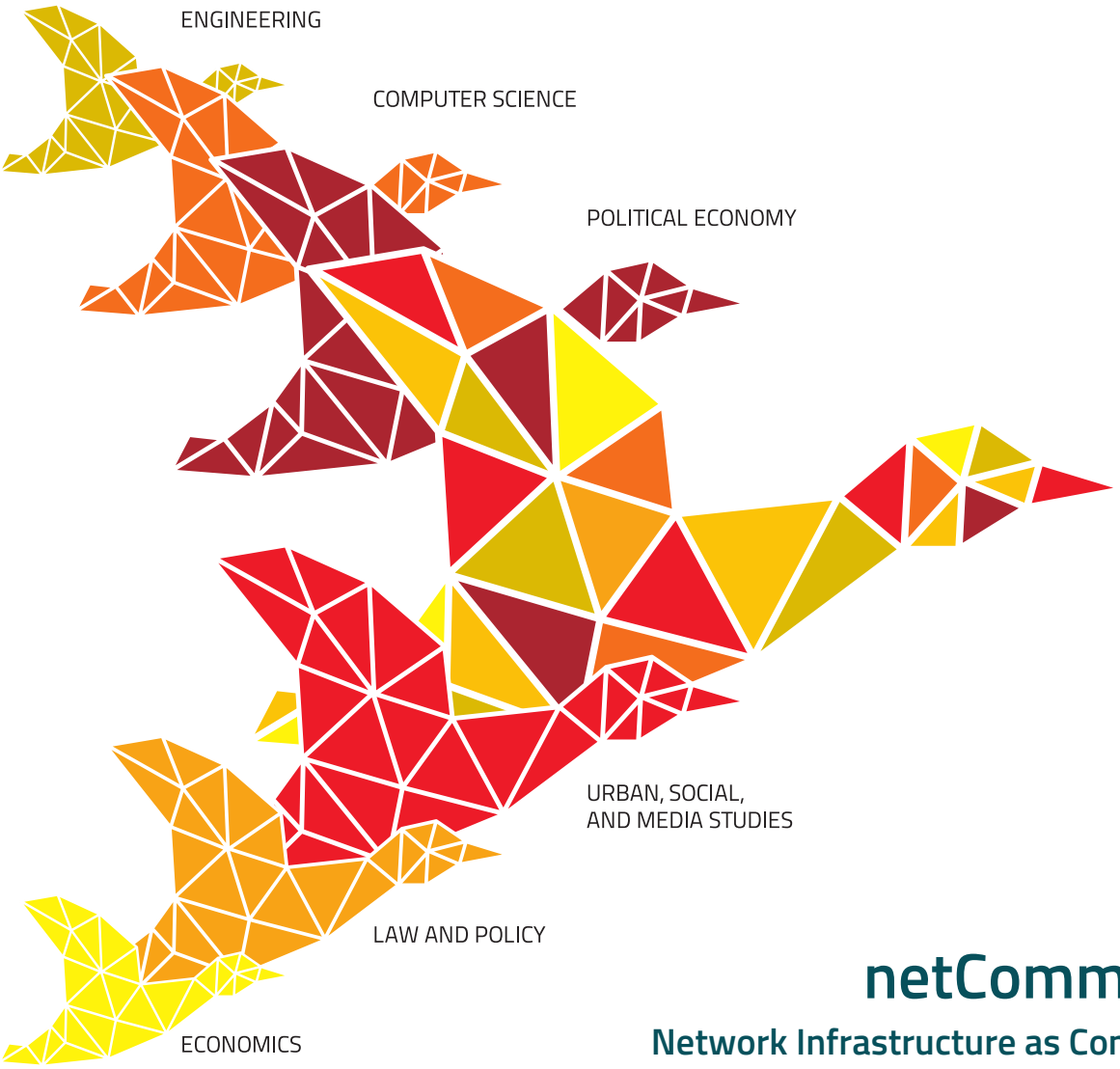


netCommons

Network Infrastructure as Commons

Monitoring CNs: Report on Experimentations on CNs.

Deliverable Number D2.7
Version 0.4
August 9, 2017

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

netCommons.eu

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	August 9, 2017
Number of pages:	25
Keywords:	community networks, monitoring tools, metrics, robustness
Authors:	Leonardo Maccari, UniTN
Peer review:	Renato Lo Cigno, UniTN

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

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 a set of metrics, and the related source code needed for their computation, were described, which allowed to "feel the pulse" of a CN and understanding the level of decentralization and the possible presence of single points of failure. Such metrics are applicable to the network graph of a CN, but also to the social network graph, as derived by the analysis of the communication instruments used by the community (e.g. mailing lists). With these instruments it is possible to perform a multi-layer analysis able to identify the nodes that are critical for the network to be operative, and the people that are critical for the community to thrive. Since people own nodes, the two factors are strongly correlated and must be analysed together.

This deliverable further develops this theme in two directions. The first is the development of new metrics, new source code and the realization of new scientific publications to strengthen the methodology (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 the external conditions. In fact, models that describe spatial networks (networks in which the nodes have a spatial position, which does not normally happen with other networks, such as social networks) tend to generate networks with a strong spatial hierarchy, in which every node "influences" an geographical area that is disjoint by the area of other nodes. One of the analysed CN confirms this, while another shows a different pattern, which is an important result, since it suggests that CNs may have some peculiar features that distinguish them from the other similar networks.

The second direction deals with the integration of the mentioned metrics with the existent software monitoring tools of CNs. In this regard, the work focused on two currently used platforms, NodeShot and OpenWISP2. These two pieces of software have been created in the context of the ninux.org network (the first one) and by a community of people including some developers from the ninux (the second one). While NodeShot is used only in the ninux community OpenWISP2 is of larger interest, thus, we decided to focus on developments on this platform, which we detail in Chapter 3.

Contents

List of Figures

List of Tables

List of Acronyms

CN	Community Network
NMS	Network Management System

1. Introduction

This intermediate release builds upon D2.5 [1], as well as the paper [2]. The goal is to represent and describe the advancement of the work after six additional months, which can be classified under four different "bullets":

- A new metric to understand the development of a CN under the assumption it is a spatial network;
- An analysis of ninux under these assumption has been made;
- Design and integration of metrics and visualization tools on the OpenWISP and NodeShot projects;
- Initial collection of feedbacks from the communities.

In particular the analysis on ninux seems to indicate that the network is evolving diverging from any generic model already known and in general also from trends observed in other CNs and distributed meshes.

2. Spatial Networks and Network Hierarchy

2.1. Ninux as a Spatial Network

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-line distribution networks [3], and their growth can be modelled using a Cost-Benefit Analysis (CBA) framework [4]. 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 model [5]). The second effect is given by a real life constraint: the cost of a physical edge increases with its length, so 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 a in a CN both effects exist.

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, the 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.

We can now introduce a precise definition of a metric that expresses how much a network can be considered hierarchical. We start from a generic metric used in the literature [4] and we customise it to our case. We omit some design details in this section which can be found in Appendix A. First of all, a root node in the network is chosen, and to each node a level is assigned with a function we call $l(v_i)$. In our case we tested several centrality metrics to define the root node, and we finally choose the node with the highest eccentricity. $l(v_i)$ simply computes the distance (in terms of hops) of v_i from the root node. For each v_i a subset $N'(i)$ of its neighbors is defined as:

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

Given $N'(i)$, an "influence zone" is defined, that is an area that contains $N'(i)$ whose specific definition is application-dependent. In our case we use the convex hull of all the nodes in the subset. Given a level l and a node $v_i | l(v_i) = l$ 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:

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

$$\mathcal{I}_{l+1} \subset \mathcal{I}_l \ \forall \ l \tag{2.3}$$

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:

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

And consequently the separation index for level l, which we refer to as intra-level separation is given by the average of the separation of the zones in the same level. We call V_l the subset of V containing all the nodes at level l, then:

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

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:

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

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 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 [4], we want to verify to what extent the ninux community unwillingly built a hierarchical, not redundant and thus, fragile network.

2.1.1. The separation of the Ninux network

Tab. 2.2 reports the intra-level separation at each level in the ninux network, together with the number of zones per level. Tab. 2.3 reports the values of the inter-level separation, the corresponding percent of the total area covered by level l, and the percentage of nodes included in each level. Tab. 2.2 shows that for each 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.

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. 2.1 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 a nodes at the same level are often disjoint, if a link that goes from one

Attribute	*Value*
# 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 2.1: Main attributes of the geo-located ninux communication network, Jan. 2014.

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 mean 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 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.

level	*zones*	s_l
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 2.2: Average intra-level separation in ninux.

2.1.2. 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 FFWien[1]. 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 or 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 4 layers, with 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

[1] See http://funkfeuer.at

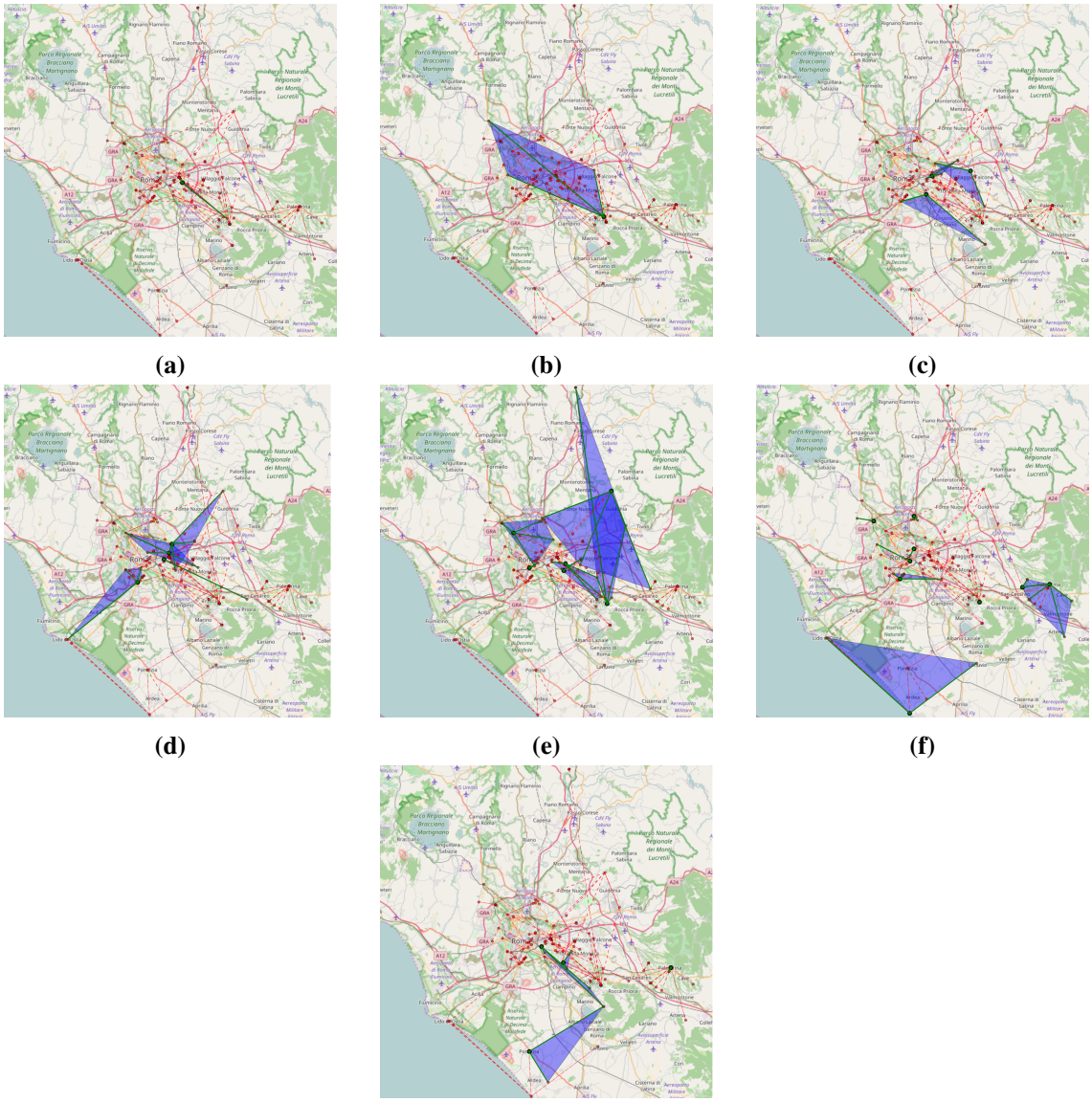


Figure 2.1: The Interest zone for the levels of the ninux network. Note that level 0 is made of a single zone that is very narrow and thus, barely visible in the picture.

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. 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.

level	$\hat{s}_l$	*Area (%)*	*Nodes (%)*
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 2.3: 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 is fully overlapping with the previous ones.

level	*zones*	s_l
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.

level	$\hat{s}_l$	*Area (%)*	*Nodes (%)*
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.

3. Monitoring Ninux

This section describes the various software platforms that are at the base of the development that T2.4 is currently undergoing, and will keep on doing in the second half of the second year of the project. Two platforms are described, the first one is the NodeShot architecture used by the ninux CN to visualize and manage the network. We will briefly describe the platform and then pass to the description of a set of other tools that are based on the NetJSON standard and the OpenWISP platform. While the former is used just by ninux and is not actively supported, the latter is the approach that ninux has taken for its future developments and involved 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.
- It is 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 already assigned IPs 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.

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-karte for Freifunk, NodeWatcher for Wlan-Slovenija and counting. 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.

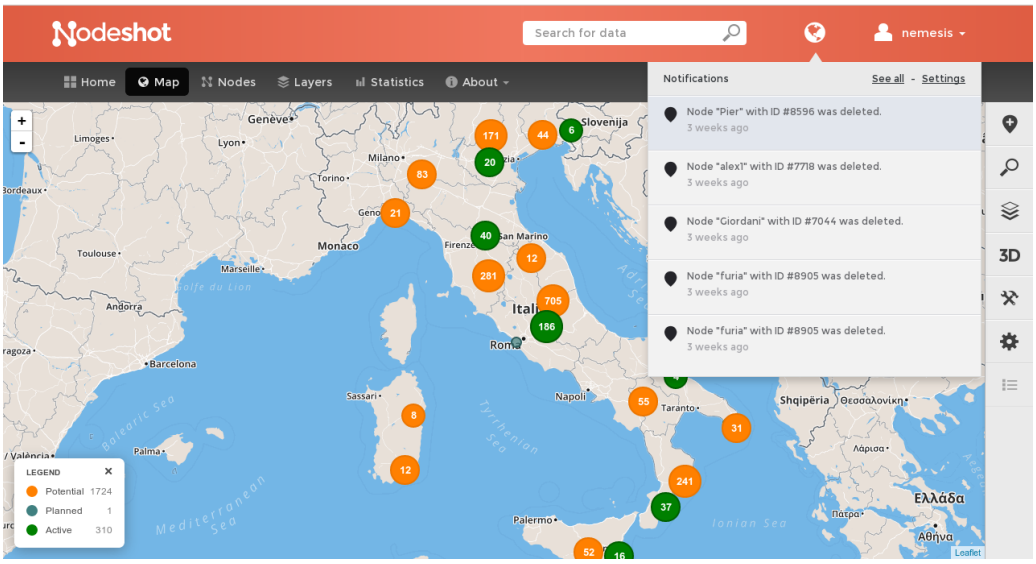


Figure 3.1: The ninux network as presented in Nodeshot

3.2. The NetJSON and OpenWISP Ecosystem

3.2.1. The NetJSON Specifications

As reported in the official netJSON website[1]:

> *"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 RFC[2]. 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 such effort and while its standardisation is lagging its use is instead growing among the various communities and developers[3].

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 www.netjson.org
[2] See draft-capoano-kaplan-netjson-00, http://netjson.org/rfc.html
[3] 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.sj[4] 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

[4]See 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.sj[5]

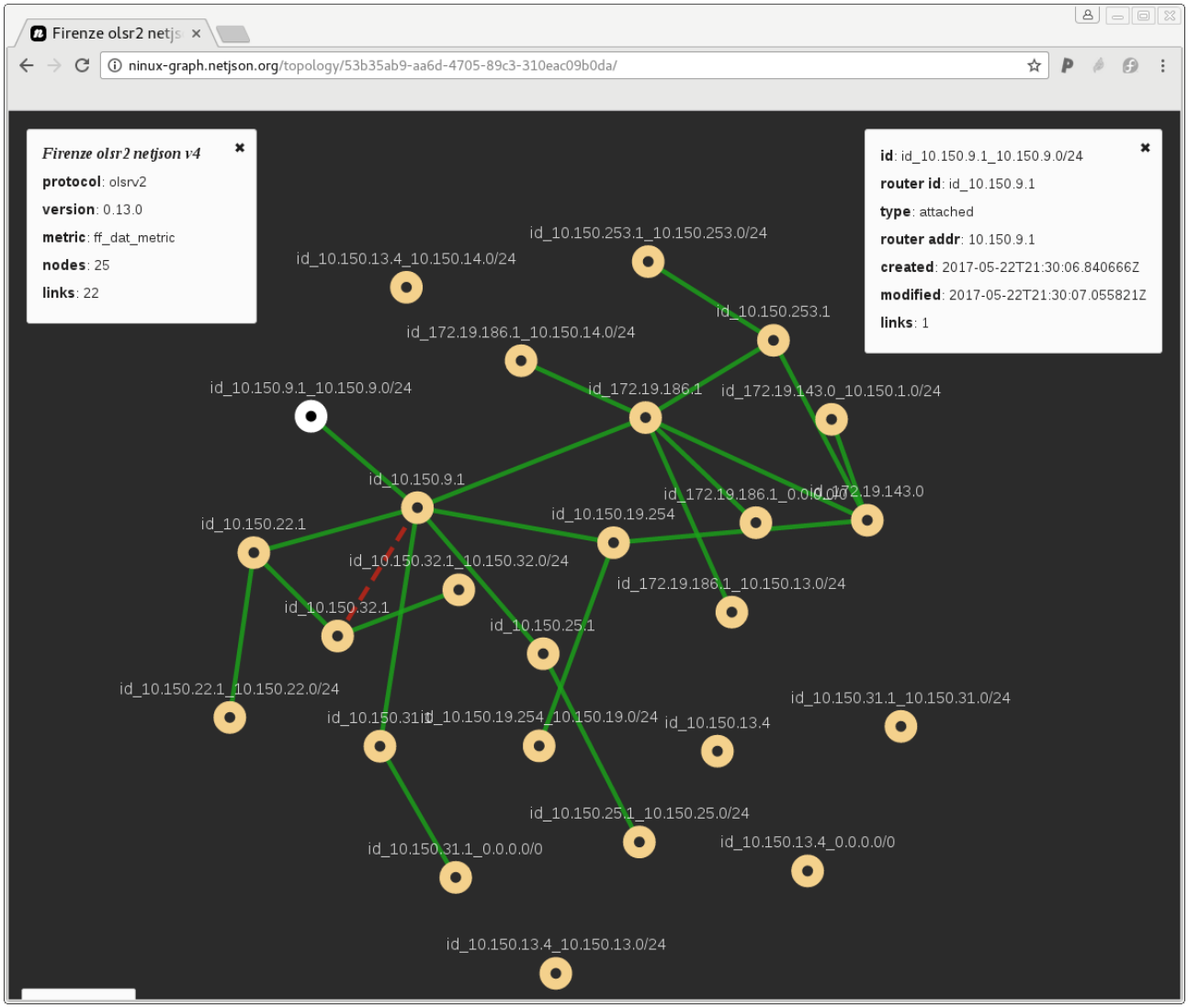


Figure 3.3: The netjson visualization of the ninux network in Florence.

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.

[5]The 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 Capoano) 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 developments[6] 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 this project is the django-netjsongraph library[7], 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"*[8]. Django is used on tens of thousands of websites, it has a very large developer community and 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.

Resuming 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 netdiff[9]
- 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, while NetJSON is a standard language to describe networks, and netjsongraph is a javascript library to visualize the topologies, django-netjsongraph adds the needed instruments to acquire the information from the NetJSON, elaborate them and save them in a database, and use 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 an URL pointing at a NetJSON (or even other kinds of topology description) file.

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 abroad[10].

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)[11]. Since

[6] See http://openwisp.org/gsoc/ideas-2017.html.
[7] See https://github.com/netjson/django-netjsongraph.
[8] See https://www.djangoproject.com/.
[9] netdiff is a library that converts topologies in several known formats into NetJSON.
[10] http://openwisp.org/history.html
[11] See the project mailing list https://groups.google.com/forum/#!forum/openwisp and Github repository https://groups.google.com/forum/#!forum/openwisp

OpenWISP2 is primarily intended for ISPs, from now on we will refer to an hypothetical "network owner" to describe OpenWISP2, we will then explain how this applies to Community Networks (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 a network manager he should reconfigure singularly each wireless router, while the network manager 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 for 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 Ansible, 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 informations, similarly to what Nodeshot does (see Fig. 3.1).

Regardless of its current early state of maturity the project is growing, there are 128 members in the official mailing list that is pretty active[12], the code updates are frequent, there were about 400 installation of the related Ansible role[13] and one public administration that is already using it to manage a running network[14]. 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.

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 every user must have the permission to manage only his own nodes.

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

[12] See https://groups.google.com/forum/#!aboutgroup/openwisp
[13] https://galaxy.ansible.com/openwisp/openwisp2/
[14] https://twitter.com/openWISP/status/826763786728140801

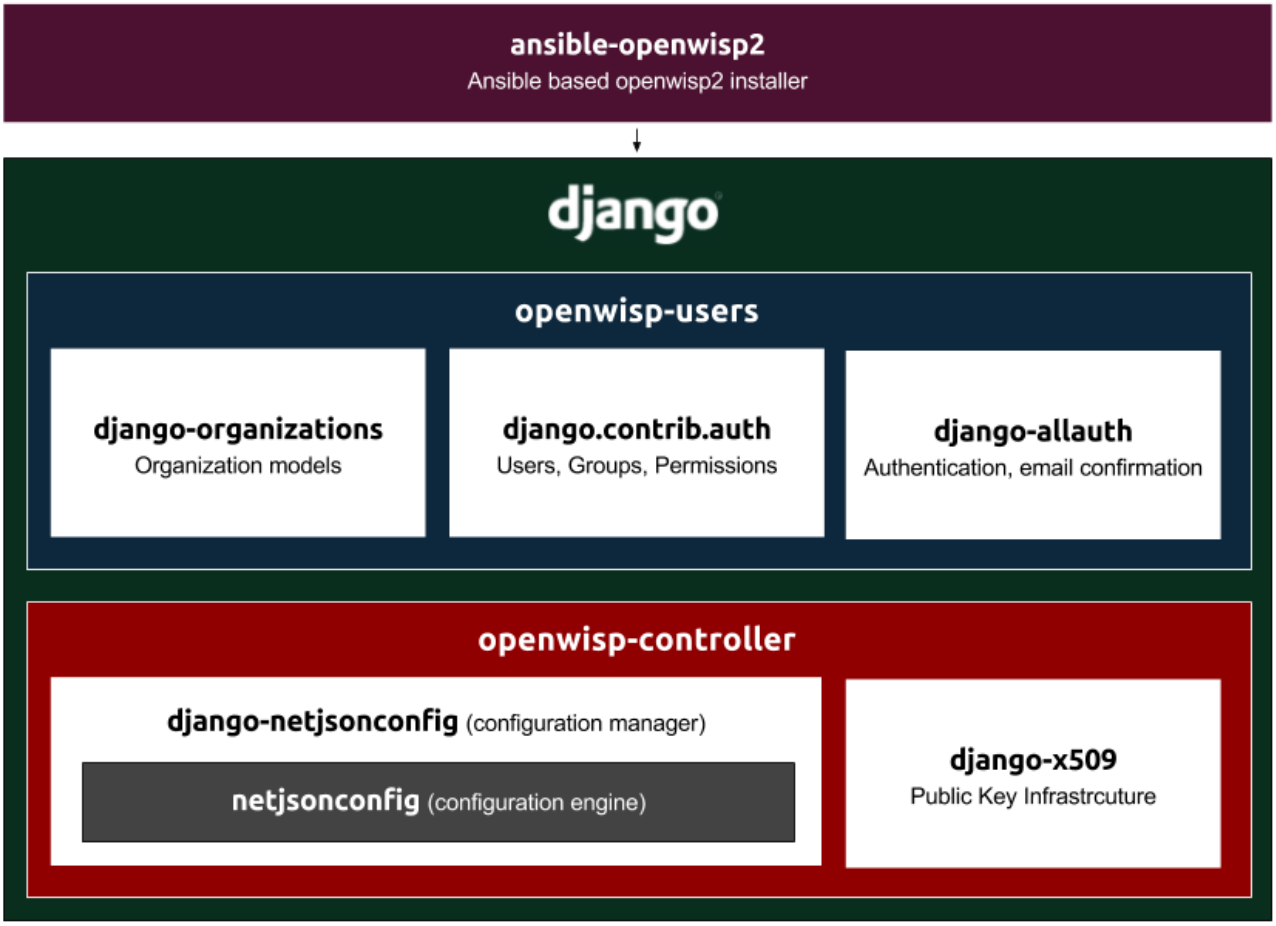


Figure 3.4: The components of the OpenWISP2 platform, taken from the github project page.

3.3. Planned Developments

At this stage of the T2.4 it is certainly more rewarding, for the outcome of the project to integrate the functions developed in T2.4 in the OpenWISP2 code base rather than pushing them in the NodeShot 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 CN 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.

The results that we presented in the first deliverable of T2.4 [1], extended with the contents of Chapter 2 deal with the computation of centrality and robustness metrics on the various layers of a community network, the network layer, the physical layer, and the social layer. The current development plans are the integration of these metrics in OpenWISP2 and/or in the related modules. Specifically, given the information that OpenWISP2 already manages (or plans to manage in the next releases) we will introduce the necessary code to realize:

D1 a network visualization that highlights the presence of critical nodes, under the various definitions we have used for this term. We plan to highlight nodes whose failure would provoke network partitions and nodes that have an high centrality and could therefore become bottlenecks.

D2 A network visualization that highlights the spatial hierarchy of the network, in order to visually represent

the presence of zones that are more (or less) redundant, and eventually the need to add density in some critical zones.

Since OpenWISP2 will actually replace Nodeshot, it will also include information on the node ownership, thus, we plan to also add:

D3 a visualization of the distribution of the node ownership and of the owner robustness as defined in [1].

Pending on the possibility of integration of the data coming from communication tools we will introduce also:

D4 the correlation of the ownership metrics with the centrality in the social network.

At the current state of things, D4 is the point that will be more delicate to realize. The analysis of the social network is very network specific, and thus, it is impossible to realize general instruments that can be used to correlate owners with user IDs in various media. For example, the same ninux community in the last two years switched from a massive use of the mailing list, as documented in the previous deliverable, to the Telegram instant messaging system, which is today attracting many users. At the same time, regular meetings take place on IRC chats, and other instruments have been proposed (like the Matrix decentralized communication system or the Slack chat). Thus, keeping up to date with the frequent change of platforms of the communities is hard in the time-frame of the project, and it is hard to imagine the realization of generic monitoring systems. While the development D1-D3 will be realized in fall 2017, the requirements for D4 will be analysed in the next months and eventually implemented before end of 2017, and monitored in the beginning of 2018.

4. Conclusions

This intermediate deliverable reports on the advancement of the developments planned for T2.4, which will include the realization of open source instruments in the OpenWISP2 platform, or in the related sub-modules. The deliverable describes the platforms, the decisions taken on their choice, and the details the way we are going to include the libraries that we realized in the first half of T2.4 (all Python-based) in the available open source projects.

The integration phase will start in M19, and will continue up to M22, then, two months will be left to monitor the usage of the instruments and to ask for feedback from the users to understand their usefulness.

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. Maccari, "On the Technical and Social Structure of Community Networks," in *The First IFIP Internet of People Workshop, IoP*, Vienna, Austria, May 20 2016.

[3] M. Barthélemy, "Spatial networks," *Physics Reports*, vol. 499, no. 1, pp. 1–101, 2011.

[4] 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

[5] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.

A. Spatial Separation

The spatial separation metrics were partly modified from their original definition in [4] 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.

level	s_l *(eccentricity)*	s_l *(closeness)*	s_l *(betweenness)*
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:

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

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*"[4]. 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.

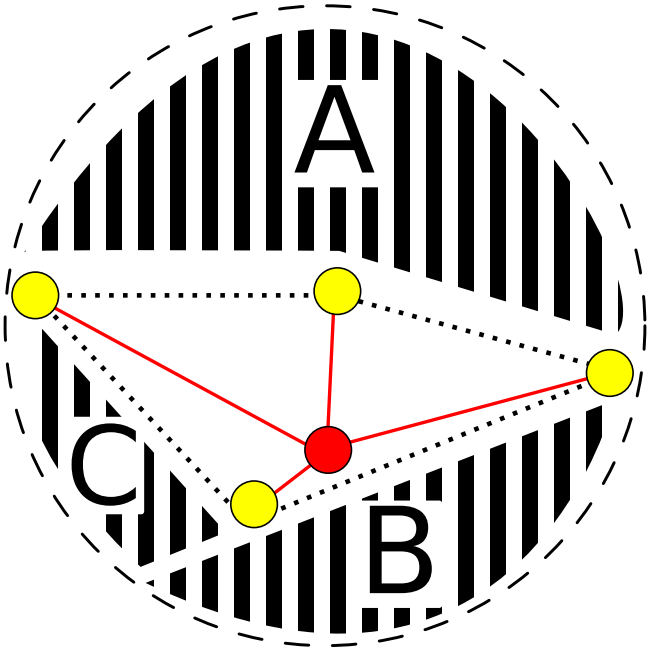
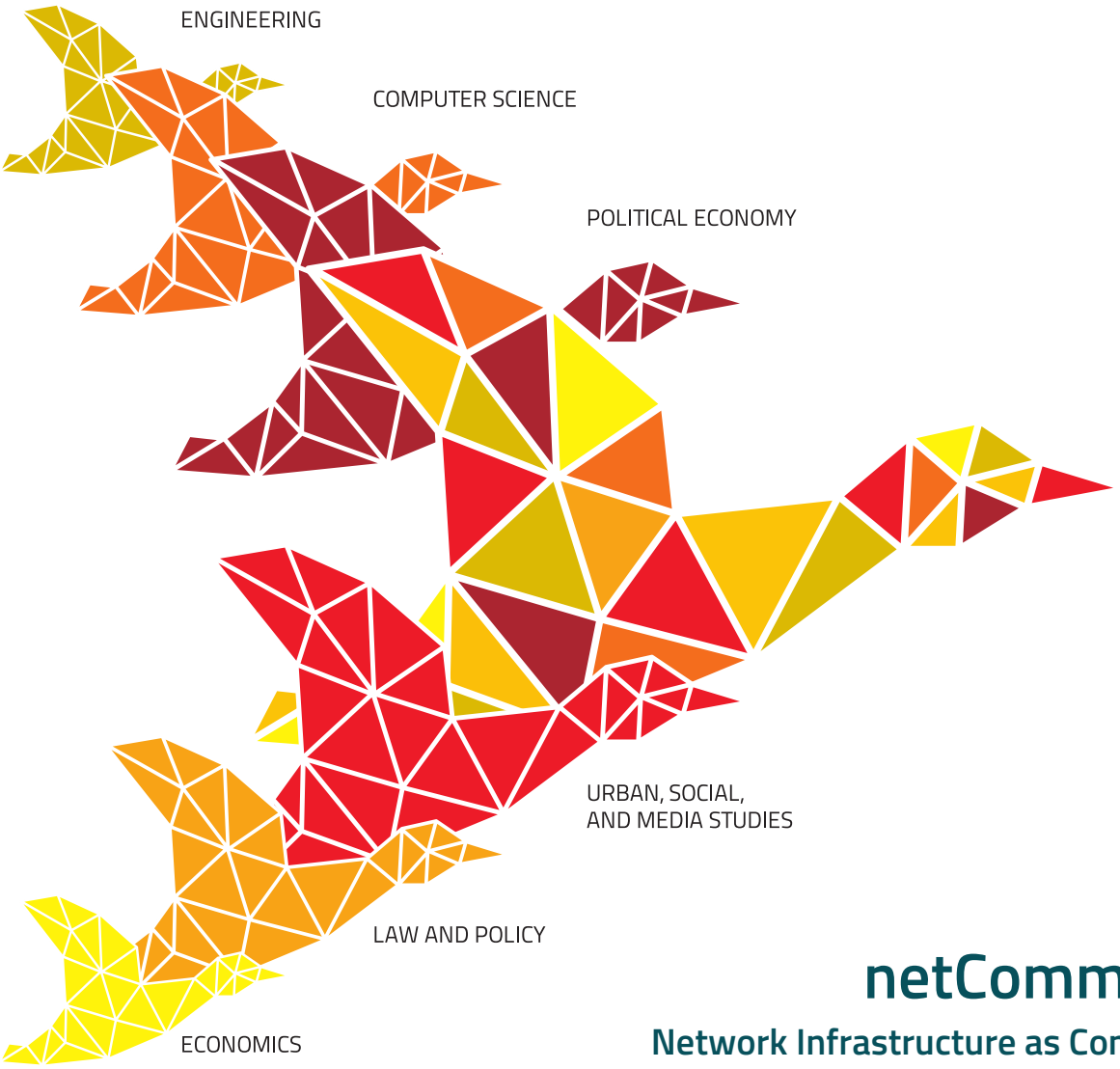


Figure A.1: An example definition of influence zone.



netCommons

Network Infrastructure as Commons

Monitoring CNs: Report on Experimentations on CNs.

Deliverable Number D2.7
Version 0.4
August 9, 2017

netCommons.eu