

netCommons
Network Infrastructure as Commons

Monitoring CNs: Report on Experimentations on CNs.

Deliverable Number D2.7
Version 1.0
December 28, 2017



Co-Funded by the Horizon 2020 programme of the European Union.
Grant Number 688768



Project Acronym: netCommons
Project Full Title: Network Infrastructure as Commons.
Call: H2020-ICT-2015
Topic: ICT-10-2015
Type of Action: RIA
Grant Number: 688768
Project URL: <http://netcommons.eu>

Editor:	Renato Lo Cigno, UniTN
Deliverable nature:	Report (R)
Dissemination level:	Public (PU)
Contractual Delivery Date:	Dec. 31, 2017
Actual Delivery Date	December 28, 2017
Number of pages:	40
Keywords:	community networks, monitoring tools, metrics, robustness
Authors:	Leonardo Maccari, UniTN; Renato Lo Cigno, UniTN
Peer review:	Panayotis Antoniadis, NetHood; Merkouris Karaliopoulos AUEB-RC

History of Revisions

Rev.	Date	Author	Description
v0.1	1/6/2017	Leonardo Maccari	First draft with incomplete content
v0.2	20/6/2017	Leonardo Maccari	Second draft with full content
v0.3	28/6/2017	Leonardo Maccari	Reworked the metrics
v0.4	2/8/2017	Renato Lo Cigno	Introduction and publication of the interim version
v0.5	31/11/2017	Leonardo Maccari	Update to the work up to December
v0.8	16/12/2017	Leonardo Maccari	Corrections after internal review
v0.9	26/12/2017	Renato Lo Cigno	Rework of Sect. 2, 3, and 5
v1.0	27/12/2017	Leonardo Maccari	Final proofreading

Executive summary

This Deliverable reports the advancements in the development and in the application of the monitoring instruments for Community Networks (CNs). In Deliverable D2.5 [1] we described a set of metrics, together with the source code needed for their computation, which let someone “feel the pulse” of a CN and understand their level of decentralization and the possible presence of single points of failure. Such metrics are applicable to the network topology graph of a CN, but also to the social network graph, which can be derived through the analysis of communication instruments used by the community such as mailing lists. With these instruments, it is possible to perform a multi-layer analysis and identify those CN nodes that are critical for the robust operation of the network as well as those people who are key for the community to thrive. Since people own nodes, the two factors are strongly correlated and must be analysed together. It is key to note that robustness is a key factor for the sustainable growth of a CN. While the other tasks in WP2 focus on defining, motivating, and embedding by design the CN sustainability, these are largely offline processes. This task complement this vision with tools that make it possible to continuously (on-line) monitor the evolution of the CN, in order to prevent failures which hinder the growth of a network.

This deliverable further develops this theme in two directions. The first one is the development of new metrics and new source code. An important part of this process is the realization of scientific publications for delving deeper into the methodology and developing it further according to the feedback from the academic and research community (in Chapter 2). The analysis of two running networks suggests that CNs, albeit being part of the family of “spatial networks”, behave in a different way depending on external conditions. In spatial networks, contrary to other networks such as social networks, the nodes have a position in space. Models of spatial networks typically tend to generate networks with a strong spatial hierarchy, where the geographical areas under the “influence” of each node are disjoint. In our experimental study, one of the analysed CN confirms this, while another exhibits a different pattern. We consider this an important result since it suggests that individual CNs may bear distinct features that distinguish them from other similar networks.

The second direction of progress over D2.5 has dealt with the integration of the aforementioned metrics with existing software monitoring tools of CNs. In this regard, our work has focused on two currently used platforms, NodeShot and OpenWISP2. The two pieces of software have been created by members of the ninux.org network and by a community of people including a few developers from ninux, respectively. Whereas NodeShot is used only in the ninux community, OpenWISP2 is of broader interest; thus, we have focused our developments on this platform, which we detail in Chapter 3.

The deliverable details the contribution of Task 2.4 on netCommons goals, discussing the impact on ninux and other CNs. There is an ongoing discussion with ninux members to encourage the adoption of solutions studied and proposed in netCommons Task 2.4, and the integration with OpenWISP2 will help disseminating these concepts outside the ninux community. Further adoption and impact of our software in the third year of the project will be documented as dissemination activities in WP6.

Contents

1. Introduction	7
1.1. Contribution to netCommons Goals	7
1.2. Impact on Community Networks	8
2. Spatial Networks and Network Hierarchy	9
2.1. Preferential Attachment vs. Cost-Based Analysis in CNs	9
2.2. Network Separation	9
2.3. Ninux as a Spatial Network	11
2.3.1. Comparison with the FFWien CN	12
3. Monitoring Ninux	15
3.1. The NodeShot Architecture	15
3.2. The NetJSON and OpenWISP Ecosystem	16
3.2.1. The NetJSON Specifications	16
3.2.2. netjsongraph.js	17
3.2.3. django-netjsongraph.js	19
3.2.4. The OpenWISP2 Platform	19
3.2.4.1. OpenWISP2 in Ninux	20
3.3. Developments in the Period M12-M24	21
4. Developments on OpenWISP2	23
4.1. The netjson_robustness library	23
4.2. Modifications to the netjsongraph.js Module	24
4.3. Modification to django-netjsongraph	29
5. Impact on the ninux Community	31
6. Conclusions	32
Bibliography	33
A. Spatial Separation	35
B. Appreciation Letters	37

List of Figures

2.1.	The Interest zones for all the levels of the ninux network. Note that level 0 is made of a single zone that is very narrow and thus, barely visible in the picture. Level 1 contains a single Interest zone, and Level 5 is completely separated.	13
3.1.	The ninux network as presented in NodeShot	15
3.2.	An example of a NetJSON dump.	17
3.3.	The netjson visualization of the ninux network in Florence.	18
3.4.	The components of the OpenWISP2 platform, taken from the github project page.	21
4.1.	The standard visualization of the OpenWISP2 platform	25
4.2.	The new visualization given by the use of <code>netjson_robustness</code> and <code>netjsongraph.js</code>	26
4.3.	Detail of the information displayed when clicking on a cut-point.	27
4.4.	The AWMN network.	28
4.5.	The block-cut tree visualization of the AWMN network.	28
4.6.	Integration of the different visualization in OpenWISP2	29
A.1.	An example definition of influence zone.	36

List of Tables

2.1.	Average intra-level separation in ninux.	11
2.2.	ninux inter-level separation, percent of area covered at each level and corresponding number of nodes. Since areas are overlapping the third column does not sum to 100%, and the sixth level areas are fully overlapping with the previous ones.	11
2.3.	Main attributes of the geo-located ninux communication network.	12
2.4.	Average intra-level separation within each level in FFWien.	14
2.5.	FFWien inter-level separation, percent of area covered at each level and corresponding percentage of nodes.	14
A.1.	Average separation within each level with various choices of root node. Dash means that there are less than 2 zones in the level.	35

List of Acronyms

ARCI	Associazione Ricreativa e Culturale Italiana
CBA	Cost-Benefit Analysis
CN	Community Network
NGO	Non-Governmental Organization
PA	Preferential Attachment
NMS	Network Management System
WISP	Wireless Internet Service Provider

1. Introduction

This is the final deliverable of Task 2.4 “Monitoring instruments for CNs”. Naturally, the starting point of this report is D2.5 [1], reporting on the development of multi-layer graph metrics that Community Networks (CNs) can use to understand the degree of decentralization of both their communication and social networks. The analytic classification work provided by D1.2 [2] and D1.4 [3] as well as the social and legal analysis of D2.2 [4] and D4.2 [5] are clearly also part of the background needed to develop and understand this work. Additional background is provided by the paper [6], developed thanks to netCommons support. Its ideas and results were partially included in D2.5 and partially into this Deliverable. D2.5 has documented how the analysis of a CN at different logical levels of its structure and organisation can highlight weak points, vulnerabilities and structural patterns that are not easily spotted by a third party looking into the CN. Most importantly, these are also invisible to the owners and developers of the networks itself, as for instance the dominance level of a small clique (or, even, a single person) in the social structure of the network. The general background for this work is framed by the scientific works [7, 8, 9, 10, 11, 12, 13, 14, 15], the most recent of which are also partially supported by netCommons.

This Deliverable documents the additional steps taken in the second year of the project to introduce a new analysis based on the interpretation of a CN as a spatial network. A spatial network is a network in which every node is assigned a position in a coordinate system. The study of spatial networks is important in systems that represent physical infrastructure such as road maps and energy distribution systems [16]. A community network is indeed a spatial network. The peculiarity of spatial networks, as opposed to other kinds of network graphs (a social network graph, for instance) is that links have an associated cost, and the cost increases with the length of the link. In the literature, it was observed that for this reason spatial networks tend to develop a spatial hierarchy [17]. In this Deliverable, we first detail this concept. Then we analyse two networks for which we have localization data as to whether they have developed the same kind of hierarchy, and we show that this is partially true. This suggests that the spontaneous evolution of a CN may lead to another kind of vulnerability, which should be addressed by the community, and we give simple suggestions on how to prevent this.

This Deliverable also reports on the experiments with the tools developed in D2.5, and their integration with an open source monitoring instrument for CNs, and wireless networks in general, to increase the potential impact of netCommons work. The OpenWISP2 Network Management System (NMS) is an open source platform that is gaining attention as management tool among community networks and small Wireless Internet Service Providers (WISPs). It is developed by a community of developers, receiving feedback from practitioners (some of them being part of the ninux network), and seems a promising solution for the management of bottom-up networks. We decided to integrate part of the instruments detailed in D2.5 into OpenWISP2. Currently OpenWISP2 does not include the information about node position and node ownership, so at the time of writing, we need to restrict the analysis to the communication network. As soon as OpenWISP2 is extended with new data-sets, the code we introduced in OpenWISP2 will be able to show most of the metrics we proposed so far. As detailed in Chapter 3 in order to achieve a higher impact on communities, we preferred to integrate a subset of the functions of our code (with an extensible design for future improvements) into OpenWISP2 rather than maintaining our own monitoring system.

1.1. Contribution to netCommons Goals

Historically, the adoption of distributed systems and functions instead of centralized ones has always been related to the reduction of single points of failure. A distributed system is generally imagined as a resilient,

self-healing system that is able to re-configure itself after a failure. As a relevant example, since the very beginning of packet switching, the Internet was imagined as a distributed (or at least decentralized) system in order to make it robust to the failure of some of its components.

However, the sheer difference between distributed and decentralized is sometimes not well understood, and the principles underlying distributed systems are interlaced with peer-to-peer paradigms, bottom-up or grassroots development and the like. How are distributed systems indeed designed (or not-designed) and deployed? There is a general tendency of considering “distributed” whatever system was not planned to be hierarchically organized from its beginning. This is the case of a community network, in which the participants avoid as much as possible the actions that would introduce an explicit hierarchy in the network or in the community. However, if we look at both the literature in the technical and in the social sciences fields, we observe that implicit hierarchies tend to be spontaneously created in networks, even when there is no explicit design decision to introduce them. Some systems naturally tend to create a hierarchy, as also Chapter 2 illustrates. A system that includes an implicit hierarchy should be more protected, as all hierarchical systems tend to be more vulnerable to failures of the higher tiers. As already explained in D2.5 the designer of an hierarchical system must take into consideration the robustness of a network, and add in key elements providing the infrastructure with some redundancy. If a system, instead, is thought to be distributed (and thus, naturally resilient), but it actually hides a hierarchical organization, the system is vulnerable.

The goal of this Task has been to introduce some metrics and the open source tools necessary to compute them, which could unveil to what extent a community network is fragile in the sense of presenting single points of failure. While D2.5 introduced the metrics, this deliverable extends that analysis and documents the effort in integrating these metrics into an existing network monitoring system. The final result is an open source instrument that can tell, at a glance, what are the single points of failure of a network, and thus help a community improve its resilience.

In the context of WP2, we point out that resilience is a key element of sustainability. In general, a system is considered sustainable if, in first place, is able to endure. This concept is naturally extended to the impact of a system on the outside world [18] but, in first place, a CN needs to be able to grow while maintaining intact its funding principles. If a CN can not grow because periodically some key components crash or abandon the community, the CN is not sustainable. Similarly, if it grows, but it changes its nature (from a P2P network to a network driven by a very small minority of people) again, its initial model can not be thought as a sustainable one. This Task helps communities highlight their points of failure, and thus evolve in a sustainable way which is one of the explicit goals of netCommons.

1.2. Impact on Community Networks

The impact of Task 2.4 is rather long-term since monitoring requires time, the analysis of the data obtained with the monitoring activity even more, and modifications of Community Networks (CNs) or their management can only happen after the analysis. The following list summarizes the actions that are expected to have the major impact on CNs, while Chapter 5 discusses in detail the impact already documented on ninux.

- The metrics introduced in this Task, and the source code developed and made public, are a powerful tool for analysing the state of CNs and understanding their eventual fragility.
- Carrying out an analysis on ninux unveiled vulnerabilities that were not earlier perceived. These results have been shared with the community, which is now taking actions to mitigate them.
- Part of the research work has been integrated with a promising platform for the management of CN called OpenWISP2. The platform makes it possible to monitor some key features of a network in order to give direct and continuous feedback to the community members, and prevent the creation of single points of failure.

2. Spatial Networks and Network Hierarchy

This chapter details the work done in the characterization of a CN under the lens of “spatial networks”, a specific type of networks that presents peculiar features. It is at the base of a scientific paper that is currently under preparation, and will extend the work done in [6].

A spatial network is a graph $G(V, E)$ made of a set V of nodes and a set E of edges in which every node has a “position” attribute. Spatial networks are used to represent physical systems, such as road networks or power grids [16], and their growth can be modelled using a Cost-Benefit Analysis (CBA) framework [17]. A CBA seeks a balance between two competing effects: the first is the classical “rich gets richer” effect, in which the probability that a new node v_i is connected to an existent node v_j increases with the number of neighbors that v_j already has (as in the Preferential Attachment (PA) model [19]). The second effect is given by a real life constraint: the cost of a physical edge increases with its length, moreover, beyond a certain distance, setting up a link becomes also very difficult, thus the probability of adding an edge between v_i and v_j decreases with the distance between v_i and v_j . It has been shown that when the second effect has a non-negligible influence, spatial networks tend to become hierarchical networks, and we are interested to understand if this happens also with CNs. Before we give a precise measure of the hierarchical structure of a network, we justify how in a CN both effects exist.

2.1. Preferential Attachment vs. Cost-Based Analysis in CNs

Consider a node v_i placed in a dominating position (for instance on the top of a hill). Potentially, many newcomer nodes will have line of sight with v_i , and thus, v_i will probably have many neighbors. As a consequence, the community will improve the node adding radio devices, which will increase the horizontal angle covered by the node (recall that large-scale CNs primarily use directive antennas). This will make it even more likely that new neighbors can be added, so the “rich gets richer” effect takes place. However in a CN the performance of a wireless link decreases with its length and links of arbitrary length can not be realized. These two competing effects exist and it is interesting to understand if a CN, like other spatial networks modelled with a CBA, shows a hierarchical structure.

2.2. Network Separation

We can now introduce the definition of a metric that expresses how much a network can be considered hierarchical. We start from a generic metric used in the literature [17] and we customise it to support the specific characteristics of CNs. We omit some design details in this section, which can be found in Appendix A.

The first step to define a spatial network and its separation is the identification of a root node, say v_0 . Next, every other node in the network is assigned a “level” $l(v_i)$, which measures the distance from the root node v_0 . The selection of v_0 is critical, but also somewhat arbitrary, unless a pre-defined structure is present, or is sought for, which is not the case for CNs. One logical way of selecting v_0 is choosing the “most central” node, but centrality must be defined before hand. We tested several centrality metrics to define the root node (see Appendix A for the tests and selection criteria), and we finally chose the node with the highest eccentricity¹. The level $l(v_i) = d(v_0, v_i)$ is simply defined as the distance (in terms of hops) of v_i from the root node.

¹The eccentricity of node v_i is given by $\frac{1}{\max_j(d(v_i, v_j))}$ where $d(v_i, v_j)$ is the distance from v_i to v_j . Thus the eccentricity is a centrality metric that measures how much a node is barycentric in the network, or in other words how much it will cost to distribute a message from that node to the most distant node in the network.

For each node v_i a subset $N'(i)$ of its neighbors $N(i)$ is defined as:

$$N'(i) = \{v_j \mid v_j \in N(i) \wedge l(v_j) > l(v_i)\}. \quad (2.1)$$

i.e., among the neighbors of v_i $N'(i)$ collects the nodes v_j that have a higher level than v_i , or in other words those that are farther away from v_0 given the distance selected.

Given the subset $N'(i)$, we can define an “influence zone” of v_i , which is an area that contains all the nodes in $N'(i)$; the specific definition of the area is application-dependent, meaning that it must be representative of some specific features of the application domain considered. For CNs, we use the convex hull of all the positions of the nodes in the subset, because this spatial definition correctly captures the notion of influence for a physical network whose growth is greatly influenced by the possibility of setting up a link with a given node v_i . Given a level l and a node $v_i \mid l(v_i) = l$ we call \mathcal{I}_l^i the influence zone of v_i , and we call $\mathcal{I}_l = \cup_i \mathcal{I}_l^i$. A spatial network is said to be geographically separated if both these conditions hold:

$$\mathcal{I}_l^i \cap \mathcal{I}_l^j = \emptyset \quad \forall l \text{ if } i \neq j \quad (2.2)$$

$$\mathcal{I}_{l+1} \subset \mathcal{I}_l \quad \forall l \quad (2.3)$$

Real networks are never completely separated, so a metric to measure their degree of separation is needed. The separation of two zones in the same level is defined as:

$$s_l(i, j) = 1 - \frac{\text{Area of the overlap between } \mathcal{I}_l^i \text{ and } \mathcal{I}_l^j}{\min(\text{Area of } \mathcal{I}_l^i, \text{Area of } \mathcal{I}_l^j)} \quad (2.4)$$

Similarly, we can define the separation index for level l , which we refer to as intra-level separation, as the average of the separation of the zones in the same level. If we call V_l the subset of V containing all the nodes at level l , then:

$$s_l = \frac{2 \sum_{i=0}^{\|V_l\|} \sum_{j=i+1}^{\|V_l\|} s_l(i, j)}{\|V_l\|(\|V_l\| - 1)} \quad (2.5)$$

where $\frac{\|V_l\|(\|V_l\|-1)}{2}$ is the number of all the possible couples (v_i, v_j) of nodes in V_l and $\|\cdot\|$ is the size of a set. If s_l is close to 1, then the zones inside level l are almost perfectly separated, otherwise, there is overlapping in the intersection between each zone in the graph.

We also define a metric to measure inter-level separation:

$$\hat{s}_l = 100 * \frac{\text{Area of } (\cup_{i=0}^{l-1} \mathcal{I}_i)}{\text{Area of } (\cup_{i=0}^{l_{max}} \mathcal{I}_i)} \quad (2.6)$$

where l_{max} is the number of levels in the network. The progression of the values of \hat{s}_l tells if the levels of the network are progressively covering larger portions of the territory or the levels are organized like Russian dolls. The reason why we are interested in spatial separation is that a network that is fully separated is strongly hierarchical: Eq. (2.2) says that nodes at the same level can not communicate without first ascending and then descending again in the network hierarchy, and in practice the topology of the networks becomes a tree. In such a network the nodes that are close to the root node are more important than the others, and their failure will dramatically impact the rest of the network. Eq. (2.3) says that the influence of every node is included in the influence of its parent, in practice, the network does not grow if the root node does not enlarge its influence zone (and the same stands for the other levels in cascade).

Since the literature shows that in spatial networks modelled with a CBA the average value of s_l quickly saturates to 1 as soon as the cost-per-mile of a link becomes non-negligible [17], in the sequel we analyse if and to what extent the ninux community unwillingly built a hierarchical, but not redundant network, thus making it fragile

to both physical and social impairments. This analysis can be replicated in any other network with the tools we developed, giving a mathematically and technically sound measure of the networks resilience.

2.3. Ninux as a Spatial Network

Tab. 2.1 reports the intra-level separation at each level in the ninux network, together with the number of zones per level. Tab. 2.2 reports the values of the inter-level separation, the corresponding percentage of the total area covered by level l , and the percentage of nodes included in each level. Tab. 2.1 shows that for every level, s_l is fairly high, that means that the nodes of the network in the same level are not densely connected and the chances that there are “horizontal” paths from a node to another are few. On the other hand the values of \hat{s}_l suggests that the network expands from its periphery, so that the higher the level, the more area is covered.

<i>level</i>	<i>zones</i>	<i>s_l</i>
0	1	-
1	1	-
2	3	0.93
3	4	0.95
4	6	0.90
5	9	1.00
6	6	0.80

Table 2.1: Average intra-level separation in ninux.

<i>level</i>	\hat{s}_l	<i>Area (%)</i>	<i>Nodes (%)</i>
0	0.37	0.37	5
1	22.85	22.85	44
2	31.33	7.78	63
3	54.87	12.55	78
4	66.37	45.66	81
5	100.00	31.60	100
6	100.00	9.63	100

Table 2.2: ninux inter-level separation, percent of area covered at each level and corresponding number of nodes. Since areas are overlapping the third column does not sum to 100%, and the sixth level areas are fully overlapping with the previous ones.

The first characteristic is similar to the other spatial networks described in the literature, which means that CNs are not an exception compared to other networks. In practice, there is no direct incentive in making the network more dense and redundant, while there are incentives in connecting the highest number of nodes with the lowest number of edges. As we see from table Tab. 2.3 the network is made with a number of edges that is close to the minimal number (number of nodes minus one). On the other hand, the inter-level separation shows that, contrarily to other networks, a CN does not enlarge from its center, but from its edges. This means the network is not strictly hierarchical (like a gas distribution system, for instance) and allows the fringe of the network to expand without having to add antennas on the nodes in the center of the network. This characteristic is quite evident observing that the covered area increases with the level, and it is clear looking at Fig. 2.1, that reports the zones on top of the map of Rome. Note also that there is no evident correlation between the level and its s_l ; excluding level 5, that is fully separated, all the others are only partially separated.

This analysis shows that ninux failed to realize a spatially distributed network, and realized a spatially hierarchical network. Since interest zones of nodes at the same level are often disjoint, if a link that goes from one level to the higher one is broken, not only there is no way to re-route traffic across some other zone in the same level, but it is also improbable that some other node in the neighborhood can be re-configured (re-pointing its antenna) to cover the area that was left uncovered. Thus, the higher the level of a node, the more important it is to guarantee a wide geographical coverage of ninux.

A possible means to reduce network separation would be to increment the density of links per node. At the current state of things, a new node v_i is connected only to the closest node in line of sight, and then, a new device is added to it only when some other new node v_j needs to connect to v_i to enter the network. One way

<i>Attribute</i>	<i>Value</i>
# nodes	114
# edges	128
average degree	2.246
diameter	13
average path length	6.014
modularity	0.79
density	0.02
average clustering coefficient	0.067

Table 2.3: Main attributes of the geo-located ninux communication network.

of increasing the density is to mount an additional device on v_i at its creation, even if there is no other node to connect to at that time. Adding spare devices pointing in an uncovered direction will make it more likely that in the future new nodes will join, or that existing nodes will be connected to more than one other node, in order to increase the density. With an even stricter approach this could be translated into avoiding leaf nodes, so that a new node is added to the network only if it can connect to at least two other nodes, or at least proper incentives are given to avoid leaf nodes if possible.

Note that, after the diffusion of the initial results on the analysis of ninux at the ninuxday 2016, one ninux island decided to start a discussion on the formal agreement that every ninux participant needs to accept to participate (the so-called Picopeering agreement²). The proposed modifications include clauses that improve the communications between peers forcing participants to make themselves available to receive communications from other potential peers. If they do not respect this clause, they may be excluded from the network, as they do not respect the agreement. This simple modification, together with a proposal of standardisation of network nodes in a way that makes it easier to transform a leaf node into a non-leaf node, are the answer that the ninux community (in particular, the impulse came from the ninux community in Calabria, South of Italy) to minimize the possibility of creating leaf nodes in ninux. This was one of the hot topics of discussion in ninuxday 2017, after the discussion on centralization and decentralization that took place in 2016, also with the input of netCommons.

2.3.1. Comparison with the FFWien CN

The spatial coverage of ninux depends on a number of factors that are specific to that network. In order to broaden our analysis we were able to collect the geo-referenced data of another network, the FunkFeuer network in Vienna (Austria), which we refer to as FFWien³. FFWien is a much denser network, made of 196 nodes and 249 edges, concentrated in an area that is 13 times smaller than the area of ninux. Tabs. 2.4 and 2.5 report the separation metrics for FFWien and show a quite different situation, with lower average values, especially in the first 3 layers, which contain 77% of the nodes.

This difference outlines that a CN is strongly influenced by the external conditions, such as the density of nodes, the capacity and reach of the wireless devices, and the altitude profile of the area.

To understand the general properties of a CN, it would be extremely interesting to develop a network topology generator that takes into consideration these conditions and easily generates realistic topologies, in order to study their properties in different environments, such as densely inhabited cities or rural areas. Let us expand this vision with some more concrete examples. In the scientific literature, the models used to describe networks have passed through several generations. At the beginning, simple models like the Erdős graphs were used to describe networks. These models are now the base of any graph theory book, but have been found to be strongly

²Directly derived from <https://picopeer.net/>.

³See <http://funkfeuer.at>

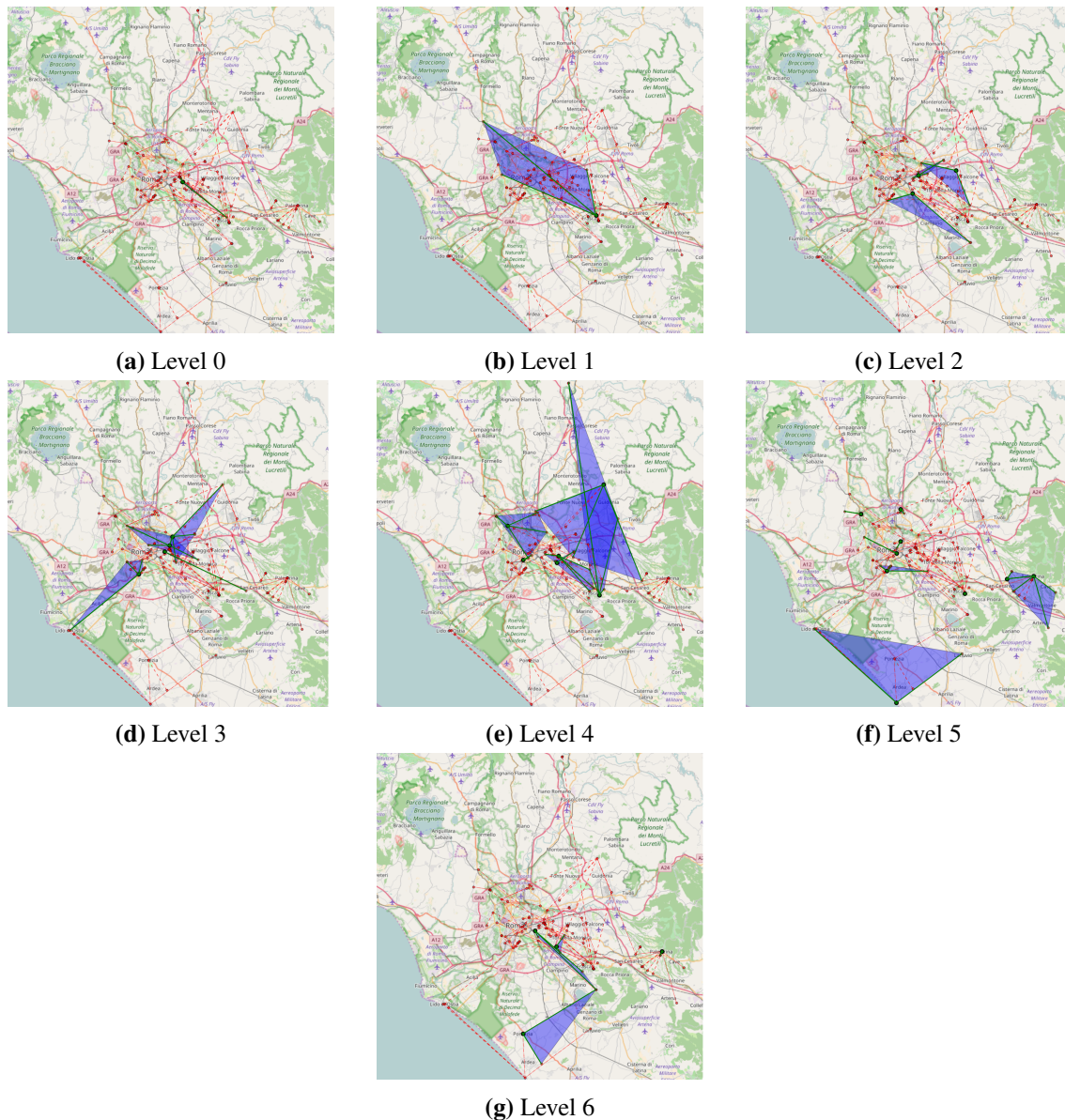


Figure 2.1: The Interest zones for all the levels of the ninux network. Note that level 0 is made of a single zone that is very narrow and thus, barely visible in the picture. Level 1 contains a single Interest zone, and Level 5 is completely separated.

different from real networks. A second step is represented by the Waxman model [20], which introduces the concept of “distance” in the generator, and subsequently, the research due to Barabási [19] produced a large interest in power-law graphs. Finally, Willinger and Doyle disputed the approach of Barabási and proposed a new family of models for real networks [21]. In the last decade, this theme lost scientific appeal in favour of efforts that try to map the real structure of the Internet. Today, a new wave of graph generators could be triggered by the analysis of CNs. First of all, these networks are innovative, and large scale networks have been appearing only in the last few years. Second, their future evolution is unknown, and, as we stated multiple times, it depends on the community that runs the network.

A synthetic network generator could be helpful to generate potential topologies that leverage on the availability of many sources of open data. City maps, buildings topologies, elevation data, could be helpful to imagine the evolution of a network given the underlying real world constraints. Moreover, adding cost consideration

<i>level</i>	<i>zones</i>	<i>s_l</i>
0	1	-
1	4	0.78
2	10	0.53
3	14	0.82
4	11	0.98
5	4	1.00

Table 2.4: Average intra-level separation within each level in FFWien.

<i>level</i>	\hat{s}_l	<i>Area (%)</i>	<i>Nodes (%)</i>
0	1.39	1.39	6
1	18.59	15.33	46
2	54.62	45.56	77
3	74.95	48.80	92
4	99.63	41.42	97
5	100.00	0.72	100

Table 2.5: FFWien inter-level separation, percent of area covered at each level and corresponding percentage of nodes.

to this generator (the cost of setting-up and maintaining a wireless link) could help understand the cost to the final user of such networks, and thus, the amount of people that could be interested in this technology. As the data about population income are publicly available by national surveys, and are geo-located, matching cost considerations with geographic distribution of people could help understand how and where these networks could possibly evolve. With such an instrument it would be possible to generalize the features of CNs and compare them with the topologies of wired networks to understand if there are any intrinsic differences rooted in the technology used to realize CNs. These differences would be not only technological, but also of social impact.

This is not a goal that can be reached in the netCommons project, but is part of future research that will be carried on, based on the results and experiences of netCommons.

3. Monitoring Ninux

This section describes the two software platforms that T2.4 used as a base for the development. The first one is the NodeShot architecture used by the ninux CN to visualize and manage the network. The second platform is composite: a set of other tools that are based on the NetJSON standard and the OpenWISP platform. While the first platform is used only by ninux and is not actively supported, the latter is the approach that ninux has taken for its future developments and involves a community of users that is potentially much larger than ninux alone. Given the larger potential impact on CNs we decided to focus on the second solution, even if, being currently under development, this may introduce some delay in the monitoring phase.

3.1. The NodeShot Architecture

NodeShot is the software currently used by the ninux community to manage the network. NodeShot has the following goals:

- Display the map of the network;
- To be the entry-point for the new user that wants to add a new node to the network. The new user will add a new “potential” node to the map and contact (via the platform) the owners of the nodes that are in line of sight with his/her new node;
- Play the role of a Local Internet Register. When a new node is added to the network, an IP address and a subnet needs to be allocated for the node, in a way that does not conflict with the already assigned ones. Being ninux a distributed network, the IPs already assigned are reported in a Wiki page, but also on the mapserver.

NodeShot is an open source software developed internally by the ninux community that achieves these goals. There are currently two versions of NodeShot, both connected to the same database, both written in Python. The first one is the legacy platform, the second one is a full rewrite of the legacy platform, and can be visited at <https://ninux.nodeshot.org/>, Fig. 3.1 presents a screenshot of the new interface.

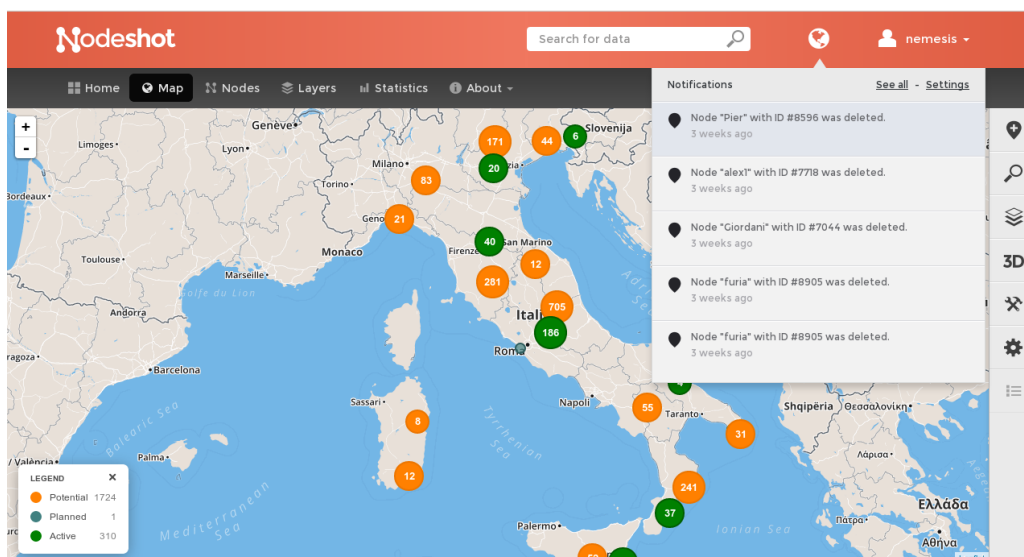


Figure 3.1: The ninux network as presented in NodeShot

Albeit the platform reached a relatively stable state, and it is used by many ninuxers, the author of the platform decided to move to a new project, based on the NetJSON specification. There are two main reasons for this choice, the first is that the platform was realized as a stand-alone monolithic framework, which quickly turned to be hard to maintain, the second is that the author of the project felt that this was “yet another CN mapserver”. This second issue deserves to be detailed, since it describes a trend that is present in the world of CNs. While CNs cooperate and communicate to develop the basic building blocks of the networks (drivers, routing protocols etc. . .) the rest of the applications needed to run a network are generally realized ad-hoc by every community. There are today several map visualizers available for CNs: NodeShot for ninux, freifunk-karte¹ for Freifunk, NodeWatcher for Wlan-Slovenija², and many more. All these pieces of software share a common core of functions but also implement some peculiar feature related to the needs of the community that developed it. As a result, they are not interoperable and there is a large duplication of efforts from community to community. Federico Capoano, the main developer of NodeShot, decided to start from scratch with a new approach, based on the definition of open standards to make the projects interoperable, and modular tools to visualize and present the data, plus perform network and device management.

The standard for the description of a community network is NetJSON, and the suite of tools currently under development revolve around the OpenWISP platform, described in Sec. 3.2.

3.2. The NetJSON and OpenWISP Ecosystem

3.2.1. The NetJSON Specifications

As reported in the official netJSON website³:

“NetJSON is a data interchange format based on JSON designed to ease the development of software tools for computer networks. NetJSON defines several types of JSON objects and the manner in which they are combined to represent a network: configuration of devices, monitoring data, network topology and routing information.”

The stated goal of netJSON is to:

“build an ecosystem of interoperable software tools that are able to work with the basic building blocks of layer2 and layer3 networks, enabling developers to build great networking applications faster.”

NetJSON is under development and it is described in an informational RFC⁴. The final goal of NetJSON is to let every community network interoperate easily with a common format. In the past in fact, many CNs developed non-portable and non-interoperable software that do approximately the same things such as an IP database, a map server, a monitoring tool. While it is very hard to coordinate these efforts in a global way (every community has specific needs and every developer has specific skills) it is easier to define a common format that developers can use to at least make their software partly interoperable. NetJSON is today the main effort toward such an interoperable platform, and, while its formal standardisation is obviously following the lengthy and uncertain paths of IETF processes, its use is instead growing among the various communities and developers⁵.

Fig. 3.2 reports an example of the NetJSON syntax, and shows how NetJSON has the advantage of being clear, human-readable and extendable with custom data (included in the “properties” tag).

¹ See <https://www.freifunk-karte.de/>

² See <https://github.com/wlanslovenija/nodewatcher>

³ See www.netjson.org

⁴ See draft-capoano-kaplan-netjson-00, <http://netjson.org/rfc.html>

⁵ See <http://netjson.org/docs/implementations.html> for a set of software pieces that use NetJSON

```

{
  "type": "NetworkGraph",
  "protocol": "olsr",
  "version": "0.6.6",
  "revision": "5031a799fcbe17f61d57e387bc3806de",
  "metric": "etx",
  "router_id": "172.16.40.24",
  "nodes": [
    {
      "id": "172.16.40.24",
      "label": "node-A",
      "local_addresses": [
        "10.0.0.1",
        "10.0.0.2"
      ],
      "properties": {
        "hostname": "node1.my.net"
      }
    },
    {
      "id": "172.16.40.60",
      "label": "node-B",
      "properties": {
        "hostname": "node2.my.net"
      }
    }
  ],
  "links": [
    {
      "source": "172.16.40.24",
      "target": "172.16.40.60",
      "cost": 1.000,
      "cost_text": "1020 bit/s",
      "properties": {
        "lq": 1.000,
        "nlq": 0.497
      }
    }
  ]
}

```

Figure 3.2: An example of a NetJSON dump.

3.2.2. netjsongraph.js

The `netjsongraph.js`⁶ project is a Javascript library, based on the D3 framework that is able to visualize topologies expressed in the NetJSON format. This simple library is currently able to serve a dynamic output to a

⁶See <https://github.com/netjson/netjsongraph.js>.

browser out of a network description in NetJSON. It depends only on D3, which is a well known Javascript framework and it is used to quickly visualize a network topology.

While netjsongraph is extremely simple compared to other visualization libraries, it has the advantage that being based on NetJSON, any network can be visualized. If the routing protocol that is used to route packets in the network supports the generation of NetJSON output (like the OLSRd version 1 and version 2 daemons), then its output can directly be visualized in a browser. Fig. 3.3 reports an example of the output generated by netjsongraph.sj⁷

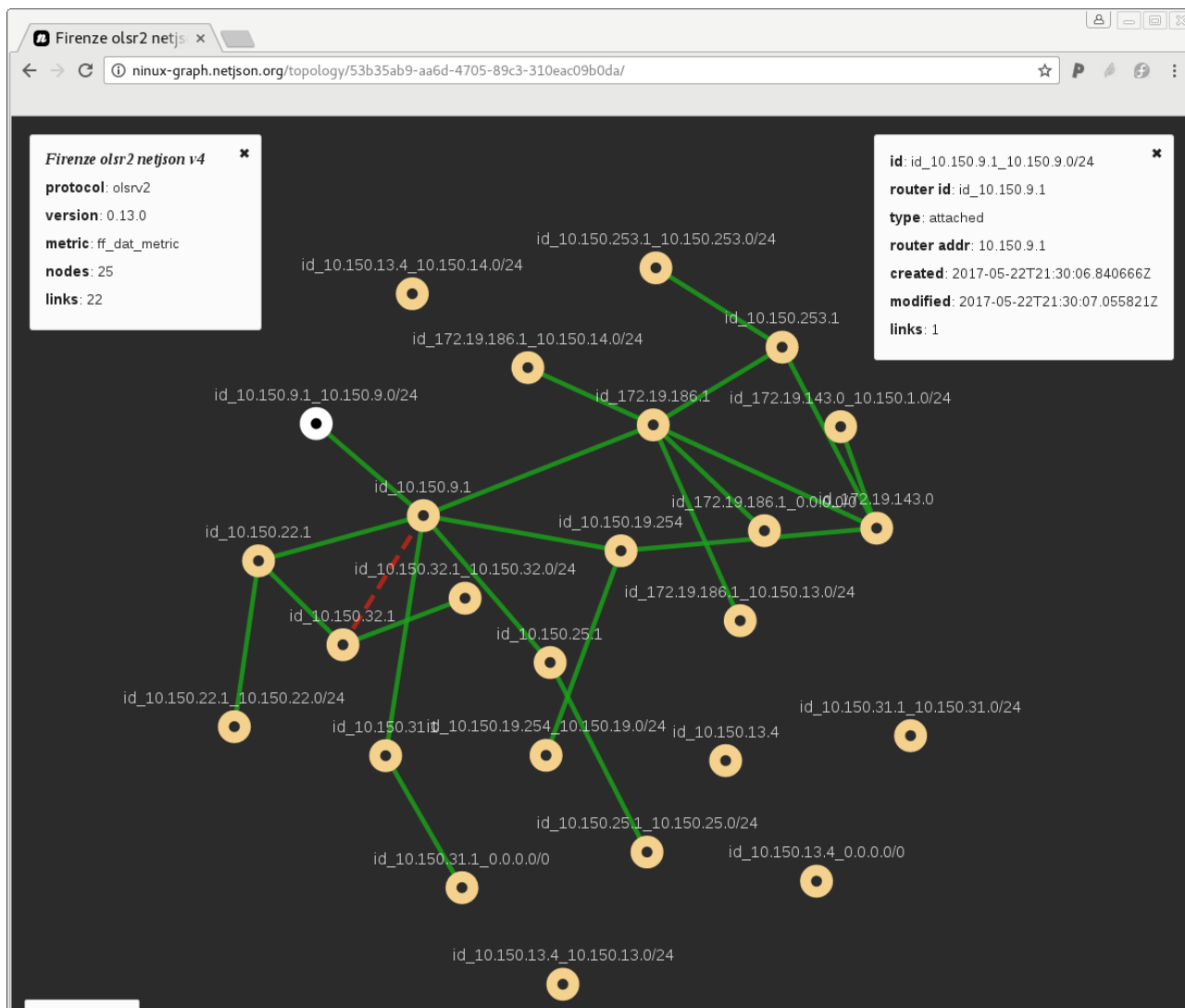


Figure 3.3: The netjson visualization of the ninux network in Florence.

Note that the map is dynamic and the box on the top right is the description of the node “10.150.9.1” on which the user clicked (top left) before taking the screenshot.

Netjsongraph is thus a convenient choice to offer network monitoring tools to community networks, because it relies on NetJSON, so in principle any network that exports its topology in the NetJSON format could be using the monitoring tools developed in T2.4.

⁷The screenshot is taken from <http://ninux-graph.netjson.org/> that is an initial attempt to export all the ninux islands in NetJSON. Note that NetJSON can be natively exported by the OLSRv2 protocol, which is currently used only in a small fractions of the networks. The support for OLSRv1 relies on a dedicated plugin and thus it is not widespread.

The functions of netjsongraph.js are currently being expanded with a series of efforts, both from the main developer (Federico Capovano) and from some students that were able to receive a Google Summer of Code scholarship. The OpenWISP project, in fact, was recognized as a valid association for the Google Summer of Code project and this way it was able to fund several developments⁸ among which:

- Adding canvas to netjsongraph.sj visualizer, in order to geolocate the nodes and visualizing them as an overlay on a map;
- Extending OpenWISP2 and django-netjsongraph.sj in order to support the visualization of mesh network topologies on OpenWISP2.

3.2.3. django-netjsongraph.js

The last component of the NetJSON ecosystem that is relevant to netCommons is the django-netjsongraph library⁹, which is a Django module to display topologies extracted by real networks using NetJSON.

Django is a well known “*high-level Python Web framework that encourages rapid development and clean, pragmatic design*”¹⁰. Django is used on tens of thousands of websites, it has a very large developer community and it is well documented, thus we will not give more details about it. Django has a modular structure, so it is easy to compose different modules to realize a web application.

Summarising the contents of the project page, some of the goals stated for django-netjsongraph are:

- Make it easy to visualize network topology data for the formats supported by netdiff¹¹;
- Expose topology data via RESTful resources in NetJSON NetworkGraph format;
- Make it easy to integrate in larger django projects to improve reusability;
- Make it easy to extend its models by providing abstract models.

In a nutshell, NetJSON is a standard language to describe networks; netjsongraph is a javascript library to visualize the topologies. django-netjsongraph adds the instruments needed to acquire the information from the NetJSON description, elaborate them and save them in a database, and finally leverage the power of Django to manage them. With django-netjsongraph a developer can, in a few minutes, set-up a network visualizer for any network, given only an URL pointing to a NetJSON file. Other formats for the topology description are obviously supported by django, but we are not interested in them.

3.2.4. The OpenWISP2 Platform

OpenWISP is an open source Network Management System (NMS) for wireless networks initially developed by the Italian CASPUR university consortium (now integrated into Cineca, a non profit Consortium made up of 70 Italian universities, 8 Italian Research Institutions and the Italian Ministry of Education) currently used by tens of public administrations in Italy and abroad¹².

OpenWISP2 is the new version of the OpenWISP project, it’s a full rewrite of the original project and its goal has now broadened, from a platform devoted mainly to the management of public networks to a generic platform for the management of wireless networks. The development of OpenWISP2 is currently undergoing and it is partly financed by Cineca, partly it is community-driven, with several developers and early users that contribute to its evolution. The interactions between the members of the community take place in the project mailing list, in the Github code repository and on several other informal channels (such as IRC chats)¹³. Since OpenWISP2 bases its terminology on a standard ISP-oriented architecture, from now on we will refer to an

⁸See <http://openwisp.org/gsoc/ideas-2017.html>.

⁹See <https://github.com/netjson/django-netjsongraph>.

¹⁰See <https://www.djangoproject.com/>.

¹¹netdiff is a library that converts topologies in several known formats into NetJSON.

¹²<http://openwisp.org/history.html>

¹³See the project mailing list <https://groups.google.com/forum/#!forum/openwisp> and Github repository <https://github.com/openwisp>

hypothetical “network owner” to describe OpenWISP2 and its dynamics. Further on we will then explain and detail how it can be easily tailored for CNs.

A NMS is an instrument that is used to manage large enterprise networks, and it is made of a specialized program running on each node and a centralized manager. The manager is aware of the presence of the nodes, and contains a directory of configurations for each of them. The nodes periodically poll the manager asking for updates on their configuration, and when an update is present, the manager pushes the update to the nodes. As a simple example consider the case in which the owner wants to change the name of the wireless network (the ESSID exposed in beacon frames, in IEEE 802.11 terminology). Without an NMS he should reconfigure singularly each wireless router, while the NMS makes it possible to change the configuration in one single place and have it deployed on all the routers. This is a very simple example of what can be done with a NMS, which in general can be used also as a diagnostic tool and for user management.

To use OpenWISP2 the owner will have to replace the firmware of the wireless nodes he intends to use (this will not be strictly necessary anymore in the future for products of some supported brands) with a specially crafted version of OpenWRT/LEDE that includes the `openwisp-config` daemon. The daemon is configured to periodically poll the manager, which stores a set of configurations that can be used on the wireless nodes. Several configurations templates that can be used to manage the nodes of a CN are already present in OpenWISP2, and that’s why some islands of the ninux community are already using OpenWISP2 to manage their network.

As Fig. 3.4 shows OpenWISP2 is a web-based software developed using the Python language and the Django web framework. OpenWISP2 can be installed on a virtual machine using Ansible¹⁴, for which a specific role is available. At the time of writing OpenWISP2 has a management back-end that is used by the network owner to modify the configurations of each node, and a minimal diagnostic interface that tells if the node is currently active (it is polling the manager) and has successfully applied the last configuration update. What is currently missing is a public front-end that can expose the network status and additional information, similarly to what NodeShot does (see Fig. 3.1).

Regardless of its current early state of maturity the project is growing, there are 271 members in the official mailing list that is pretty active¹⁵, the code updates are frequent, there were about 967 installations of the related Ansible role¹⁶ and one public administration that is already using it to manage a running network¹⁷. This attention is expected to increase as soon as the product stabilizes and the various users that currently use the older version will switch to the new product.

OpenWISP2 obtained this year 5 students grants from the Google Summer of Code project to develop features that are useful for CNs. OpenWISP2 also become part of the Google Code-In project, which helped the visibility of the project.

3.2.4.1. OpenWISP2 in Ninux

Being OpenWISP2 open source, the ninux community is experimenting its use for the management of the network nodes. The lead developer of OpenWISP2 is also part of the ninux community network and the lead developer of the NodeShot platform, which he intends to replace with OpenWISP2 as soon as a public front-end will be ready. The currently missing features that are necessary in order to use OpenWISP2 in the context of a CN are mainly two, the first is a network visualizer that also integrates geographical information and ownership information, as it is the case for NodeShot. The second is a way to implement multi-tenancy in the network manager, that is now intended to be used by a single owner to manage the whole network. This is clearly not the case of a CN in which all users have the permission to manage only their own nodes.

The realization of these features are currently undergoing, and will be based on the NetJSON standard, the

¹⁴See <https://www.ansible.com/>

¹⁵See <https://groups.google.com/forum/#!aboutgroup/openwisp>

¹⁶<https://galaxy.ansible.com/openwisp/openwisp2/>

¹⁷<https://twitter.com/openWISP/status/826763786728140801>

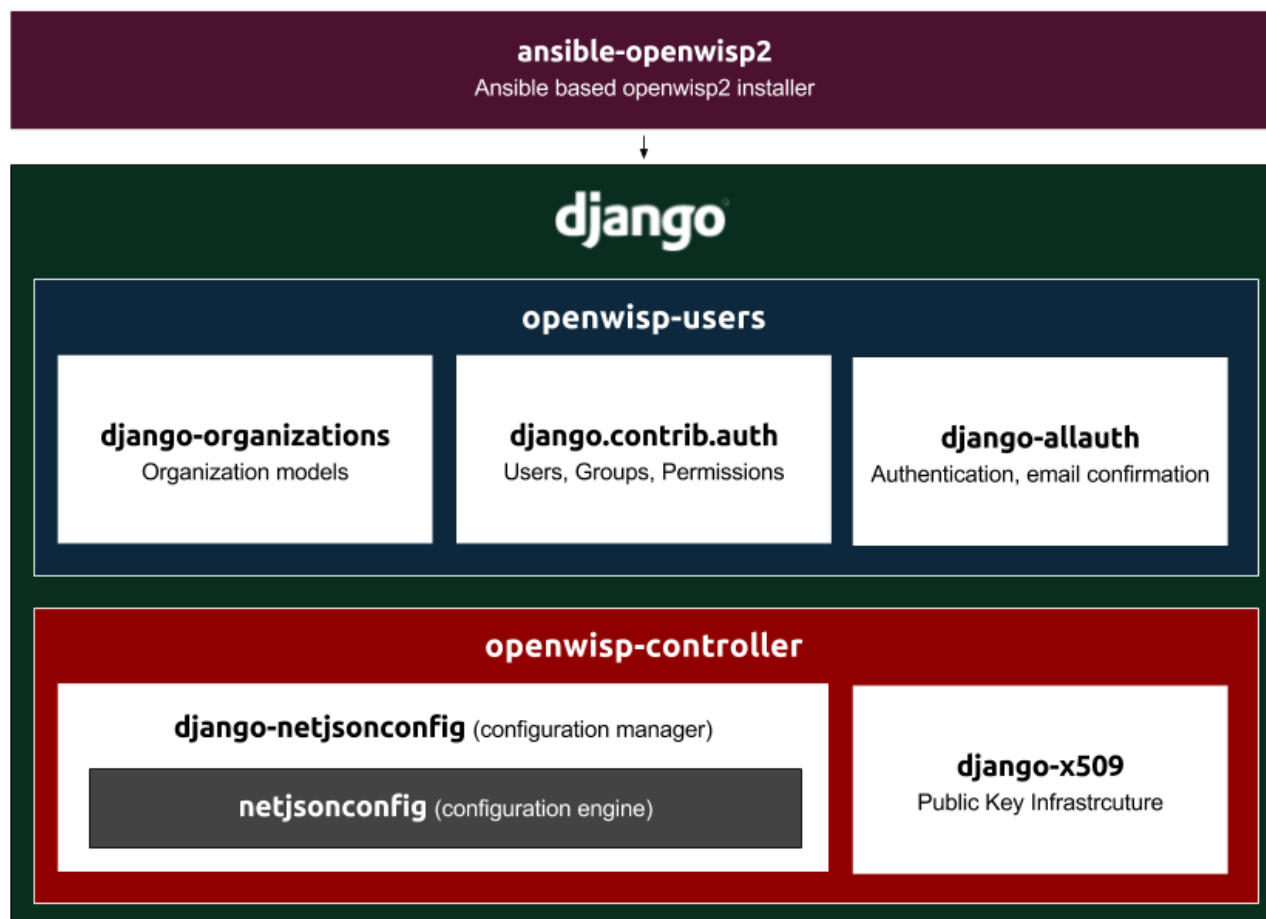


Figure 3.4: The components of the OpenWISP2 platform, taken from the github project page.

netjsongraph.js library and the django-netjsongraph module. Thus, it is important to follow the development of these modules and to contribute to their advancement.

3.3. Developments in the Period M12-M24

After an initial analysis and interaction with the ninux community and Federico Capovano we decided that it would have been certainly more rewarding, both for the outcome of the project and for the ninux communities to integrate the functions developed in T2.4 in the OpenWISP2 code base rather than pushing them in the NodeShot platform or creating another monitoring platform. Even if currently the OpenWISP2 platform is not ready for production, in the future it will replace NodeShot and will be used potentially by tens of other wireless networks, small WISPs and community networks. We believe that OpenWISP2, being an innovative solution not only for CNs but also for the market (in fact, several private initiatives use OpenWISP2 in their own networks) and being developed also with the funding coming from a public administration has higher chances of success than the other currently available management platforms for CNs. As OpenWISP2 is not designed from scratch for a specific network we believe it has higher chances to be applied not only in the CN context but also in the context of other similar initiatives.

This choice obviously comes with some consequences, which can be summarized in the lack of some features that are needed to reproduce in a production environment all the results obtained in D2.5. We refer to, in particular, the lack of data on the ownership of nodes (as already explained) and the lack of access to the data coming from the social network. These two limitations do not let us perform the social analysis in OpenWISP2

which was made for D2.5, and restricts us to the topological analysis of the network for the time being. We opted for the following approach:

- We realized a Python module that analyses the robustness of a given network, and produces another network graph that is different from the original one because it compresses parts of the graph that are not critical, while putting in evidence those nodes that are instead critical ones. This module is named `netjson_robustness` and is freely available in the Github repository of the project¹⁸;
- We extended the `netjsongraph.js` library in order to support a better visualization of the elaborated network graph. This required modifications to the visualization of the graph and it is also available online¹⁹;
- We extended the functionalities of the `django-netjsongraph` python module to support the possibility of showing the same network graph with more than one “visualization”. This feature required modifications in the database model and in the visualization of `django-netjsongraph`, and are available online²⁰.

This approach, which will be better detailed in Chapter 4, allowed netCommons to reach two very important goals:

- Produce an operational modification to OpenWISP2 that can be used by CNs and small WISPs to understand at a glance the robustness properties of their network. This feature was well received from both the OpenWISP2 main developer and some ninux activists. As a confirmation of this, we are in an active discussion within the OpenWISP2 community to let our feature become part of the next stable version of OpenWISP2, as testified in the letter included in Appendix B;
- Produce a flexible instrument that supports a generic number of visualizations, which can be extended in the future when OpenWISP2 will support multi-tenancy, integration with the communication platforms and geolocation of nodes.

Of course, this approach required us a large effort to understand and contribute to an already existing platform, instead of creating our own. OpenWISP2 uses the Django Python framework, which is extremely powerful and versatile, but has a steep learning curve. It requires adherence to the conventions of the project and can not allow the flexibility of a fully home-brew project. Furthermore, producing code for a professional platform requires a level of commitment and attention that is different from a research prototype. We had to spend a considerable amount of time performing tests that are necessary to show to the project maintainer that the code is stable and behaves properly even in situations that are typical of a production system (code must be efficient, resilient etc.). These features can receive less attention when the goal of the source code is *experimenting*.

We opted for this choice because we wanted to give more importance to the direct impact on the community, while maintaining the possibility to extend the system with more features (more visualizations) when OpenWISP2 will support a larger information base.

¹⁸See https://github.com/netCommonsEU/netjson_robustness

¹⁹See https://github.com/netCommonsEU/netjsongraph.js/tree/robustness_graph

²⁰See <https://github.com/netCommonsEU/django-netjsongraph>

4. Developments on OpenWISP2

We will describe here the developments done in three modules, the `netjson_robustness`, `django-netjsongraph`, and the `netjsongraph.js` modules. All the developed code is available in the github repository of the project¹, together with the rest of the code developed in WP3, please see the netCommons website for guidance².

4.1. The `netjson_robustness` library

OpenWISP2 includes a feature that allows the network administrator to visualize the network topology, as shown in Fig. 3.3. As long as the network is of a size similar to the one in the figure, the administrator can easily spot critical nodes. When the network grows, this is impossible to do at a glance, and must be done algorithmically. In D2.5 we used some network metrics (centrality and robustness metrics) that help us to measure the importance of nodes. Part of these metrics and the original code was re-used in the `netjson_robustness` python module.

Listing 4.1: `config.cfg` source file

```
from distutils.core import setup
setup(
    name='netjson_robustness',
    packages=['netjson_robustness'],
    version='0.1.1',
    description='A library to perform some robustness analysis on '
                'NetJSON-defined graphs',
    author='Leonardo Maccari',
    author_email='maccari@disi.unitn.it',
    url='https://github.com/netCommonsEU/netjson_robustness',
    download_url='https://github.com/netCommonsEU/netjson_robustness/'
                'archive/0.1.tar.gz',
    keywords=['NetJSON', 'graph analysis', 'robustness'],
    install_requirements='networkx',
    classifiers=[],
)
```

The `netjson_robustness` module manipulates a graph expressed in the NetJSON format and transforms it in another NetJSON graph in the *block-cut tree* shape. Any graph can be transformed in a block-cut tree made of two kinds of nodes:

- Cut-points: nodes that if removed from the network, will partition the network in two or more components
- Blocks: sub-graphs of the original graph in which the removal of one node will not partition the network. In the new graph, blocks are compressed in one single node.

¹See <https://github.com/netCommonsEU/>.

²See <https://www.netcommons.eu/?q=content/open-source-software>.

`netjson_robustness` is a compact library that transforms a graph into an annotated block-cut tree. Each cut-point receives an attribute that expresses the fraction of nodes that will be cut from the network if this node fails. Blocks receive an attribute that expresses their size. These attributes are needed to produce the right visualization using `netjsongraph.js`

`netjson_robustness` is the only module developed from scratch for the project, it contains a set of tests that can be run using the canonical `python -m pytest` and a `python.cfg` file as shown in Listing 4.1.

The `cfg` file makes it possible to upload the file to the Pipy packages repository, which was actually done using the appropriate Pipy tools³, so that from anywhere the module can be installed as easily as typing “`pip install netjson_robustness`”.

4.2. Modifications to the `netjsongraph.js` Module

`netjsongraph.js` is a Javascript library that takes in input a NetJSON graph and visualizes it in the SVG format, which can be shown in any recent browser. In order to ease the visualization of the elaborated topology, we extended the code and we modified the CSS used by default, to clarify the block-cut tree and to change the kind of information displayed to the user.

Fig. 4.1 and Fig. 4.2 show the final results of the elaboration.

The difference is evident, in the original visualization the network appears flat and the differences among the nodes are not visible. In the second visualization the blocks are very clearly shown (the blue circles) and make it evident which part of the topology has more “weight”. The cut-points are instead shown with a gamma of colors, from pale orange to red which indicates how critical they are, that is, the number of nodes that will be disconnected from the network if that node fails (the same metric used to express the robustness of the network in the code used in D2.5). Fig. 4.3 shows that when a node is clicked, the map shows this information.

During the development of the library, OpenWISP2 received attention from another well known community, the AWMN community of Athens⁴. The members of AWMN exported the topology of their network in NetJSON and we were able to produce visualization of their network.

Figs. 4.4 and 4.5 show the original and the block-cut tree visualization of the AWMN network. Apparently, the AWMN network seems to be more robust than `ninux`, it has 443 nodes out of which 145 in a single block. We do not indulge in the analysis of this topology as we did not have time to verify the data, we report this interaction with AWMN to stress that indeed OpenWISP2 and NetJSON is gathering the interest of communities beyond `ninux`.

³See https://pypi.python.org/pypi/netjson_robustness/0.1.1.

⁴See the discussion on <https://github.com/netjson/django-netjsongraph/issues/14#issuecomment-315514656>.

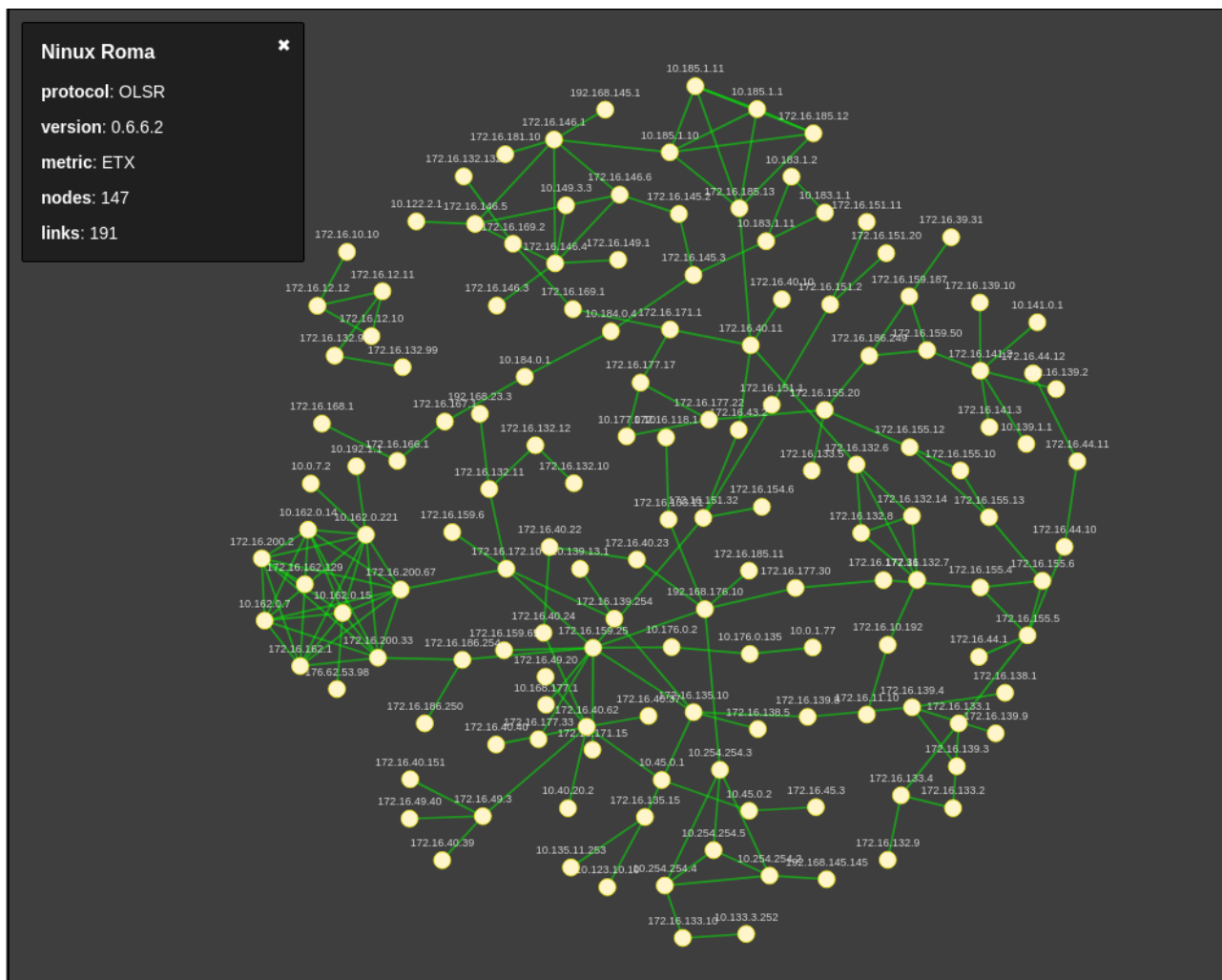


Figure 4.1: The standard visualization of the OpenWISP2 platform

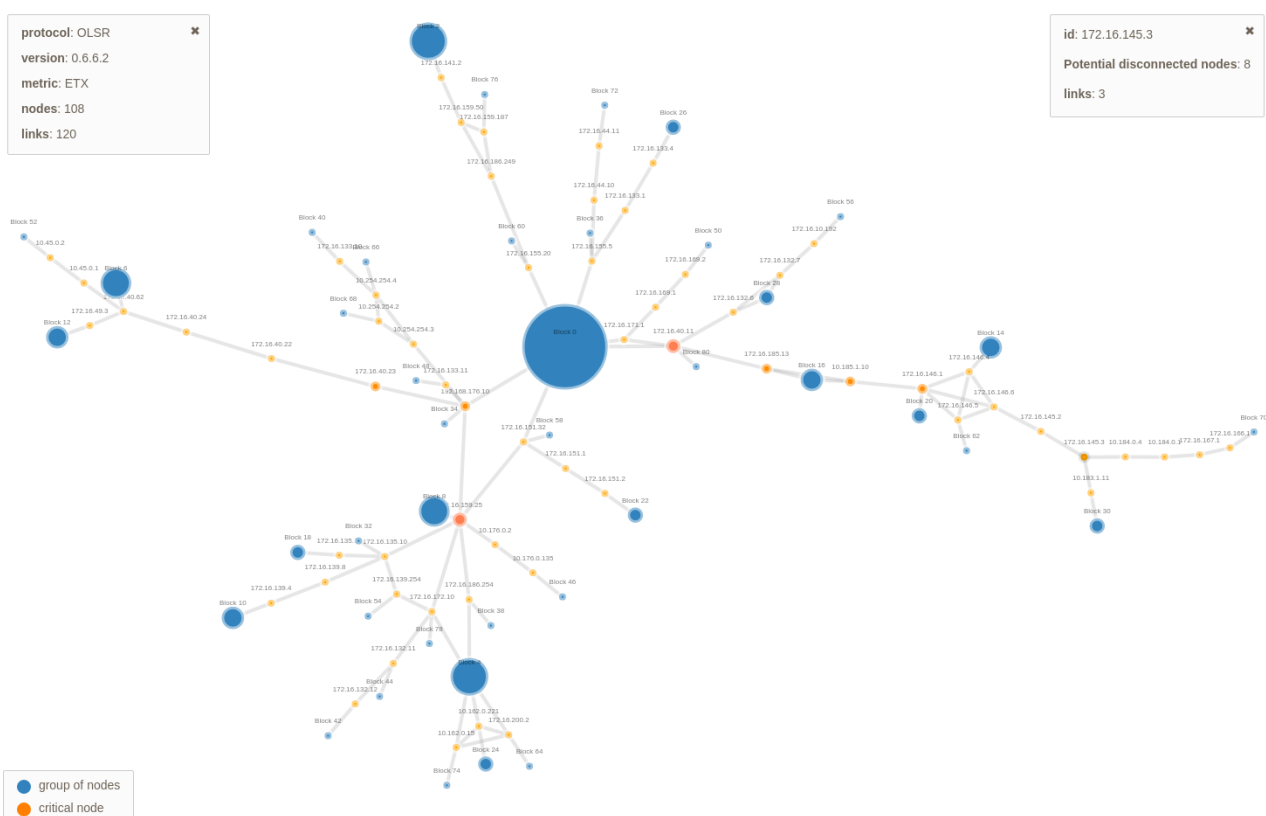


Figure 4.2: The new visualization given by the use of netjson_robustness and netjsongraph.js.

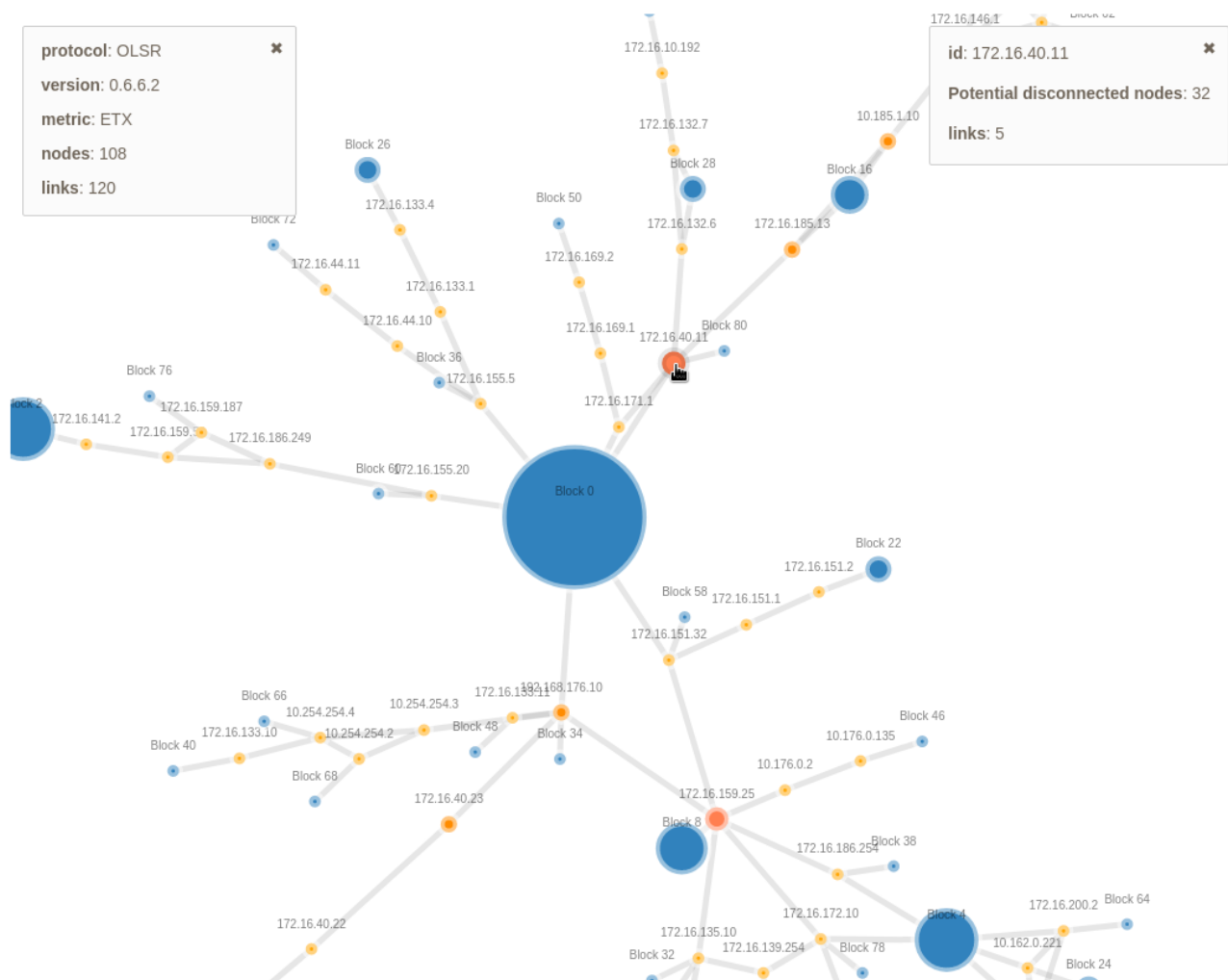


Figure 4.3: Detail of the information displayed when clicking on a cut-point.

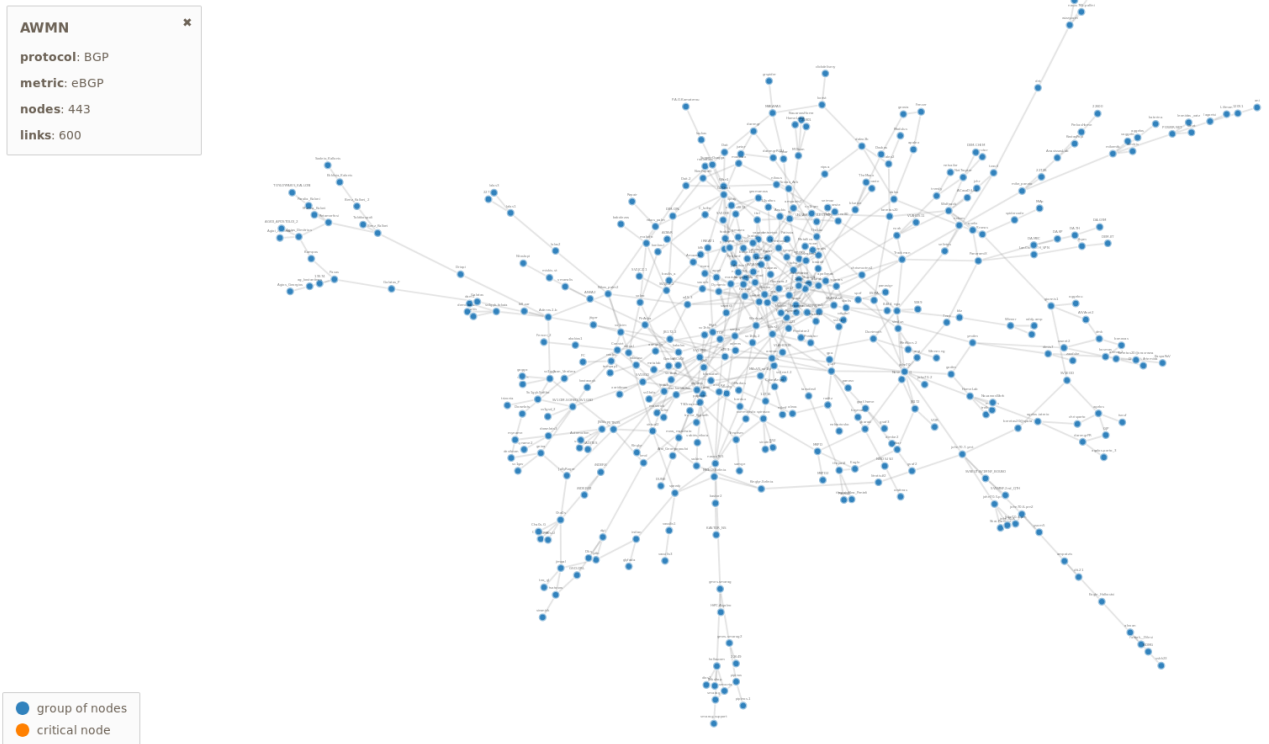


Figure 4.4: The AWMN network.



Figure 4.5: The block-cut tree visualization of the AWMN network.

4.3. Modification to `django-netjsongraph`

The developments we described so far were done to some of the single components that make OpenWISP2 but need to be integrated with the whole platform. `django-netjsongraph` is the component of OpenWISP2 that takes care of loading a topology, and embeds `netjsongraph.js` which will visualize it. The modifications we did to `django-netjsongraph` are substantial, even if the final result to the user is the addition of a single button to the interface, shown in Fig. 4.6.

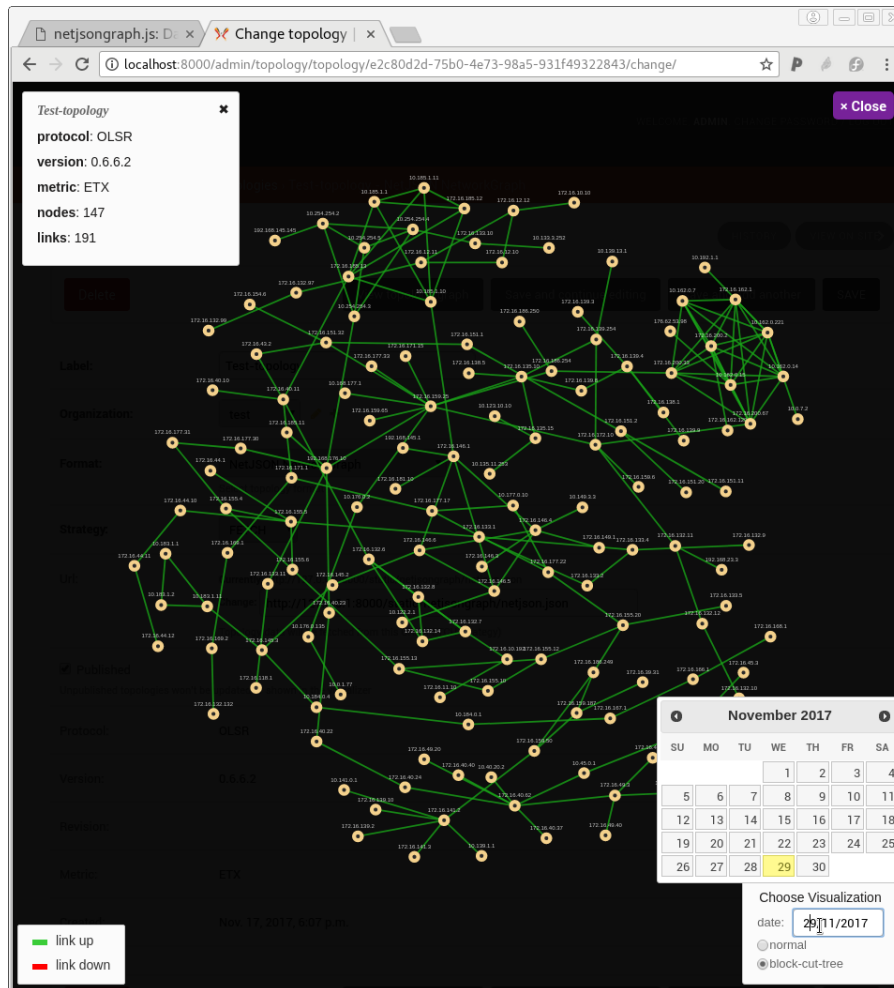


Figure 4.6: Integration of the different visualization in OpenWISP2

Fig. 4.6 shows the user interface that OpenWISP2 presents to the user when visualizing the topology. This is the typical topology we already are familiar with, and a small calendar that can be used to pick from the database of saved snapshots, one snapshot that the user wants to see. Our modification resides in the addition of a radio-button below the calendar with which the user can choose if the graph to display will be the normal one or the one in block-cut tree format.

Albeit this may seem a minimal modification, it required work to modify some of the components of `django-netjsongraph`. In fact, now `django-netjsongraph` depends on `netjson_robustness` for the elaboration of the graph from the normal one to the block-cut tree. If the network is large this may require an amount of time that is not compatible with the usability of a web platform (seconds). Since this transformation can not be done on the fly, we had to change the model of OpenWISP2, implementing an extension to the database to periodically save visualizations of the topology together with the snapshots. The structure of Django, albeit it requires a large effort to be understood and to take practice with it, afterwards

it helps to integrate our modification in OpenWISP2 in a reasonable time with small modifications to the original code. For the time being, since we have only one possible visualization, we used a radiobutton in the user interface. In the future, if further visualizations will be made, this will be changed to a drop-down menu.

Note that, given the current architecture of the code, new visualizations are really easy to produce. The needed steps are the following ones:

- Modify the `netjson_robustness` module in order to produce a new variant of the graph, and save it in NetJSON format in the database as a new visualization;
- Eventually modify `netjsongraph.js` to optimize the CSS for the new view. The CSS is chosen at runtime;
- Add the new visualization to the OpenWISP2 controller.

These easy steps can be done when new features are added to OpenWISP2. For instance, now the multi-tenancy feature is only roughly implemented. In the future, a user will be able to log-in, add to the manager their own node and assign it to a node in the topology (which is periodically downloaded from one of the nodes that is controlled by OpenWISP2). When this feature will be done, it will be possible to visualize also a graph annotated with the *owner robustness* metric that was introduced in D2.5.

5. Impact on the ninux Community

Several actions, and interactions, were undertaken during the course of T2.4 in order to share the results with the ninux community, which was our principal partner for the development of the monitoring tools and for their application. Among them, we mention the participation of Leonardo Maccari to two ninux-day events (the aperiodic meeting of the ninux community) and the continuous interactions with at least three ninux islands, the island of Rome (in particular with Federico Capoano, the maintainer of OpenWISP2), Florence, and ninux Calabria. This latter interaction is more related to WP4, as they were interested mostly in the legal implications and it is documented in more detail in Deliverable D4.2 [5].

The interaction with netCommons materialized in several ways, testified by the letter of appreciations that we attach to this report in Appendix B. In particular:

- The interactions with Federico Capoano led to the developments we documented in the report, and helped him and the OpenWISP2 community to adopt new features and gain visibility. Moreover, the continuous interactions with Leonardo Maccari helped to shape two project proposals for the expansion of the NetJSON ecosystem. Albeit these proposals were not financed, the interactions with opened the way for the OpenWISP2 maintainer to try new forms of funding in order to make OpenWISP2 sustainable.
- The interactions with the ninux community in Florence led to the adoption of the OpenWISP2 platform to monitor the network, and again, to a project proposal submitted to the RIPE community project grants¹. The ninux community of Florence recognized the need to broaden the participation to the group and to include different people, in order to make the network less hierarchical. This was motivated by the fragility observed in the network and in the community. The project submitted (and currently under evaluation) deals with two themes, one of which is the development of modules for OpenWISP2 that will ease the interaction among people and the horizontal participation to improve the sustainability of the network and the community. Interestingly, the ninux community of Florence, in order to broaden its chances of success decided to include in the project proposal also the Associazione Ricreativa e Culturale Italiana (ARCI), the largest Italian Non-Governmental Organization (NGO)². ARCI works in the field of social inclusion, and can help ninux in several ways, from practical issues (making spaces available for new nodes) to diversifying the people participating to the community.

The two letters of appreciation that we include in Appendix B are one from Federico Capoano, and one from ninux Florence, undersigned by one person from ninux and one from ARCI. Besides the appreciation brought by the letter, it is clear from the discussion in this deliverable that the interaction with netCommons, which represent a more structured and mature way of locating CNs into society and inform communities (in this case ninux) themselves on a different perspective of the work they are doing and its consequences and potential impact, can easily spawn a course of action that can make CNs more aware of what they actually are (monitoring and measuring with a scientific approach the characteristics of the network), and thus take measure or counter-measures to change some of their habits and behaviors toward a more sustainable structure.

¹RIPE stands for “Réseaux IP Européens” and is the European body for the assignment of IP addresses, see <https://www.ripe.net/support/cpf>

²See www.arci.it.

6. Conclusions

This reports concludes the work of T2.4 on the development of open source instruments to monitor the sustainability of a community network. Part of this work resided in the definition of multi-layer metrics that can help to spot points of failure in the network graph and in the social network graph of a CN. This work produced publications and enabled the second part of the work, which was to translate this research into usable code for the community to monitor their evolution.

In this second phase, a decision was taken to prefer direct and long-term impact over the production of immediate research results. In a way, it would have been much easier to continue with the research work started with D2.5, enlarge the monitoring period, update the data-set and make everything converge in a home-brew web application showing all the data in a friendly way. This choice would have been short-sighted as such stand-alone platform would have had little chances of staying up-to-date with the evolution of the CN. As an example, the communication tools that the community use change with time, the ninux mailing list in the last year was substantially abandoned in favour of a few Telegram chats, that today carry the large majority of the conversations. If we had insisted with a custom instrument to visualize the results of our research done in D2.5, at the time of writing, this tool would already be outdated.

Instead, we chose a different path, less rewarding in terms of research results but more rewarding in terms of impact on the community. We chose to adapt our methods to a promising instrument that the community is developing, even if right now this instrument is not stable, even if it required an amount of work that is not scientifically productive and even if we can not implement all the features in this platform. The advantage is that OpenWISP2, due to strong effort in the standardization of its components, plays the role of an abstraction layer, on top of which we can place visualizations of our custom metrics computed of the data models. As a consequence the software that we produced will not become *abandonware* as soon as the project ends, and using our modular design it will be easy to add new features.

It is in the interest of this task to keep monitoring the development of OpenWISP2 and help introducing new features even after the end of T2.4. These new features will be mirrored in the Github repositories of the project and eventually documented on a small update of this deliverable to be published on the project website.

Bibliography

- [1] L. Maccari and R. Lo Cigno, “Monitoring Instruments for CNs (v1),” netCommons deliverable D2.5, Dec. 2016. <http://netcommons.eu/?q=content/monitoring-instruments-cns-v1>
- [2] L. Navarro, R. Baig, F. Freitag, E. Dimogerontakis, F. Treguer, M. Dulong de Rosnay, L. Maccari, P. Micholia, and P. Antoniadis, “Report on the Existing CNs and their Organization (v2),” netCommons Deliverable D1.2, Sept. 2016. <http://netcommons.eu/?q=content/report-existing-cns-and-their-organization-v2>
- [3] L. Navarro, A. Lertsinsruttavee, V. Chryssos, C. Rey-Moreno, S. Luca de Tena, C. Conder, L. Annison, E. Huerta, F. Tréguer, L. Maccari, and R. Srivastava, “Report on the Governance Instruments and their Application to CNs (v2),” netCommons Deliverable D1.4, Jan. 2018. <https://netcommons.eu/?q=content/report-governance-instruments-and-their-application-cns-v2>
- [4] C. Fuchs, M. Michalis, and D. Boucas, “The Multiple Aspects of Politics of Sustainability in Community Networks: Definitions, Challenges, and Countermeasures (v2),” netCommons Deliverable D2.2, Jan. 2017. <http://netcommons.eu/?q=content/multiple-aspects-politics-and-sustainability-cns-definitions-challenges-and-countermeasure-0>
- [5] F. Giovanella, M. Dulong de Rosnay, A. Messaud, and F. Tréguer, “European Legal Framework for Community Networks (CNs) (v2),” netCommons Deliverable D4.1, Dec. 2017. <http://netcommons.eu/?q=content/european-legal-framework-cns-v2>
- [6] L. Maccari, “On the Technical and Social Structure of Community Networks,” in *The First IFIP Internet of People Workshop, IoP*, Vienna, Austria, May 20 2016.
- [7] L. Maccari and R. Lo Cigno, “A Week in the Life of Three Large Wireless Community Networks,” *Ad Hoc Networks*, Elsevier, vol. 24, Part B, pp. 175–190, Jan. 2015.
- [8] L. Baldesi, L. Maccari, and R. Lo Cigno, “Improving P2P streaming in community-lab through local strategies,” in *10th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Larnaca, Cyprus, Oct. 8-10 2014, pp. 33–39.
- [9] L. Maccari, L. Baldesi, R. Lo Cigno, J. Forconi, and A. Caiazza, “Live Video Streaming for Community Networks, Experimenting with PeerStreamer on the Ninux Community,” in *Proceedings of the 2015 Workshop on Do-it-yourself Networking: An Interdisciplinary Approach (DIYNetworking '15)*, May 22, 2015.
- [10] S. Traverso, L. Abeni, R. Birke, C. Kiraly, E. Leonardi, R. Lo Cigno, and M. Mellia, “Neighborhood Filtering Strategies for Overlay Construction in P2P-TV Systems: Design and Experimental Comparison,” *IEEE/ACM Transactions on Networking*, vol. 23, no. 3, pp. 741–754, June 2015.
- [11] L. Maccari, N. Facchi, L. Baldesi, and R. Lo Cigno, “Optimized P2P streaming for wireless distributed networks,” *Pervasive and Mobile Computing*, vol. 42, no. Supplement C, pp. 335 – 350, Dec. 2017.
- [12] L. Maccari and R. Lo Cigno, “Pop-routing: Centrality-based tuning of control messages for faster route convergence,” in *35th Annual IEEE International Conference on Computer Communications (INFOCOM)*, San Francisco, CA, USA, April 10-14, 2016 2016, pp. 1–9.
- [13] L. Maccari, Q. Nguyen, and R. Lo Cigno, “On the computation of centrality metrics for network security in mesh networks,” in *IEEE Global Communications Conference, (GLOBECOM)*, Dec. 2016.
- [14] L. Baldesi, L. Maccari, and R. Lo Cigno, “On the Use of Eigenvector Centrality for Cooperative Streaming,” *IEEE Communications Letters*, vol. 21, no. 9, pp. 1953–1956, Sept. 2017.

-
- [15] L. Maccari, L. Ghio, A. Guerrieri, A. Montresor, and R. Lo Cigno, “On the Distributed Computation of Load Centrality and Its Application to DV Routing,” in *36th Annual IEEE International Conference on Computer Communications (INFOCOM)*, Apr. 2017 – To Appear.
- [16] M. Barthélemy, “Spatial networks,” *Physics Reports*, vol. 499, no. 1, pp. 1–101, 2011.
- [17] R. Louf, P. Jensen, and M. Barthelemy, “Emergence of hierarchy in cost-driven growth of spatial networks,” *Proceedings of the National Academy of Sciences*, vol. 110, no. 22, pp. 8824–8829, 2013. <http://www.pnas.org/content/110/22/8824.short>
- [18] C. Fuchs, M. Michalis, and D. Boucas, “The Multiple Aspects of Politics of Sustainability in Community Networks: Definitions, Challenges, and Countermeasures (v1),” netCommons Deliverable D2.1, June 2016. <http://netcommons.eu/?q=content/multiple-aspects-politics-and-sustainability-cns-definitions-challenges-and-countermeasures>
- [19] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, no. 5439, pp. 509–512, 1999.
- [20] B. M. Waxman, “Routing of Multipoint Connections,” *IEEE journal on Selected Areas in Communications*, vol. 6, no. 9, pp. 1617–1622, 1988.
- [21] L. Li, D. Alderson, W. Willinger, and J. Doyle, “A first-principles approach to understanding the internet’s router-level topology,” in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004.

A. Spatial Separation

The spatial separation metrics were partly modified from their original definition in [17] due to the different context in which we use them. The first modification is due to the fact that CNs are not tree-shaped but they are undirected graphs (a functional Wi-Fi link must be able to transmit in both directions, so there is no need to use directed edges to represent the network graph, albeit, link performance can be asymmetric). As such, in the ninux graph there is no root node that is naturally identifiable. We repeated the measures with three choices for the root node, selecting the node that maximises the following metrics: closeness centrality, betweenness centrality, and eccentricity. All the choices yield qualitatively similar results even if they identify different root nodes. Results are reported in Tab. A.1.

<i>level</i>	s_l (<i>eccentricity</i>)	s_l (<i>closeness</i>)	s_l (<i>betweenness</i>)
0	-	-	-
1	-	0.93	-
2	0.93	0.86	0.93
3	0.95	0.99	0.95
4	0.90	0.88	0.90
5	1.00	-	1.00
6	0.80	0.99	0.80
7	-	-	1.0

Table A.1: Average separation within each level with various choices of root node. Dash means that there are less than 2 zones in the level.

The second difference is in the definition of Eq. (2.1). In a tree v_i has by definition only a neighbor with a lower level (its parent in the tree) and all the other neighbors have a higher level (its descendants in the tree). In a graph, among the neighbors there can be also nodes with the same level, and it is important to consider these links, or else separation is artificially increased. Thus we modified 2.1 as follows:

$$N'(i) = \{v_j \mid v_j \in N(i) \wedge l(v_j) \leq l(v_i)\}. \quad (\text{A.1})$$

Finally, we modified the original definition of the influence zone in order to better reflect the behaviour of a CN. In the original definition the zone for node i is defined as “the circle centered on the barycenter of i ’s neighbours that belong to the next level, of radius the maximum distance between the barycenter and those points”[17]. As we said, in CN long links are realized with directional antennas, thus, there is not a well-defined concept or “radius” of the influence zone. Consider Fig. A.1 in which the node for which we compute the interest zone is the red one (v_i), and the yellow nodes are the nodes in $N'(i)$. The dashed circle is the influence zone as per the original definition, and the dotted polygon is the hull envelop of the points. The area marked with the letter B and C are areas for which there is no assurance that v_i has any coverage, since there is potentially no antenna pointed in their direction. Area A extends beyond an existing node, and in that direction the line of sight may be obstructed by obstacles. Consequently, for its application to CNs, the convex hull of the area including $N'(i)$ is a more realistic choice than the original definition.

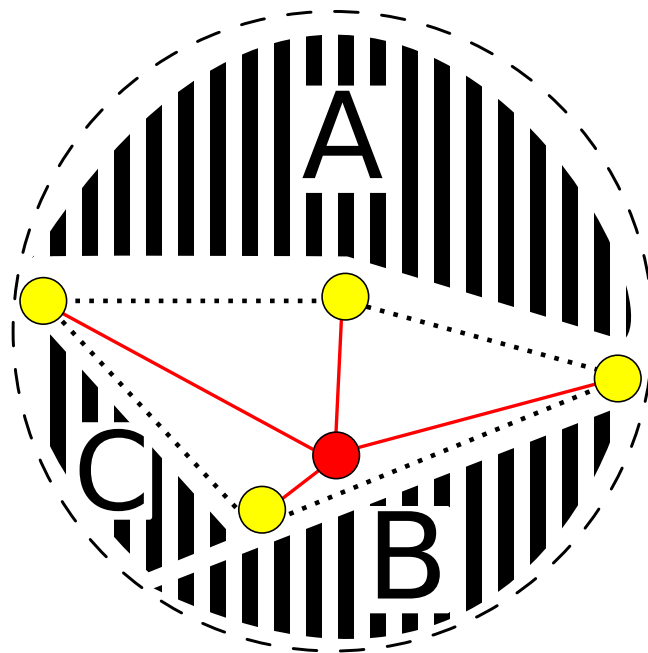


Figure A.1: An example definition of influence zone.

B. Appreciation Letters

The letters reported here testify the two long-listing interactions between ninux and netCommons researchers. The first one relates to the development of OpenWISP2, while the second one to the adoption of OpenWISP2 and active monitoring tools in the Florence island of ninux.

Rome 11/12/2017

To the European Commission, to the evaluators of the netCommons project, to whom it may concern.

My name is Federico Capoano, I am a member of the Rome ninux.org community network and the lead developer of the OpenWISP2 software platform.

OpenWISP2 is a network management system tailored at small Wireless ISPs and community networks, it is open source and it is gathering attention from a community of developers, practitioners and volunteers.

I started interacting with netCommons through Leonardo Maccari, when I got to know about the work he is doing on the analysis of the robustness of ninux. Both as a ninuxer and as a developer of OpenWISP I was interested in his findings.

We started a fruitful interaction in which I suggested that he modified his initial plans to integrate his code and metrics to the NodeShot platform, and integrate it into OpenWISP2. I monitored his efforts during the development and I can conclude that the results are a great contribution to the OpenWISP2 project. The new visualization features are an interesting add-on that can be currently used to analyse the communication network robustness, but will be further expanded in order to take into account other network and community parameters.

His contributions are currently under testing in the OpenWISP2 framework and I plan to integrate them into the master branch hopefully before the end of the year.

The interaction with Leonardo was useful also in other ways, together with him, we prepared two proposals for experimentations and extension of the OpenWISP platform. He guided me through the process we already finalized one of them, which unfortunately was not accepted, but is at the base for another one, currently under preparation. Leonardo and netCommons are a valuable partner for such activities, which will play an important role in the future development of OpenWISP2.

Federico Capoano



Florence 11/12/2017

To the European Commission, to the evaluators of the netCommons project, to whom it may concern.

Our names are Salvatore Moretti and Marzia Frediani, respectively one of the founders of ninux in Florence and the vice-president of ARCI Florence.

Ninux.org is an Italian Wireless Community Network, whose goal is the creation and expansion of a free, open and experimental network, on the model of a wireless mesh network. Ninux is a distributed community, with several "islands" around the whole country that sum up to hundreds of wireless nodes. Ninux is primarily a vibrant community of people with an interest in technology and digital rights, ninux Florence is the island based on the Florence city, in Tuscany.

ARCI is the largest Italian NGO (www.arci.it) with more than one million members. ARCI works in the promotion of culture and solidarity, peace, rights and democracy. ARCI Florence is the largest of the local ARCI sections, with 45.000 members divided in 250 chapters in the province of Florence.

Ninux Florence and ARCI cooperate, as the ninux Florence network has one node in an ARCI building and, and regular meetings are held there.

Thanks to the continuous interaction with Leonardo Maccari, and the results (in terms of documentation, source code and support) that we received from the netCommons project (presented during the "ninux day", during personal meetings and on the project website) ninux Florence decided to adopt the OpenWISP2 open source monitoring system and, together with ARCI to participate to an open call for a RIPE funding (<https://www.ripe.net/support/cpf>).

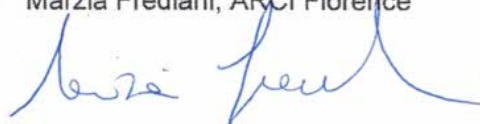
We, ninuxers, believe that diversity is important to make a network sustainable, and we observed patterns of unsustainability in our network, both under the technical point of view (as our network is surely fragile) and under the social point of view (as we need more participation and diversity). Ninux is a community primarily made of hackers and tech-savvy people, and thus, it strives to open up to the broader society, ARCI is instead fully rooted into society but does not generally treat communication technologies among its core themes. Therefore, ninux and ARCI decided to participate to this project to complement each other's skills, and relaunch our activity. If financed, the project will give us the chance of improve OpenWISP in order to include community-management features (communication, task sharing etc.) that can help us grow the community in a sustainable way. Both the signers of this letter are among the key personnel the submitted project.

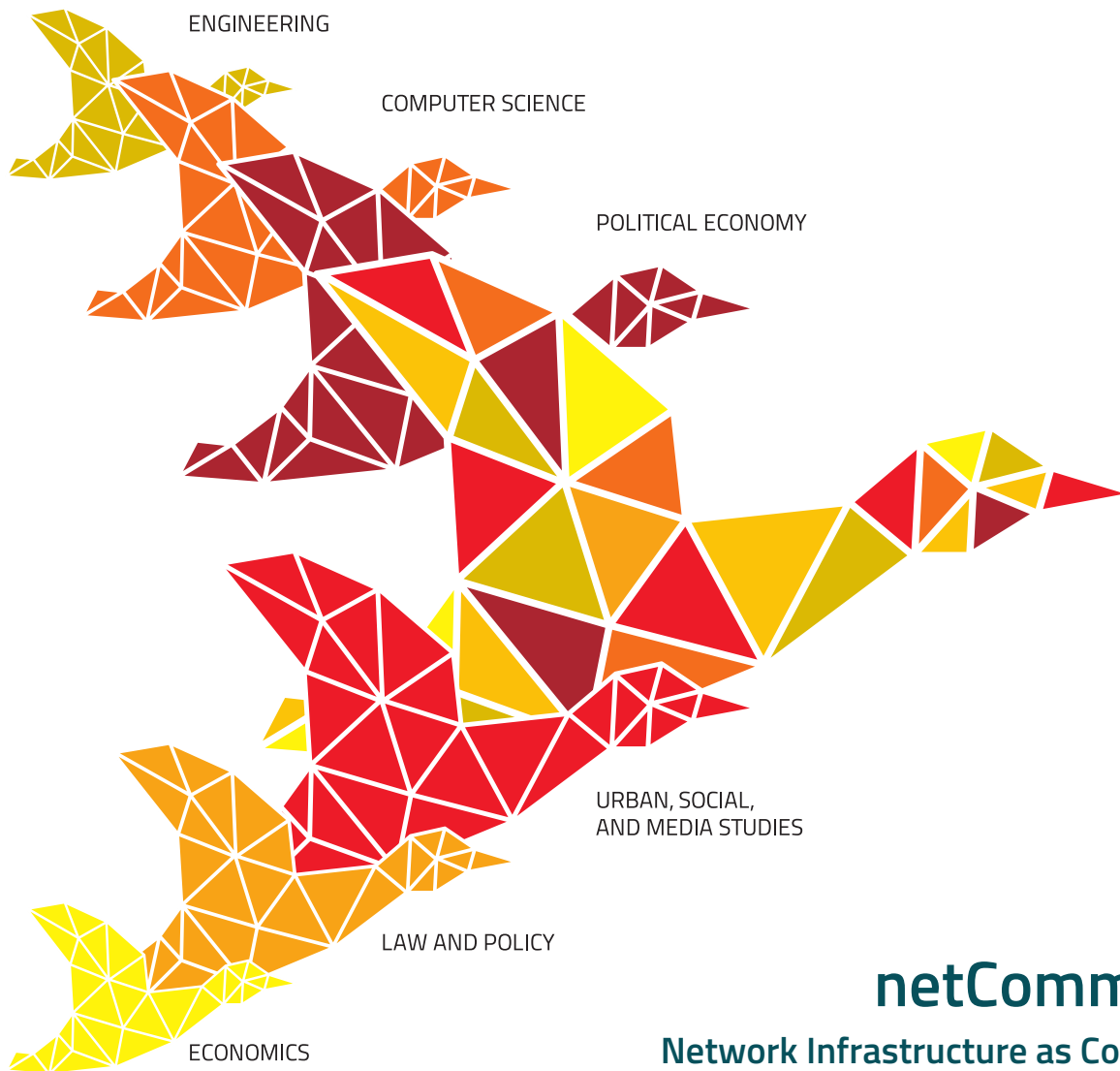
Leonardo was a key element in this process, both for the input coming from netCommons and for his help in writing the project, which we believe will be, if financed, an important starting point for our collaboration.

Salvatore Moretti, ninux Florence



Marzia Frediani, ARCI Florence





netCommons
Network Infrastructure as Commons

Monitoring CNs: Report on Experimentations on CNs.

Deliverable Number D2.7
Version 1.0
December 28, 2017



This work is licensed under a Creative Commons "Attribution-ShareAlike 3.0 Unported" license.

