# Mapping a Community Network

Bart Braem, Johan Bergs, Jeroen Avonts, Chris Blondia
Department of Mathematics and Computer Science
University of Antwerp - iMinds - MOSAIC Research Group
Middelheimlaan 1, B-2020, Antwerp, Belgium
Email: {bart.braem,johan.bergs,jeroen.avonts,chris.blondia}@uantwerpen.be

***Abstract*—Community networks are self-organized networks which exist all around the world. For a large number of users, they are the only form of communication in the community itself and with the Internet. All community networks maintain some database where all information about the nodes in the community network is stored. The database information is manually inserted in the database, which commonly leads to a mismatch between the documentation and reality. In this paper we present a dynamic community network mapper, complementary to the node database. We explain the goals of this approach, show the challenges and steps we took to realize such a mapper, and give initial results.**

## I. Introduction

Often founded by networking enthusiasts, community networks all around the world form a means to experiment with networks. They offer internet access where commercial carriers will not offer it, or they provide a forum for free expression and communication. The possibilities of a community network extend well beyond this small list, and is only limited by the imagination and effort of the members of the community network.

Community networks exist all over the world, from Afghanistan[1] to Vienna[2], from Montreal[3] to Buenos Aires[4]. Community networks are very diverse in nature, each with its own characteristics. Some consist of only wireless links, while most networks combine wireless links with VPN connections between sites. Some networks like Guifi.net[5] even deploy their own fiber cables. [6] gives an extensive overview of the characteristics of community networks around the world.

The community network consists of interconnected nodes, where each node consists of multiple routers and/or antennas located on a single site. Depending on their age and popularity, the community networks vary largely in scale: a typical community network will consist of about fifty nodes, but networks of thousands of nodes also exist[7].

The community networks often rely on open source software, running on cheap off the shelf hardware. However, given the low price and the higher usability, most networks also blend in commercial hardware like routers from MikroTik[8] or antennas and/or wireless routers from Ubiquiti[9]. Routing between the devices is usually handled by either standard protocols like BGP or OSPF. Some networks also run custom protocols like BATMAN advanced or BMX6[10].

With an expanding network comes an increasing overhead from maintenance and documentation. The settings and characteristics of each new device have to be documented and monitored. This allows maintenance and system administration to be performed efficiently. Also, the expansion of the network will also introduce hardware updates and topology changes, which have to be documented too. At the moment, to the best of our knowledge, all community networks take an offline, static approach to documenting their network, in the form of a node database. While the exact properties and functionality of a node database is debatable and under active development[11], it suffices to know that the database is manually maintained. In this work we focus on the opposite approach: an automatic community network mapper, which automatically discovers and documents the community network.

In what follows, we will first describe our goals in more detail. This is followed by the encountered

challenges and the implementation itself. Afterwards, we will present results and lessons learned, with a number of conclusions and future work.

## II. Goals

The development of a community network mapper is motivated by a number of goals. The first and foremost is to lower the burden of documenting a network in a node database. Especially when the community network is altered, e.g. moving a node or upgrading its hardware, manually updating the node database is often neglected. Note that the community network mapper is envisioned to be complimentary to existing community network node databases, as outlined in section V.

A second goal stems from our work in the EU FP7 research project CONFINE[12]: community networks form a very interesting research subject. Both to improve the community networks itself and to use the community network to test existing research ideas in a more realistic context and at a larger scale. These are the primary goals of the CONFINE project.

The CONFINE project is part of the Future Internet research initiative as outlined by the EU. Community networks perfectly match this research on the Future Internet, as a they can be considered to be a viable ISP alternative in the Future Internet. Even today, a community network is the only possibility for a number of end users to access the public internet. Given this context, the CONFINE researchers also want to provide open data sets on the community networks they cooperate with. This provides feedback to the community networks and generates data for external researchers to study.

A mapping system that runs at fixed intervals, e.g. daily, can also provide feedback on and document the dynamics of a community network. As a simple example, the growth figures from Guifi.net[13] show how the community network increased during its lifetime. With more data available, e.g. the geographical data of new nodes or the resulting changes in the routing protocols, this allows researchers to study the dynamics of a network. And this could also help community networks to understand how other networks grow and possibly learn from it. Again, this forms interesting feedback to the deployment of community networks as a future internet.

Finally, the data generated by a mapper can also be fed into systems that can monitor the community network and provide feedback to the community network. Section V provides results obtained by analyzing the results of our community network mapper.

In what follows, we describe the challenges, implementation and results for a community network mapper that was tested on the Wireless Antwerpen community network[14]. We have tried to keep our implementation generalized and applicable to a wide range of community networks. Community network specific details are mentioned explicitly.

## III. Challenges

Three major challenges were met when developing a community network mapper that meets the goals outlined above: tool availability, API access and storage.

### A. Tool Availability

A number of tools to perform network discovery and mapping exist, both proprietary[15], [16], [17], [18], [19], [20], [21] and open source[22], [23]. However, none of these tools meets the goals for the community network mapper.

More specifically, the tools are often not tailored towards mapping of wireless networks with a diverse mix of proprietary and open source software. A second important mismatch lies in the definition of a node, and the grouping required to identify a node from network data, as outlined in section IV-C. To the best of our knowledge, no existing tools allow for such a grouping.

### B. API

Because off the shelf tools are not applicable, a number of data sources have been considered.

A perfect candidate for data gathering from network devices is the Simple Network Management Protocol (SNMP), providing a standardized interface and with broad adoption[24]. Numerous tools are available with SNMP support, both to offer data access on a system and to query the

offered data. Unfortunately, the broad adoption of SNMP comes with a fragmented implementation of data offered by the devices. The standard defines how to exchange data, it does not define which data to offer. In essence, each vendor can choose which functionality to implement. The implemented features vary strongly, even from one firmware release to the next. As such, relying on SNMP is a good idea only when taking care that the required features are implemented. As an illustration, a number of Ubiquiti devices do not expose any information on their ARP table or IP addresses via SNMP. This makes grouping devices cumbersome.

In the case of Wireless Antwerpen and some other community networks, another API is available: the RouterOS API offered by the MikroTik devices. While binary and proprietary to devices running RouterOS, the API allows querying for a large number of device characteristics. Similar to SNMP, using this API requires it to be enabled it on each individual device (which is not the default case). We have chosen to use this API where no SNMP was available.

Some devices like the ones from Ubiquiti offer a web interface for configuration. Besides offering easy configuration for end-users, this opens up possibilities to web scraping. In this case, the HTML is parsed and data is generated. This option should be considered to be a final resort, as the web pages to configure embedded devices often change from release to release. In our work, we have not resorted to the web scraping approach.

### C. Storage

A final challenge is often overlooked: storing the intermediate and final results of a mapper. We envision the system to have regular runs, at the moment our implementation runs at intervals of six hours. The amount of data generated by the system can be quite large. As such, storage must be efficient and flexible. E.g., data archival and access should be efficient, and changes to the fetched characteristics of a node should also be possible quickly.

Relational databases form a seemingly obvious solution. However, we do not believe that this presents a good solution for storing mapper results because the goal is flexibility. A database structure that can handle flexibly selected types extracted data and allows for flexibility when fetching different node attributes quickly becomes complex. Also, the data generated by a mapper grows very quickly, and might require database segmentation or archival. While certainly feasible, we believe these complications make the choice for relational databases less obvious[1].

As an alternative, our mapper is implemented using plain text storage. Each mapper run gets its own directory, and the directory name contains the time the mapper ran. Results of specific queries, e.g. for the node name or its ARP table, are stored in small, separate files. Processing the data is handled by a number of UNIX shell scripts. Progress of the run is logged in a simple plaintext log file. The resulting storage architecture is surprisingly scalable and flexible. Archival can be achieved by compressing the folder contents. Querying can be performed with simple grep commands, combined with awk in a shell script. Most importantly: the resulting data can be easily exported for other systems, or for external researchers as open data.

## IV. Implementation

In what follows the implementation of the community network mapper is outlined. It is based on SNMP and the RouterOS API for data retrieval and plain text storage. The mapper consists of four phases: *discovery*, *data extraction*, *grouping* and *result generation*.

### A. Discovery

During the first phase, *discovery*, the mapper tries to discover all devices by their IP address. To realize this, a starting IP address has to be configured. Starting from this device, a breadth first search algorithm is run. It is based on the neighbors of a device, as exposed by its neighbor discovery protocol. This can be the set of OLSR neighbors[25], IP addresses discovered by the Cisco Discovery Protocol (CDP)[26] or IP addresses discovered by the MikroTik Neighbor

[1]One could consider using NoSQL databases, although archival could also be cumbersome.

Discovery Protocol (MNDP)[27]. After a number of iterations this results in a stable set of discovered IP addresses.

It is important to take care of devices with multiple IP addresses during this phase, because a device can be discovered over multiple addresses. In our implementation, upon discovery we add all IP addresses of a node to the list of discovered IP addresses.

### B. Extraction

In the second phase, *extraction*, data is extracted from the discovered devices. The data to extract depends on the interests of the community network and/or the researchers operating the mapper. However, in the next phase of our implementation, the interface table and the ARP table are required. More data can be fetched with additional scripts, using SNMP and/or the RouterOS API to query the devices.

The interface table is of interest to avoid data duplication and to allow simple mapping of multiple IP addresses to one device. The data de-duplication is realized as follows. For each IP address of a device, all data files relating to the device are symbolically linked to the corresponding data files of the first discovered IP address of the device. As such, the storage overhead is limited.

### C. Grouping

One could argue that the interface table already performs some form of grouping. However, the goal of the *extraction* phase is the grouping of multiple devices into one node rather than grouping multiple IP addresses into one device. This phase also handles identification of wireless connections, which are grouped together in connections.

To group nodes, we want to identify devices in a single location. We assume that devices connected via a wire belong to a single node, and that devices connected over a wireless connection belong to a different node. This assumption holds in the studied community network Wireless Antwerpen. However, this assumption has to be improved for community networks like Guifi.net who roll their own fiber. One could argue that the interface type (Ethernet, IEEE 802.11, virtual interfaces, ...) will simply identify wired connections. However, this information is not available directly nor consistently over SNMP. The authors did not want to use proprietary APIs, to maintain general applicability.

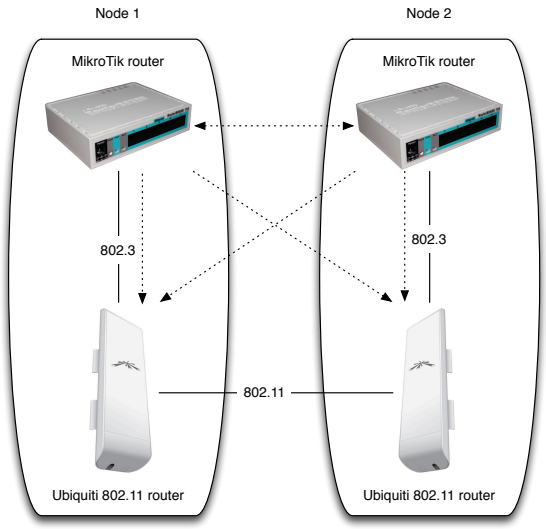


Fig. 1. Grouping scenario, dotted lines show neighbor discovery.

The main complexity for grouping can be observed in figure 1, depicting two nodes. From a device point of view, two MikroTik routers are connected over two Ubiquiti wireless routers. As mentioned before, the Ubiquiti devices do not expose any IP information over SNMP, and have to be discovered via the MikroTik routers. The MikroTik routers discover each other and the Ubiquiti devices via MNDP, as depicted by the dotted line. However, they do not discover any hop count information because the Ubiquiti routers form a layer 2 link. In this case grouping can be considered determining a reduction from six (discovered) neighbor edges to two wired and one wireless connection.

A first attempt to identify devices connected via a wire involved taking the intersection between the IP addresses in the ARP table of a node, and its discovered neighbors. Only considering the IP addresses in the ARP table of a node would result in grouping of too many devices, e.g. clients connected to the access point of a device would also be included. The result of the intersection is not sufficient because the ARP table entries are by design short lived. Especially in parts of the

community network with low traffic, the ARP table information will not always be complete.

A second attempt to identify devices connected via a wired interface involved taking the discovered neighbors, and looking up host specific routes for those neighbors with SNMP. The idea behind this is that connected wireless routers acting as layer two devices need specific routes for management. (Note that this assumption is community network specific, some networks handle this differently.) Considering only host specific routes is not sufficient. These routes might have propagated from other nodes. However, the intersection of discovered neighbors and host specific routes does not meet the mapper goals either, because routers connected to routers usually do not have host specific routes. Instead, they have IP addresses in an IP address range which the router routes for.

At the moment, the solution that is used in our implementation is the union of both solutions. This is outlined in the first part of listing 1. Essentially, this involves the intersection of the discovered neighbors on the one hand, and the union of the IP addresses in the ARP table of a node and its host specific routes on the other hand. Improving this result is certainly considered future work.

Finally, we identified wireless connections between grouped nodes. The main challenge in this case comes from the fact that wireless connections are usually handled by devices that act like ISO layer two bridges rather than ISO layer three routers. As such, the devices are not visible at IP level in the data of the access point they are connected to. Identifying these connections can be solved by SNMP, because both the BSSID a node is connected to and the BSSID of the access point are exposed. Combined, we can easily match access points clients with their access point, as outlined in the second part of listing 1.

Listing 1. Grouping Discovered Results

```
Function Group (discovered devices)
  For each discovered device i (* Group wired
      links *)
    wired_links(i) := neighbors(i) intersect
       (specific_routes(i) union arptable(i))
  End
  SSIDs = the SSIDs of all discovered devices
  For each discovered device i (* Group
      wireless links *)
    If (SSID of i is known and SSID of i is
        in SSIDs)
      wireless_links(i) := SSID of i
    End
  End
End
```

### D. Result Generation

After discovery, data extraction and grouping, the final phase of the community network mapper is *result generation*. To match the grouping information and the topology data, as generated by the community network mapper, we have decided to output our result in the form of a Graph Exchange XML Format (GEXF) graph[28], annotated with the extracted node and link properties. This XML format allows for easy manipulation in different systems, while being very extensible. A simple graph could look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.2draft"
    ... version="1.2">
<graph mode="static" defaultedgetype="
    directed">
<attributes class="node">
 <attribute id="hardware" title="hardware"
     type="string"/>
 <attribute id="IPs" title="IPs" type="
     liststring"/>
</attributes>
<attributes class="edge">
 <attribute id="type" title="type" type="
     string"/>
 <attribute id="txrate" title="txrate" type="
     integer"/>
 <attribute id="rxrate" title="rxrate" type="
     integer"/>
 <attribute id="frequency" title="frequency"
     type="integer"/>
 <attribute id="signalstrength" title="
     signalstrength" type="integer"/>
</attributes>
<nodes>
<node id="n1" label="Device 1" >
 <attvalues>
  <attvalue for="hardware" value="MT"/>
  <attvalue for="IPs" value="
      10.2.3.4,10.6.8.9"/>
 </attvalues>
</node>
<node id="n2" label="Device 2" >
 <attvalues>
  <attvalue for="hardware" value="UBNT"/>
  <attvalue for="IPs" value="10.2.3.4"/>
 </attvalues>
```

```
</node>
</nodes>
<edges>
<edge id="n1-n2" source="n1" target="n2" >
 <attvalues>
  <attvalue for="type" value="wireless"/>
  <attvalue for="txrate" value="104000000"/>
  <attvalue for="rxrate" value="130000000"/>
  <attvalue for="signalstrength" value="-56"/
      >
  <attvalue for="frequency" value="5520"/>
 </attvalues>
</edge>
<edge id="n2-n1" source="n1" target="n2" >
 <attvalues>
  <attvalue for="type" value="wired"/>
 </attvalues>
</edge>
</edges>
</graph>
</gexf>
```

The pseudo code in listing 2 describes the discovery and extraction phases of our community network mapper, and refers to the grouping phase shown before.

Listing 2. Community Network Mapper: discovery, extraction and grouping phases

```
Function Extract (d_data)
(* Extract data of device d over SNMP or some
    other API *)
Function Symlink (d_data, e_data)
(* Create symbolic link from data file(s) of
    device d to data file(s) of device e *)
discovered := {startingdevice} (* discovered
    devices (IP addresses grouped by device)
    *)
previous := {} (* previously discovered
    devices *)
While (discovered != previous)
  For each i in discovered and not in
      previous
    Extract(i_data)
    Extract(i_interfaces)
    Extract(i_neighbors)
    For each j in i_interfaces
      Symlink(i_data, j_data)
      Symlink(i_interfaces, j_interfaces)
      Symlink(i_neighbors, j_neighbors)
    End
  End
  previous += discovered
  For each i in discovered
    discovered += i_interfaces + i_neighbours
  End
End
Group(discovered)
```

## V. Results and Lessons Learned

Currently, we have a working Community Network Mapper which runs three times per day on the Wireless Antwerpen community network. Figure 2 gives the current resulting network graph, after node grouping. A number of unconnected nodes can be observed, because the grouping is not optimal at this moment. On average, one entire run takes about three hours to discover about 3000 IP addresses. This can vary depending on the performance of SNMP on the devices and the load on the host system while grouping.

A first lesson learned comes with this result: developing a community network mapper proved to be a difficult task. Tweaking the system to be faster or to include new grouping approaches takes time. Generating and processing all data multiple times can be very demanding for the hardware. To this end, we are happy to announce that we will make our code available under an open source license, allowing other researchers and community networks to use our work.

We also notice a strong need for documentation in community networks, and hope this can help leverage the cost of documenting. In our tests with Wireless Antwerpen, the differences between our mapped network data and the contents of the node database became clear very quickly. The difference usually originates from the strong growth of this community network. Documenting networks of this scale is hard and as such we believe this work helps community networks.

This is exactly where a community network mapper should become the complementary live documentation of the community network. We would like to take this concept even further in the future, and want to advocate for node databases to contain both static and dynamic information. In these *hybrid* node databases, at fixed intervals the dynamic information is fed back into and compared to the static documentation.

Another interesting result from our Community Network Mapper is the feedback we can provide to the community network. By documenting the live network situation, we can more quickly give feedback on a number of aspects and also use this

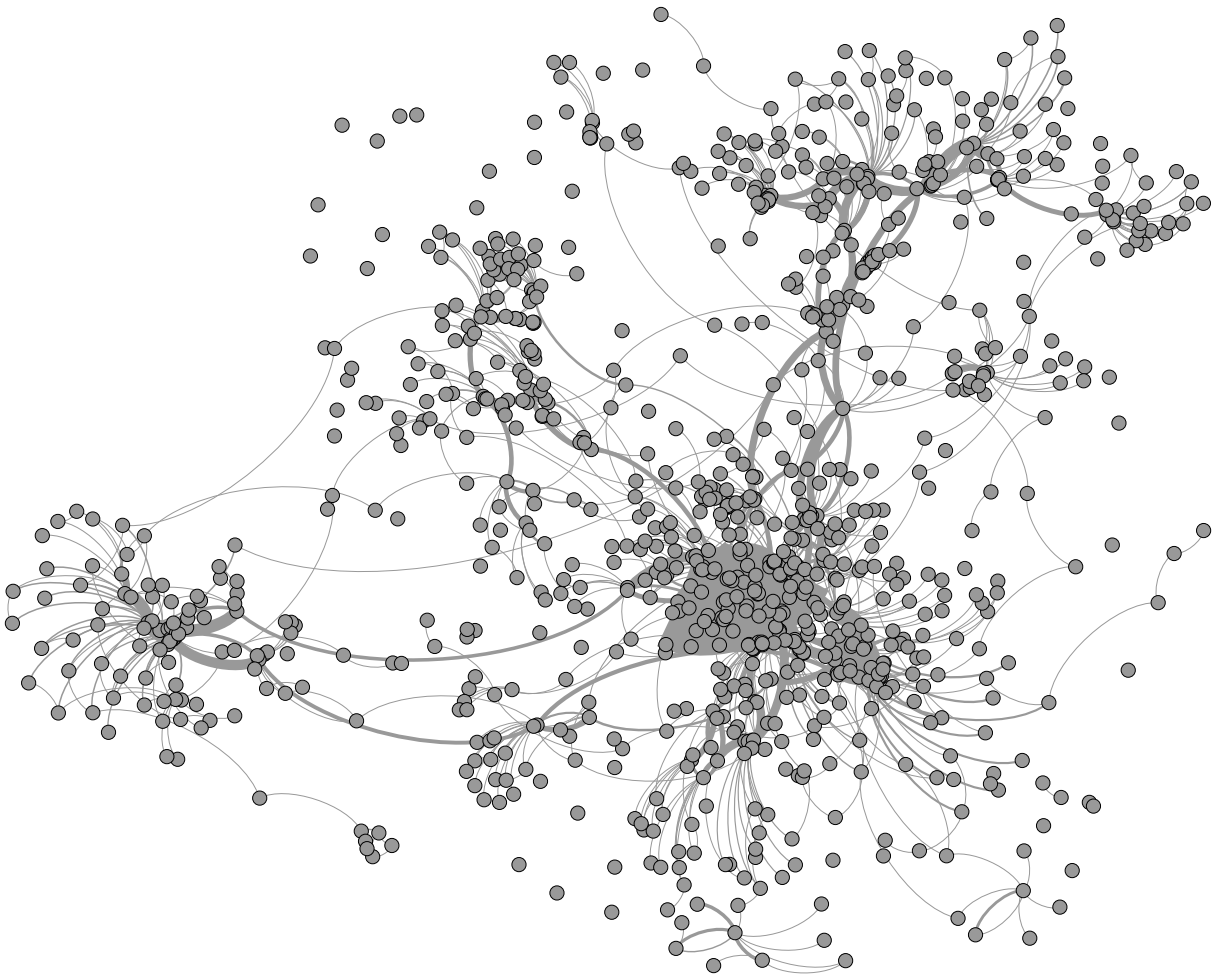

Fig. 2. Our current graph visualized.

data. As a nice example, we have been able to generate a list of duplicate IP addresses present in the community network. The community network has been able to verify and correct these problems, which also strengthens our faith in this stage of our development.

A final lesson learned considers the data access, or to be more exact the API. From our work, we believe it will become more important to ask device vendors to enable data access, e.g. via SNMP. As community networks become larger, using a standardized API like SNMP should become the standard way of handling device data. Proprietary interfaces and incomplete SNMP data complicate network maintenance. However, vendors keep resorting to proprietary APIs which have less tool support and are even more subject to change.

## VI. Conclusion

This work presented our efforts on the development of a community network mapper. The goals of this tool include: to complement the static node databases, to enable open data on community networks and to offer a historic view of the community network. We have presented challenges we encountered while implementing our system, and followed by a number of suggestions to solve those challenges and lessons learned.

While our system is still in development, we can already present some conclusions. The most important one is that mapping a community network is worth it. The data which can be derived from a live network is very valuable to both the community networks and the researchers. The resulting data is

a snapshot of the community network state, and will help to analyze and prevent certain issues. As an example, we identified duplicate IP addresses in the community network.

Going further, we believe community networks should develop a new type of node database, a hybrid node database where both static and live data is incorporated. This involves integrating a community network mapper, and results in a more rich database which can help with both documentation and network operation.

As we have indicated, our community network mapper is still being developed. At this moment, the development focus lies on the grouping of IP addresses into nodes. The grouping as outlined in this work is not perfect, and we will continue working to improve this. When this grouping is more stable, we will open source our results to allow the community networks to build hybrid node databases.

Finally, we are also testing our community network mapper on other like AWMN and Guifi.net. Only then this is finished, can we validate our general applicability claims, and further test our solutions.

## ACKNOWLEDGMENT

This work is supported by the CONFINE Integrated Project 288535. The authors would like to thank everyone involved in community networks and especially Wireless Antwerpen, for their access to the community network.

## REFERENCES

[1] "Fabfi, Jalalabad, Afghanistan," http://fabfi.fablab.af/.
[2] "Funkfeuer, Vienna, Austria," http://www.funkfeuer.at/.
[3] "Ile Sans File, Montreal, Canada," http://www.ilesansfil.org/.
[4] "Buenos Aires Libre, Buenos Aires, Argentina," http://buenosaireslibre.org/.
[5] D. Vega, L. Cerdà-Alabern, L. Navarro, and R. Meseguer, "Topology patterns of a community network: Guifi. net," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*. IEEE, 2012, pp. 612–619.
[6] J. Avonts, B. Braem, and C. Blondia, "A questionnaire based examination of community networks," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*. IEEE, 2013, pp. 8–15.
[7] "Athens Wireless Metropolitan Network," http://awmn.net/.
[8] "MikroTik," http://mikrotik.com/.
[9] "Ubiquiti networks," http://www.ubnt.com/.
[10] A. Neumann, E. López, and L. Navarro, "An evaluation of BMX6 for community wireless networks," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*. IEEE, 2012, pp. 651–658.
[11] "Node Databases," http://interop.wlan-si.net/wiki/NodesDatabase.
[12] B. Braem, C. Blondia, C. Barz, H. Rogge, F. Freitag, L. Navarro, J. Bonicioli, S. Papathanasiou, P. Escrich, R. Baig Viñas, A. L. Kaplan, A. Neumann, I. Vilata i Balaguer, B. Tatum, and M. Matson, "A case for research with and on community networks," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 3, pp. 68–73, Jul. 2013.
[13] "Guifi.net Growth Map," http://guifi.net/en/guifi/menu/stats/growthmap.
[14] "Wireless Antwerpen," http://wirelessantwerpen.be/.
[15] "LOGINventory," http://www.loginventory.com/.
[16] "Whatsup Gold," http://www.ipswitch.com/.
[17] "Alloy discovery," http://www.alloy-software.com/.
[18] "Networkview," http://www.networkview.com/.
[19] "The Dude," http://www.mikrotik.com/thedude.
[20] "NetCure Discovery," http://www.rocketsoftware.com/.
[21] "Tikmap," http://tikmap.com/.
[22] "NeDi," http://www.nedi.ch/.
[23] "Nmap," http://nmap.org/.
[24] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, "Simple network management protocol (SNMP)," United States, 1990.
[25] T. Clausen, P. Jacquet, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, L. Viennot *et al.*, "Optimized link state routing protocol (olsr)," 2003.
[26] "Cisco Discovery Protocol," http://www.cisco.com/en/US/docs/ios-xml/ios/cdp/configuration/15-mt/nm-cdp-discover.html.
[27] "MikroTik Neighbor Discovery Protocol," http://wiki.mikrotik.com/wiki/Manual:IP/Neighbor_discovery.
[28] "Graph Exchange XML Format," http://www.gexf.net/format/.