX6 for community wireless networks, in: IEEE WiMob 2012, no. CNBuB, 2012, pp. 651–658.

[4] A. Neumann, R. Baig, V. Oncins, Mobility performance evaluation of mesh routing protocols (memo), Tech. rep., Fed4FIRE (December 2014).

[5] R. Baig, Evaluation of Dynamic Routing Protocols on Realistic Wireless Topologies, Master's thesis, Universitat Autònoma de Barcelona (2012).

[6] olsrd - an adhoc wireless mesh routing daemon, `http://olsrd.org`.

[7] BMX6 mesh networking protocol, `http://bmx6.net`.

[8] J. Chroboczek, The babel routing protocol, RFC 6126 (Experimental) (2011).

[9] U. Ashraf, G. Juanole, S. Abdellatif, Evaluating Routing Protocols for the Wireless Mesh Backbone, in: IEEE WiMob 2007, pp. 40–40.

[10] A. Zakrzewska, L. Koszalka, I. Pozniak-Koszalka, Performance Study of Routing Protocols for Wireless Mesh Networks, in: IEEE ICSEng 2008, pp. 331–336.

[11] Q. Feng, Z. Cai, J. Yang, X. Hu, A Performance Comparison of the Ad Hoc Network Protocols, in: IEEE WCSE 2009, pp. 293–297.

[12] F. Z. Mughal, Comparative analysis of proactive, reactive and hybrid ad hoc routing protocols in Client based Wireless Mesh Network, in: IEEE ICIET 2010, pp. 1–6.

[13] S. Kumar, J. Sengupta, AODV and OLSR routing protocols for Wireless Ad-hoc and Mesh Networks, in: IEEE ICCCT 2010, pp. 402–407.

[14] A. Zakaria, H. Mohamad, N. Ramli, M. Ismail, Performance Evaluation of Routing Protocols in Wireless Mesh Networks, in: IEEE ICACT 2013, pp. 1111–1115.

[15] M. Ikeda, E. Kulla, M. Hiyama, L. Barolli, M. Takizawa, Experimental Results of a MANET Testbed in Indoor Stairs Environment, in: IEEE AINA 2011, pp. 779–786.

[16] D. Johnson, N. Ntlatlapa, C. Aichele, A simple pragmatic approach to mesh routing using BATMAN, in: IFIP WCITD 2008, Pretoria, South Africa.

[17] E. Borgia, Experimental Evaluation of Ad Hoc Routing Protocols, in: IEEE PerCom 2005 Workshops, 2005, pp. 232–236.

[18] D. Rastogi, S. Ganu, Y. Zhang, W. Trappe, C. Graff, A Comparative Study of AODV and OLSR on the ORBIT Testbed, in: IEEE MILCOM 2007, pp. 1–7.

[19] J. Friginal, D. de Andrés, J.-C. Ruiz, P. Gil, Towards benchmarking routing protocols in wireless mesh networks, Ad Hoc Networks 9 (8) (2011) 1374 – 1388.

[20] D. Murray, M. Dixon, T. Koziniec, An experimental comparison of routing protocols in multi hop ad hoc networks, in: IEEE ATNAC 2010, pp. 159–164.

[21] F. Maan, N. Mazhar, Manet routing protocols vs mobility models: A performance evaluation, in: IEEE ICUFN 2011, pp. 179–184.

[22] F. Bai, N. Sadagopan, A. Helmy, Important: a framework to systematically analyze the impact of mobility on performance of routing protocols for adhoc networks, in: IEEE INFOCOM 2003, Vol. 2, pp. 825–835 vol.2.

[23] N. Shah, D. Qian, K. Iqbal, Performance evaluation of multiple routing protocols using multiple mobility models for mobile ad hoc networks, in: IEEE INMIC 2008, pp. 243–248.

[24] N. Javaid, M. Yousaf, A. Ahmad, A. Naveed, K. Djouani, Evaluating impact of mobility on wireless routing protocols, in: IEEE ISWTA 2011, pp. 84–89.

[25] Y. Huang, S. Bhatti, D. Parker, Tuning olsr, in: IEEE PIMRC 2006, pp. 1–5.

[26] S. Azzuhri, M. Portmann, W. L. Tan, Evaluating the performance impact of protocol parameters on ad-hoc network routing protocols, in: IEEE ATNAC 2012, pp. 1–6.

[27] J. Fang, T. Goff, G. Pei, Comparison studies of ospf-mdr, olsr and composite routing, in: IEEE MILCOM 2010, pp. 989–994.

[28] M. Ikeda, M. Hiyama, L. Barolli, F. Xhafa, A. Durresi, Mobility effects on the performance of mobile ad hoc networks, in: IEEE CISIS 2010, pp. 230–237.

[29] E. Kulla, M. Ikeda, L. Barolli, R. Miho, V. Koliçi, Effects of source and destination movement on manet performance considering olsr and aodv protocols, in: IEEE NBiS 2010, pp. 510–515.

[30] Serval project, `https://www.servalproject.org`.

[31] Village telco project, `https://villagetelco.org`.

[32] Wireless battlemesh, `http://battlemesh.org/`.

[33] Official BattleMesh download server, `http://download.battlemesh.org/`.

[34] J. Avonts, B. Braem, C. Blondia, A questionnaire based examination of community networks, in: IEEE WiMob 2013, pp. 8–15.

[35] T. Clausen, Comparative Study of Routing Protocols for mobile Ad-hoc NETworks, Tech. rep., INRIA

(2004).

[36] L. Cerdà-Alabern, A. Neumann, P. Escrich, Experimental evaluation of wireless mesh networks: A case study and comparison, in: NICST 2013.

[37] Libre-mesh, `http://libre-mesh.org`.

[38] Better approach to mobile ad-hoc networking (b.a.t.m.a.n.), `http://www.open-mesh.org/`.

[39] C. Adjih, E. Baccelli, T. H. Clausen, P. Jacquet, G. Rodolakis, Fish eye OLSR scaling properties, Journal of Communications and Networks (2004) 343–351.

[40] D. S. J. De Couto, D. Aguayo, J. Bicket, R. Morris, A high-throughput path metric for multi-hop wireless routing, in: Proceedings of the 9th Annual International Conference on Mobile Computing and Networking, MobiCom '03, ACM, New York, NY, USA, 2003, pp. 134–146. `doi:10.1145/938985.939000`.
URL `http://doi.acm.org/10.1145/938985.939000`

[41] D. Johnson, G. Hancke, Comparison of two routing metrics in {OLSR} on a grid based mesh network, Ad Hoc Networks 7 (2) (2009) 374 – 387. `doi:http://dx.doi.org/10.1016/j.adhoc.2008.04.006`.
URL `http://www.sciencedirect.com/science/article/pii/S157087050800053X`

[42] L. Cerda-Alabern, On the topology characterization of guifi.net, in: IEEE WiMob 2012, pp. 389–396.

[43] Mesh linux containers, `https://github.com/axn/mlc`.

[44] S. Bouckaert, W. Vandenberghe, B. Jooris, I. Moerman, P. Demeester, The w-ilab.t testbed, in: T. Magedanz, A. Gavras, N. Thanh, J. Chase (Eds.), Testbeds and Research Infrastructures. Development of Networks and Communities, Vol. 46 of LNICST, 2011.