System enhancements (Year 2)

Deliverable D2.3

Date: 18th September 2013

Version: 1.0

Editor:	Lorena Merino, Ivan Vilata
Deliverable nature:	Report (R)
Dissemination level:	Public (PU)
Contractual Delivery Date:	15/11/2013
Actual Delivery Date	20/09/2013
Suggested Readers:	Project partners
Number of pages:	27
Keywords:	Software, node system, tools, services

Authors:	Marc Aymerich, Axel Neumann, Ivan Vilata (Pangea). Bart Braem, Glenn Daneels (iMinds), Leandro Navarro, Manos Dimos, Navaneeth Rameshan (UPC) Henning Rogge, Anne Diefenbach (Fraunhofer FKIE) Ralph Schlatterbeck, Aaron Kaplan (Funkfeuer) Pau Escrich (Guifi)
Peer review:	Leandro Navarro, Roc Messeguer (UPC)

Abstract

This document presents an update of the architecture, design and development of the CONFINE testbed software system done during the second year of the project. It builds on the work reported in D2.1 during the first year of the project. The software can be found in the project repository at http://redmine.confine-project.eu.

Table of Contents

1. Introduction....5
1.1. Contents of the deliverable....5
1.2. Relationship to other project deliverables....5
2. Core Testbed Components....7
2.1. REST API....7
2.2. Testbed controller....7
2.3. Node software....9
2.4. Virtual CONFINE Testbed (VCT) and VCT container....9
2.5. Testing the system....9
3. Common NodeDB (user Interface/admin Interface)....10
3.1. Overview....10
3.2. Node Database....12
3.3. Common API....12
3.4. Dashboard....12
3.5. Authorization....13
3.6. Network Monitoring and Statistics....13
3.7. Conversion and Import....14
3.8. Lessons learned from importers....15
3.9. Spider....15
3.10. Improvements to the relational database back-end....16
4. Monitoring And Self-management....18
4.1. Motivation....18
4.2. Design....19
5. Software-defined Networking....21
6. COntrol And Management Framework (OMF)....22
7. DLEP....23
7.1. DLEP development....23
7.2. CONFINE DLEP implementation....23
8. Conclusions....25
9. References....26

Figures

Figure 1: Models And Relationships In Django....9
Figure 2: An Overview Of The Overall Architecture Of The NodeDB....11
Figure 3: Object Model Of Common Node Database....13
Figure 4: Funkfeuer Topology....16
Figure 5: The List Of Node Status....21
Figure 6: Historical Data For A Specific Parameter For A Node....21

Figure 7: A Treemap View Of The Historical Data For All Nodes...22

Tables

1. Introduction

1.1. Contents of the deliverable

This document presents an update of the architecture, design and development of the CONFINE testbed software system done during the second year of the project. It builds on the work reported in D2.1 during the first year of the project.

The description of work document for CONFINE mentions this deliverable as instrumental to reflect progress in the following items:

> In Indicators or advance over the state of the art:
>
> *"That work will be performed and revised over the 4 yearly iterations of the testbed in task T2.4. Revisions to the software will be released on D2.3 (M24), D2.5 (M36) and D2.7 (M48)."*
>
> *"Software components that implement proposed algorithms and integrate them into the management frameworks are defined in T2.3 and delivered in D2.3 (M24), D2.5 (M36) and D2.7 (M48)."*
>
> *"Software tools for implementing self-management actions for the automation of the testbed, dealing with the interrelated problems of distributed (global) allocation of channels, IP addresses, IP ranges, routes, routing domains. These tools will be integrated in the enhancements to the management tools and services and the embedded system software developed in T2.4 and delivered in D2.3 (M24), D2.5 (M36) and D2.7 (M48)."*

As part of T2.2, this deliverable is described as:

> *"The enhancements of tools and services, and the update of embedded node system will be reported in D2.3 (M24) (software and documentation), D2.5 (M36) (software and document), D2.7 (M48), describing in detail the problem addressed and the developed solution. D2.7 (M48) will be the final software system and a complete report of the final system."*

This document comprises the following topics:

- A description of the software developments in this period
- Links to the online documentation[1] and the software repository[2].

1.2. Relationship to other project deliverables

D2.1 Initial system software and services of the testbed – M12: A report that describes the software to construct the testbed developed during the first year. D2.3 updates D2.1 and describes the new developments in year 2.

D2.2 Initial software system for the testbed (nodes, services) – M12: A snapshot of the developed software (D2.1) at month 12. The latest version of the software can at any time be downloaded from the publicly accessible CONFINE repositories (http://redmine.confine-project.eu). D2.2 contains the implementation of the software system described in D2.1. D2.3

[1] Project wiki: http://wiki.confine-project.eu
[2] Project software repository: http://redmine.confine-project.eu

includes a reference to the updates in the CONFINE software during the second year of the project.

D2.6 Implementation of federation mechanisms for community networks – M24: Describe the federation mechanisms explored in the CONFINE project which are used to interconnect the diverse community networks involved in the project. D3.2 uses these concepts.

D3.1 Operation and support guides of the testbed – M12: The CONFINE project deploys its testbed for community networks called **Community-Lab**[3], and a virtual local testbed (VCT), which are based on the CONFINE testbed software system developed in WP2 and presented in D2.1 and D2.3. D3.1 therefore describes the testbed that instances the CONFINE testbed software system described in D2.1. D3.2 builds and refines D3.1.

D3.2 Initial management guide of the testbed – M24: Describes how the software developed in the two years of the project is used to provide an operational testbed for researchers. This is based on the software reported here.

D4.1 Experimental research on testbed for community networks (year 1) – M12: D4.1 reports on experimentally driven research that was carried out to support the development of the CONFINE testbed software system. The interaction between WP2 and WP4 was bidirectional. The development of the testbed required looking at options beyond the limits of the current state-of-the-art. Work of WP4 also comprised a review of research topics relevant for community networks in order to identify use cases to be taken into account in the architecture and design of the CONFINE testbed software system. D4.1 therefore contains research work that contributed to the development of the CONFINE testbed software system.

D4.8 Tools for experimental research (year 2) – M24. Similarly to D4.1 it reports on the research activities in the second year of the project. Several activities have used the CONFINE testbed.

D5.1 Dissemination, training, standardization activities in year 1 – M12: This deliverable reports on the interactions the CONFINE project had with different stakeholders in many kinds of events. While on one hand the CONFINE project was communicated to third parties, CONFINE also received valuable external feedback, which also contributed to the development of the CONFINE testbed software system reported in D2.1 and here.

D5.4 Dissemination, training, standardization activities in year 2 – M24: Similarly to D5.1 it reports on the dissemination, training and standardization activities in the second year of the project.

[3] http://community-lab.net/

2. Core testbed components

2.1. REST API

Besides reflecting architectural updates resulting from the stabilization and testing of CONFINE code (including the complete definition of users, groups, roles and permissions), the CONFINE REST API[4] has become more structured and modularized to better encapsulate separate subsystems (community network, management network, tinc VPN backend). It is also more standards-compliant (JSON pointer, JSON patch, HTTP headers) and RESTful (e.g. use URIs instead of numeric resource identifiers) with enhanced browseability.

2.2. Testbed controller

The CONFINE controller[5] is a software package used for managing CONFINE testbeds. It provides users with a web interface as well as a REST API, allowing them to create and manage slices of the testbed.

Additional extra functionality has been added besides the core components for managing nodes and slices. A ticket system has been developed as part of this software package allowing researchers, technicians and testbed operators to report and track all kinds of testbed related issues. Feedback from the testbed components is now provided by the controller through periodic monitoring of the node state as well as monitoring the management network connectivity of different components (nodes, slivers, hosts). Finally mechanisms for centralized node management have been put in place in order to ease the maintenance of large scale testbeds by enabling the execution of operations in multiple nodes at once.

2.2.1. REST API Resource List Filtering

The CONFINE controller REST API[6] allows listing resources; for example, calling *https://panel.community-lab.net/api/nodes/* will give a list of all nodes registered to the controller. Since depending on the size of the testbed, this list can get very long, the API defines, among other possibilities, a format for filtering this list server-side[7]. The requirement is that a key-value pair consisting of a JSON pointer[8] and a matching value be passed in the URL to serve as filters: e.g. *https://panel.community-lab.net/api/nodes/?/slivers/slice/id=60*, with */slivers/slice/id* being the pointer relative to the objects in the list (i.e. the nodes) and *60* the value, would return all nodes who have slivers belonging to the slice with ID 60. The JSON pointer may contain one or more wildcards consisting of an underscore (_), each representing a reference token in the pointer. *https://panel.community-lab.net/api/nodes/?/slivers/_/id=60* would be an example of a query containing a wildcard, and it would return all nodes that have slivers which belong to, refer to or contain any element – such as the sliver's slice, the node, the template – with ID 60.

The CONFINE controller is implemented using the web framework Django[9]. The Django REST filter package[10] does not cover this use case, so it had to be implemented specifically for the CONFINE

[4] http://wiki.confine-project.eu/arch:rest-api
[5] http://wiki.confine-project.eu/soft:server
[6] https://wiki.confine-project.eu/arch:rest-api. Last accessed 1st August 2013
[7] https://wiki.confine-project.eu/arch:rest-api#filtering. Last accessed 1st August 2013
[8] http://tools.ietf.org/html/draft-ietf-appsawg-json-pointer-03. Last accessed 1st August 2013
[9] https://www.djangoproject.com/. Last accessed 2nd August 2013
[10] http://django-rest-framework.org/api-guide/filtering.html. Last accessed 2nd August 2013

project. The JSON pointer query has to be translated into a query the Django query filtering mechanism[11] can understand. This is fairly straightforward for a query without wildcards – all that needs to be done there is to change the slashes (/) to double underscores (__); the relationship between the reference tokens is processed by Django. Processing a wildcard is more complicated because it is necessary to find all possible members which fit the wildcard(s). Following the pointer reference tokens and the Django objects or models they represent one by one allows us to first validate the path up to the wildcard and then get the candidates for the wildcard. To achieve this, we use the internal Django meta functionality which allows introspection of Django models. This is complicated by the way Django differentiates between foreign key fields, many-to-many fields, one-to-one fields, and what we will here refer to as simple fields[12], as well as the fact that, intended as they are for internal use, the Django meta functionalities are barely documented[13]. As a result, trial-and-error is the only way to find out which function accesses which sort of field and how to get the related model, for which there are at least two methods depending on the relationship. Moreover, foreign key relationships, such as are used to model the way a slice consists of slivers and a sliver belongs to a slice, must be declared on only one model of the two models involved – in our case on the sliver. The fact that a slice contains slivers is hidden in the metadata. Since we need to discover the existence of such an element, the usual way to access such backwards relationships[14] is useless to us. And once a way to access these related elements is found, we need to make sure we do not create an infinite loop between a foreign key and a reverse foreign key in case the query contains two back-to-back wildcards. Figure 1 shows how to access simple and foreign key fields. *name* is a simple field whose model is the one containing it. *template* is a foreign key field with model *Template*. *slivers* is a *RelatedObject* representing the backwards foreign key slice of model *Slice* contained in *Sliver*. Many-to-many related objects have to be accessed separately; they are retrieved with *model._meta.many_to_many* and *model._meta.get_all_related_many_to_many_objects()* respectively.

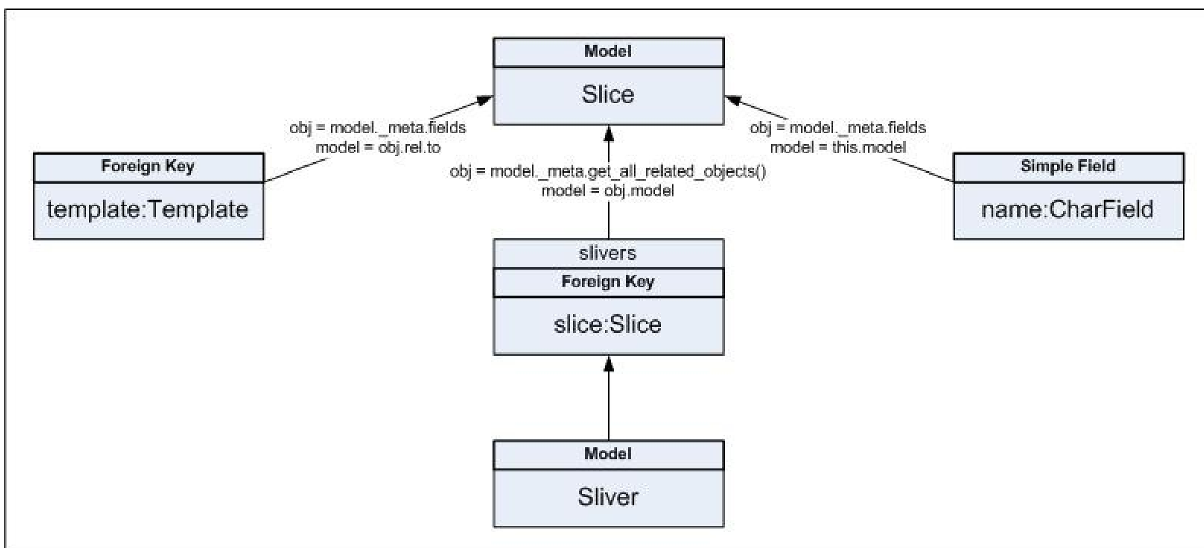

Figure 1: Models and relationships in Django

Once we have found all wildcard candidates including the related objects, they need to be validated against the rest of the pointer and the possibility of converting the query value to a matching type. All paths which pass these checks are transformed into queries for the Django filtering mechanism. Since we want to display the results for all these queries together, we need to

[11] https://docs.djangoproject.com/en/dev/topics/db/queries/#retrieving-specific-objects-with-filters. Last accessed 2nd August 2013
[12] https://docs.djangoproject.com/en/dev/topics/db/models/#fields. Last accessed 2nd August 2013
[13] https://readthedocs.org/projects/django-model-_meta-reference/. Last accessed 2nd August 2013
[14] https://docs.djangoproject.com/en/dev/topics/db/queries/#backwards-related-objects. Last accessed 2nd August 2013

use *Q* objects which allow more complex lookups such as the logical OR combination of querysets which we need here.

The possibility of combining queries using the ampersand (&) remains. For the Django filtering mechanism, this means that each resulting queryset is filtered again by the following query. This is a logical AND combination of the querysets.

2.3. Node software

Major parts of the CONFINE Node System[15] (initially based on the “A hack” milestone) have been extended and re-implemented to conform with the long-term- and polling-based CONFINE API (“Bare bones”). This implementation is now used productively in the Community-Lab CONFINE testbed. To cope with ongoing and future system developments and fixes, mechanisms have been integrated to monitor currently installed software versions, detect abnormal node behavior, and upgrade nodes at different levels ranging from full system re-installations to life updates preserving currently running experiments[16].

2.4. Virtual CONFINE Testbed (VCT) and VCT container

With the inclusion in the Virtual CONFINE Testbed[17] (VCT) of controller software, VCT has become the recommended entry point for new users, since it offers an environment that closely resembles a real CONFINE testbed. The VCT container[18] provides a packaged VCT for further ease of use. Also, VCT now supports native node images for virtual nodes, which provides a more faithful environment for testing and quality assurance.

2.5. Testing the system

For the time being, the only automated testing is carried by a continuous integration server[19] based on Jenkins[20] which checks that node software can be built. The rest of the testing is still done manually, with node software tested under the Virtual CONFINE Testbed[21] (VCT) environment and then on real hardware, and controller software tested under VCT and then on a sandbox testbed with real nodes.

[15] http://wiki.confine-project.eu/soft:node-system-bare-bones
[16] http://wiki.confine-project.eu/soft:node-upgrade
[17] http://wiki.confine-project.eu/soft:vct
[18] http://wiki.confine-project.eu/soft:vct-container
[19] http://builds.confine-project.eu/jenkins/
[20] http://jenkins-ci.org/
[21] http://wiki.confine-project.eu/soft:vct-container

3. Common NodeDB (user interface/admin interface)

3.1. Overview

For community wireless networks (CWNs), a node database serves as a central repository of network information. This "registry" functionality is separate from the testbed controller, which is described above. The testbed controller manages the CONFINE testbed and the experiments (creation of slivers, slices, etc). In contrast to this, the common NodeDB manages the network information *per se* for the community network. It is a registry, a link planning tool, an IP address assignment tool, etc. It comprises information about nodes deployed at certain locations, devices installed at these locations, information about internet addresses, and — in networks that use explicit link planning — links among devices.

All this information is maintained via a web or REST interface by the community members. Therefore the common NodeDB contains the static as well as the dynamic information about the *community network* as opposed to the experimental testbed network information. It is easy to see that a node database is a central component of any community network. Usually community networks thrive to be decentralized, however there are a few centralized components which cannot easily be distributed: IP address assignment and information on optimal channel assignments. It helps to have tools such as the NodeDB for planning these common, shared resources in a community network.

During the last reporting period work on a Node Database (see 3.2) and the Common API (see 3.3) continued. An application that already uses the prototype of this API, the Dashboard (see 3.4) is intended to display all information a user needs and allow easy maintenance of node information. It integrates both static information (node and device data, IP allocation) and dynamic performance data. It will probably be integrated with our Monitoring and Statistics (see 3.6). For the web-interface as well as the Dashboard, two new Authorization (see 3.5) mechanisms were implemented.

The Statistics Server and the Spider (see 3.9) collect data from a running network. Thus, the data which was generated as part of the D2.3 deliverable (for NodeDB) was immediately relevant for WP4 (experiments). The data is interesting for researchers to get new insights into a running mesh network. In addition the Spider data is used when importing the old Funkfeuer redeemer database into the NodeDB. See also the description in the deliverables for WP4.

Currently we offer node configuration data from Funkfeuer Vienna and Guifi.net in two NodeDB databases for research purposes, and AWMN data is coming. To import this data, Conversion and Import (see 3.7) routines had to be written. Part of the reason why the programming of the NodeDB took way longer than expected is the sheer amount of non-uniformity of the existing data which needs to be converted and imported into the NodeDB. While the NodeDB is rather strictly typed (for example a MAC address is to be written in a specific format *xx:xx:xx...:xx* (or separated by dashes ('-')), a lot of the data in the existing community networks is arbitrary and does not fit into any such strict typing schema. Writing converters for the existing data proved to be man-months of effort (see also 3.8). In other words, in order to integrate the data of the existing community networks into the CONFINE testbed and make this data available for researchers, Funkfeuer had to parse, sanitize and re-parse a lot of the legacy data which exists in the network.

Finally for supporting IP address reservation (and IP address objects in a relational database) and to address some requirements of the new common API we had to improve the relational database

back-end (see 3.10) which now supports new query mechanisms and database back-end-specific data types.

Figure 2: An overview of the overall architecture of the NodeDB

Figure 3: Object Model of Common Node Database

3.2. Node Database

During the last reporting period, the object model (see Figure 3) of the node database [FFM] was updated to reflect the necessary enhancements for IP address reservation and some other features (as described above). It can be seen from the model that IP addresses (both IP version 4 and 6 are supported) can be assigned to network interfaces. On top of this model, address reservation is implemented. There are quite some differences between different community networks when it comes to address reservation: some use RFC 1918 private IP space [PrivIP], others public IP space, others assign subnetworks to nodes. In other words, there is no clear standard, thus the *common* NodeDB has to support multiple variants which increases the complexity of the implementation and the design.

3.3. Common API

Ultimately, we would like to have other community networks interact via a standard common API. An application could be written for Funkfeuer and used with the (for example) Freifunk NodeDB. This standardization step is our goal and we made some progress towards this in year 2.

For accessing the common node database an application programming interface (API) was improved compared to the last report.

The API, which uses a Representational State Transfer[22] API (REST API) has been extended to support IP address reservation and will serve as a basis for defining a common API.

Other API improvements implemented:

- Support for polymorphic attributes[23].
- Support for accessing the cooked values of attributes via the REST API.
- Support for accessing meta information about entities, i.e., date/time and user of creation and last change.
- Add response header with link to documentation of resource.
- Support for accessing the links of an object via the REST API[24].
- Implemented a Python module to ease access to the REST API[25].
- Support for authorization (see 3.5) with REST authentication tokens.

3.4. Dashboard

Implementation of a user-dashboard started. The dashboard serves both as a general web-app for interactions of end-users with the NodeDB as well as a reference implementation, showcasing how the NodeDB can be used for rich single-page web applications.

[22] http://en.wikipedia.org/wiki/Representational_state_transfer
[23] *Support for polymorphic attributes*, Christian Tanzer April 2013: http://confine.funkfeuer.at/2013/04/support-for-polymorphic-attributes/
[24] *RESTful API improvement*, Christian Tanzer May 2013: http://confine.funkfeuer.at/2013/05/restful-api-improvement/
[25] *REST Client*, 2013: https://github.com/Tapyr/tapyr/blob/master/_GTW/_RST/_MOM/Client.py

The Dashboard is built on top of the *backbone.js* library — a single-page application MVC framework written in JavaScript — and *bootstrap* — a basic CSS library popularized by Twitter. It thus demonstrates how the NodeDB can be integrated with state-of-the art web technologies.

Several challenges in querying the NodeDB became apparent while developing the dashboard — this helped to improve the use and test cases for the NodeDB and triggered a change of the SQL interfacing back-end to allow certain queries.

The dashboard is in a functional prototype stage right now and is expected to be finished once the NodeDB is ready to be used and contains all authentication and security features needed.

3.5. Authorization

In addition to password-based authentication, two new authentication mechanisms were implemented.

Most browsers today support a method of secure — if not very user-friendly — generation of client certificates where the secret key stays with the browser. Since no Python implementation for using this mechanism on the server-side (although it exists since Netscape times) was available we wrote a library to support the server-side of using client certificates, the library *pyspkac* [pyspkac].

For use by clients of the Common API (see 3.3) the framework was extended to support REST authentication tokens, aka RAT. To get such a token, a client sends a post-request with username and password to the RAT resource and gets back an authentication token that can be used for a limited time to authenticate subsequent requests.

The dashboard currently uses REST authentication tokens for authentication.

3.6. Network Monitoring and Statistics

Note: since this is also part of WP3, we will only briefly mention it here and describe this later in the deliverables for WP3. Please note that the monitoring described here refers to the networking infrastructure and not to testbed nodes, which is described in section 4.

We started to implement a statistics server, collecting statistics from the Funkfeuer network. The collected statistics (reachability, routing and topology of the mesh network) help researchers to better understand real-life mesh-networks. A graph of the current Funkfeuer topology can be seen in Figure 4. More topology visualization is available in our blog[26] and software for an interactive version is available [topo-github].

The implementation of the statistics server triggered a privacy and anonymization discussion in the Funkfeuer network that prompted us to work on IP address anonymization. One promising approach was the Crypto-PAn [cryptopan] algorithm.

To integrate the Crypto-PAn algorithm in our software we created a Python module: *pycryptopan* [pycryptopan]. The module is published with the Python package repository and available for Python versions 2 and 3. We are especially proud of the fact that after publishing the source code for *pycryptopan* on GitHub the author, Michael Bauer, got immediately patches and feedback from completely independent and unrelated developers, meaning that this module is already used by a much wider community than just CONFINE.

[26] *Hackathon* blog post, 2013: https://confine.funkfeuer.at/2013/07/hackathon/

Figure 4: Funkfeuer topology[27]

The monitoring via smokeping[28] is making nice progress. We are getting access to VMs on different community networks in order to create a looking-glass-like ping monitoring solution. Alarms work nicely in the most recent version of smokeping.

3.7. Conversion and Import

As mentioned above, for populating the node database, importers have been written to import data from other sources into the node database.

To eventually switch over from the current Funkfeuer Vienna Node database "Redeemer", an import for redeemer has been written. It converts user data, node and device information as well as IP address assignments. Since the current Redeemer database doesn't contain enough information for the new database — notably information about wireless interface configuration is missing and it is unclear which interfaces that have an IP address assigned belong to which device — we also rely on OLSR data and data retrieved by a spider (see 3.9) to complement the data in the redeemer database.

For data of Funkfeuer Graz, an importer was started which has not been finished. It uses some of the same libraries for reading SQL dumps and contributed to our lessons learned from importers (see 3.8).

To make the Redeemer data publicly available for research purposes an anonymization option was added to not import the personal information during database import.

In addition to the Vienna Redeemer database, an importer for a subset of the node data of Guifi.net was written. AMWN importing still needs to be done as of today.

Both the anonymized data from Funkfeuer Vienna and a subset of Guifi.net data were made available for research purposes[29].

---

[27] *Visualizing OLSRD topography using d3* blog post, 2013: https://confine.funkfeuer.at/2013/07/visualizing-olsr-topology-using-d3/
[28] http://tunnel.confine.funkfeuer.at/cgi-bin/smokeping.cgi?target=CONFINE_servers
[29] *ff-nodedb.funkfeuer.at and guifi-nodedb.funkfeuer.at online, Aug 2013*: http://confine.funkfeuer.at/2013/08/ff-nodedb-funkfeuer-at-and-guifi-nodedb-funkfeuer-at-online-2/

To facilitate updates, an account migration feature was implemented[30] that can port the database accounts to the new database when new data is imported.

3.8. Lessons learned from importers

For importing from another database, we used the SQL dump of that database. This resulted in a library, part of [rsclib], that can read SQL dumps (from both PostgreSQL and mySQL) and offer the data via a Python API.

When writing importers — for both CNML data from Guifi.net[31] as well as SQL database dumps for Funkfeuer Vienna and Graz — we encountered problems with the data offered. In particular, non-sanitized data (like invalid MAC Addresses) and problems with character encodings.

The problems with character encodings were due to the long usage period of the data in question which had encountered changes of character set (from Latin1- to Unicode-based encodings like UTF-8). Some data was double-encoded. This resulted in a module for the SQL dump reader in [rsclib] to sanitize the encoding problems.

Another problem when running our importers was the time it took to complete the data import. This was traced to large transactions of the underlying SQL database. Some commit statements at appropriate places in the code improved performance drastically. So an advice when writing converters or importers boils down to:

> *Don't create all objects in a single transaction; commit every now and then.*[32]

3.9. Spider

Originally intended for augmenting the data used by the importer (see 3.7), a spider was written that extracts the following information from the web-interfaces of the nodes in the Funkfeuer network in Vienna:

- Version and type of software used
- WLAN configuration (if available): channel, signal, ESSID, BSSID, etc
- Network interfaces and configuration information
- IP Address information

The spider can be used on any network that uses OLSR for routing — currently the spider relies on the OLSR topology data for finding out from which IP addresses to retrieve data.

Funkfeuer currently uses a mix of different hardware and software components. The spider can currently handle the following software on devices:

- Freifunk Firmware
- Backfire Vienna
- OpenWRT Firmware
- OLSR "Textinfo" Plugin output

[30] *Account migration*, Christian Tanzer May 2013: http://confine.funkfeuer.at/2013/05/account-migration/
[31] *Guifi.net CNML Wiki*, retrieved 2013-09-09: http://en.wiki.guifi.net/wiki/CNML
[32] *Converter performance*, Christian Tanzer Aug 2013: http://confine.funkfeuer.at/2013/08/converter-performance/

The type of firmware running on the device is auto-detected. When testing, the software auto-detected many nodes from the Funkfeuer Graz network (which uses a different set of devices from Funkfeuer Vienna).

We currently spider the network once a day and keep the data retrieved for statistics on network parameters. Some of the data will be made available via our Statistics Server, (see 3.6).

3.10. Improvements to the relational database back-end

The NodeDB uses the Tapyr [Tapyr] framework to access relational databases. Tapyr in turn uses SQLAlchemy[33] for that task.

During the development of the common node database, we identified some weaknesses of Tapyr's SQLAlchemy wrapper. We implemented the improvements:

- Use of database-specific data types for database columns.

  Tapyr now supports the use of RDBM-specific data-types. For managing IP addresses with properties like network ask and *contains* relationship we had to extend the Tapyr framework to support IP network operations. One of the back-end databases (PostgreSQL) natively supports IP address objects while other databases don't. The framework can now use the native IP address type if supported by the database and emulate the behavior for the other back-ends.

  The back-ends for both PostgreSQL and SQLite support the same queries, with PostgreSQL doing most of the work for Tapyr, while the SQLite back-end implements IP address comparison with complex SQL expressions over the synthetic columns.

- Transitive queries of attributes of joined tables.

  Tapyr now supports the definition of query attributes that resolve to a SQL join of multiple tables. For the common node database, the most important use for transitive queries is to find all objects of a certain type that belong to a specific node. For instance, to find all antennas belonging to some node a client of the REST API can now use a get request like this:

  https://guifi-nodedb.funkfeuer.at/api/FFM-Antenna?AQ=belongs_to_node,EQ,374

  which returns all Antenna instances linked to Wireless_Interfaces connected to Net_Devices connected to the Node specified. Because *FFM.Antenna* is not directly connected to *FFM.Node*, multiple joins are necessary to find the node in question, resulting in a select statement looking like:

```
SELECT
  ...
FROM mom_id_entity
 JOIN ffm_antenna ON mom_id_entity.pid = ffm_antenna.pid
 JOIN ffm_wireless_interface_uses_antenna
   AS ffm_wireless_interface_uses_antenna_1
   ON ffm_wireless_interface_uses_antenna_1."right" = ffm_antenna.pid
```

[33] http://www.sqlalchemy.org/

```
  JOIN ffm_wireless_interface AS ffm_wireless_interface_1
    ON ffm_wireless_interface_1.pid
         = ffm_wireless_interface_uses_antenna_1."left"
  JOIN ffm_net_interface AS ffm_net_interface_1
    ON ffm_net_interface_1.pid = ffm_wireless_interface_1.pid
  JOIN ffm_net_device AS ffm_net_device_1
    ON ffm_net_device_1.pid = ffm_net_interface_1."left"
WHERE ffm_net_device_1.node = 374
```

To query Antenna instances belonging to a specific Person, the client would send a get-request like (domain elided):

/api/FFM-Antenna?AQ=belongs_to_node.owner,EQ,104

- Polymorphic queries computed by the database itself.

  Tapyr supports polymorphic attributes, i.e., attributes that refer to objects of a polymorphic type. By default, there is no database table for a partial type; a polymorphic attribute referring to such a type is a foreign key to one of the set of tables corresponding to the non-partial derived types. Queries for polymorphic types are now completely resolved by the relational database (previously, the results of several database queries needed to be combined in Python which was quite inefficient if sorting and limiting was involved).

4. Monitoring and self-management

4.1. Motivation

The monitoring system[34] is designed specifically to monitor activity on research devices of the Community-Lab testbed. Monitoring the testbed presents specific challenges in the form of large scale of infrequently-used data. The monitoring system should support active measurements that provide insight into the functioning of nodes without revealing too much information of what is running on it, gather slice and sliver specific information and should be flexible enough to add new metrics without hampering the functionality. Monitoring logs should never lose precision and should support passively measured data such as last-time ssh succeeded, number of ports in use, resource hogs (which experiments are using the most CPU, memory, bandwidth and ports).

A general purpose monitoring system does not meet these special purpose requirements of Community-Lab and are meant for different workloads and properties. Slice-specific and sliver-specific information (LXC monitoring) cannot be obtained directly by any of the existing monitoring systems. Nagios, Zenoss, ntop, Ganglia and Cacti use the RRD tool for storing data. RRD is great for storing time series data and aggregating information, but it is quite inflexible. It becomes necessary to compromise between flexibility and efficiency.

Adding new metrics would require updating the database file. Once an RRD is created, it is possible to change existing values and add new data sources, it is not possible to add or remove metrics and change their properties. If modelling of data is not considered carefully, it can lead to a number of updates as and when new slivers are created in a node. Slice-specific data implies data from different nodes and would result in a dynamic list of RRD which in turn would need additional scripts to fetch, aggregate and display data. For instance in Comon (monitoring system of PlanetLab [PlanetLab]), the data model is carefully chosen, but still old database files are deleted when the format changes. In many cases (depending on configuration) if an update is made to an RRD series but is not followed up by another update soon, the original update will be lost. This makes it less suitable for recording data such as operational metrics. There is no way to back-fill data in an RRD series and depending on the data model, a single RRD receiving data from multiple sources can be affected by this. Given the large scale varying resource consumption and the dynamic nature of Community-Lab, flexibility is a key requirement. Apart from that, sliver-centric information is not easily integrated into node-centric data provided by off-the-shelf monitoring systems. This kind of data gathering is an important motivation for developing a separate monitoring system to meet the specific needs of Community-Lab.

[34] https://github.com/navaneethrameshan/

4.2. Design

At a high level, the monitoring system consists of a monitoring daemon running on each research device, a centralized data gathering and processing infrastructure and a display facility. The daemon running on the research device provides node-centric data including sliver specific information, and it monitors periodically (e.g. every sixty seconds). It accepts HTTP requests and responds with HTTP responses, to allow them to be accessed from web browsers in addition to being used with automated systems. The response is provided in JSON format to allow researchers to query and use monitored data. The daemon stores the monitored information in a file locally until the data gathering service has seen it. This ensures that no monitored information is lost during a network partition and helps researchers diagnose any problem that may have happened during this period.

While the daemons operate on research devices, the data gathering and processing operates on a properly-provisioned machine. Data is collected from the daemons using a pull model and is fetched every 5 minutes. All fetches are performed in parallel to reduce latency. Slice centric information is generated by analysing the node centric logs and is stored in the database. Additionally information is aggregated and summaries are provided at a granularity necessary to make meaningful inference from the data. Precision of the monitored information is never lost and it supports data offloading which is then provided as an open data-set.

Confine Research Device Status

Visualize Node metrics over time

Last Seen (GMT)	Name	Disk Size	Number of CPUs	CPU Usage(%)	Load Avg-1 min	Total Memory	Memory used(%)	Network Data Sent/Sec	Network Data Received/Sec	Uptime	Ping Status ▲	Port Status
None	[fdf5:5351:1dfd:60::2]	None	None	None	None	None	None	None	None	None	rtt min/avg/max/mdev = 27.230/27.230/27.230/0.000 ms	port-80:closed, port-806:closed, port-22:closed,
2013-06-21 11:19:39	[fdf5:5351:1dfd:11::2]	84.0G	2	0.5	0.91	2.0G	2.4	0B	0B	9 days, 12:02:57.080000	rtt min/avg/max/mdev = 14.509/14.509/14.509/0.000 ms	port-80:closed, port-806:closed, port-22:closed,
2013-08-09 17:19:51	[fdf5:5351:1dfd:3d::2]	614.4M	1	100.0	0.31	247.6M	14.8	12.6K	22.4K	21 days, 4:31:44.890000	rtt min/avg/max/mdev = 10.799/10.799/10.799/0.000 ms	port-80:closed, port-806:closed, port-22:closed,
None	[fdf5:5351:1dfd:55::2]	None	None	None	None	None	None	None	None	None	rtt min/avg/max/mdev = 0.965/0.965 /0.965/0.000 ms	port-80:closed, port-806:closed, port-22:closed,
None	[fdf5:5351:1dfd:7f::2]	None	None	None	None	None	None	None	None	None	Fail	port-80:closed, port-806:closed, port-22:closed,

Figure 5: The list of node status

Confine Research Device Status

Node: [fdf5:5351:1dfd:3d::2], Value: Total CPU usage

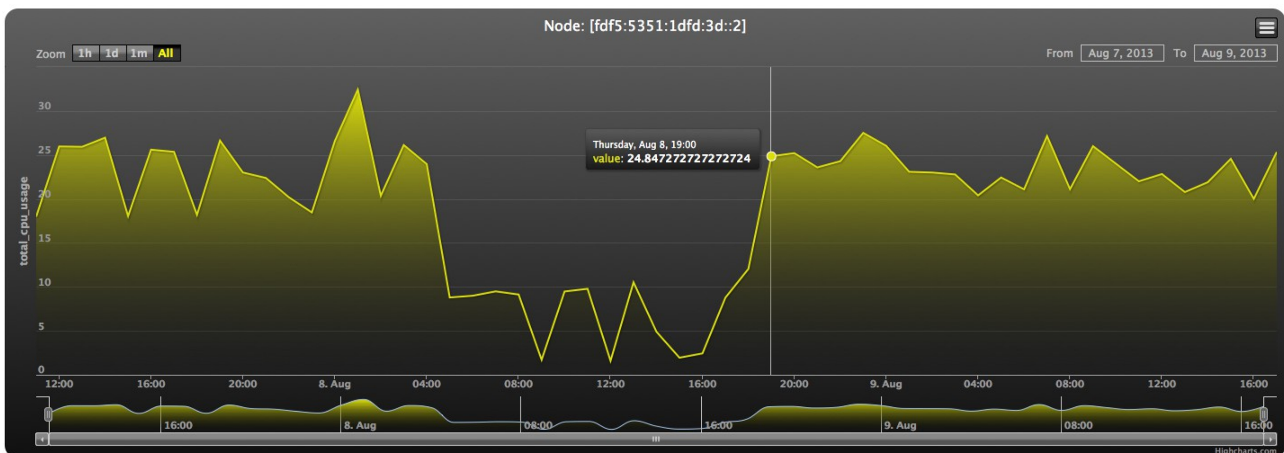

Figure 6: Historical data for a specific parameter for a node

Monitored information is reported via a web interface that supports sorting, and shows graphs of historical data (see figure 6). The reporting currently covers OS-provided metrics and metrics

synthesized from other sources on the node. The system reports the following OS-provided metrics: uptime, CPU utilization, memory utilization, total memory, disk size, disk space available, 1 minute load, network data sent and received. Synthesized data includes last time the monitoring daemon on the research device was seen, open ports, ping status and slice centric information. The system maintains only a manageable set of metrics that help the researchers get insight of any strange behavior in a given node (see figure 5). To facilitate the researchers in selecting nodes to run their experiments the web interface provides a Treemap view of all the nodes (see figure 7) based on the historical trend (customizable) of resource usage.

Confine Research Device Status

Start Time:: None None None -------------- End Time:: None None None

No. of values to calculate average: 100 go

Most Recent

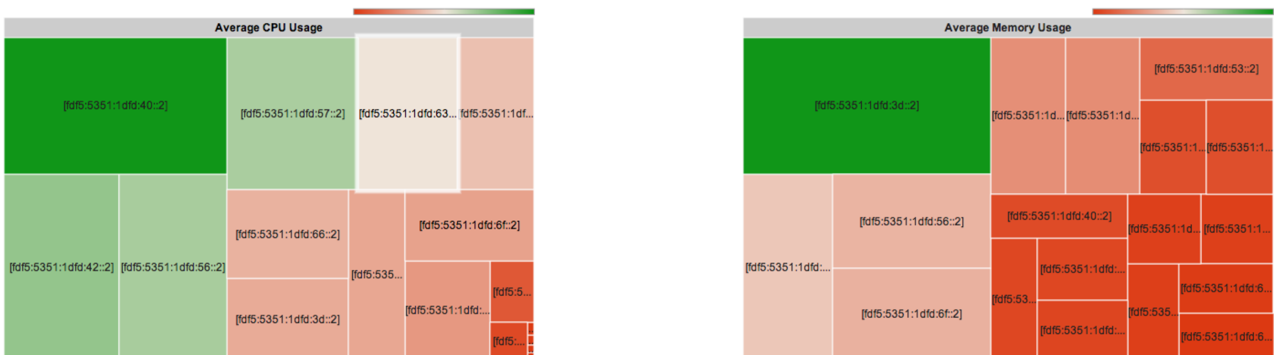

Figure 7: A Treemap view of the historical data for all nodes

5. Software-defined networking

Working with the CONFINE software collection we encountered the need to allow researchers to perform L2 experiments. In order to achieve that goal we would need a way to perform L2 topology virtualization and thus present to the researchers not a low level API but a set of abstract and manageable L2 resources. In spite of the wide variety of ways to achieve L2 topology virtualization, the most prominent approach is the one of Software Defined Networking[35] (SDN).

Network resource virtualization using SDN offers more benefits than simple resource handling. The SDN community is working on a complete networking solution where besides resource virtualization, users will be provided with a configuration and management plane. For these reasons we decided to extend the CONFINE testbed with a platform for L2 experiments[36] using the OpenFlow SDN protocol [OpenFlow].

The implementation consists of two main components:

1. A proxy for the POX OpenFlow controller[37] which is located in the research device and plays the role of a local OpenFlow proxy.
2. A software component that integrates POX with Django[38] in order to provide a UI through the CONFINE testbed server web UI.

The L2 SDN experiment platform is not yet fully integrated with the CONFINE testbed but it will be available in the near future. This effort is also described in the paper “Software Defined Networking for Community Network Testbeds” published in the international workshop on Community Networks and Bottom-up-Broadband, part of the WiMob 2013 conference.

[35]http://en.wikipedia.org/wiki/Software-defined_networking
[36]http://wiki.confine-project.eu/soft:sdn
[37]http://www.noxrepo.org/pox/about-pox/
[38]https://www.djangoproject.com/

6. cOntrol and Management Framework (OMF)

By integrating the cOntrol and Management Framework (OMF) (5.4) [OMF] in CONFINE testbeds[39], experiments on the CONFINE testbed can be described in one single experiment file. From the moment the researchers feed the experiment file to the OMF Experiment Controller (OEC), the OMF integration takes care of everything. From a researcher's point of view, no interaction with the CONFINE testbed is required anymore: this allows researchers to focus on the experiment rather than on how to execute their experiment on the CONFINE testbed.

By using the CONFINE REST API, the OMF Aggregate Manager (OAM) communicates with the CONFINE server to check for available resources and to make necessary resource allocations (i.e. sliver allocation on the available nodes). When the experiment is finished, results can be fetched by using the OML measurement library — by default — provided by OMF.

[39]http://researchinternship.blogspot.com.es/

7. DLEP

The CONFINE Test-Infrastructure is mostly about testing in Community Mesh Networks, which use wireless links to connect the nodes of the network with each other. This means that quite a few experiments will need IP and link layer information about the traffic on the wireless interfaces, including link layer metadata like signal strength or transmission speed.

In addition CONFINE research nodes are built to be integrated into existing networks, which are used by the Community participants for their own communication, both local within the Mesh and with the Internet.

Because of this, it is not feasible to give researchers direct access to Wi-Fi hardware and its configuration. The CONFINE research node must make sure that experiments stay within the defined restrictions and conditions agreed on with the local Community to preserve the privacy of the network participants and prevent experiments from disrupting the Community Mesh itself.

CONFINE uses a prototype of the Dynamic Link Exchange Protocol (DLEP) of the IETF Manet Group to deliver link layer data from the operating system to research nodes while separating them from the real control interfaces.

7.1. DLEP development

DLEP is still in active development in the IETF Manet Group, the protocol draft has yet to stabilize and find consensus about the content of the draft. Most of the CONFINE work on DLEP has been done by participating in the Manet Group to make sure that the protocol will be flexible enough for usage both with modern Software Defined Radios and with the low-cost hardware used by most Community Mesh Networks.

The IETF meeting in Berlin ended with the creation of a small DLEP research team, which is collecting a larger set of link layer metrics from several people including CONFINE members.

There has also been an announcement of a 5th revision of the DLEP draft, which should be stable enough that CONFINE can start working on a standard-compliant implementation of DLEP.

7.2. CONFINE DLEP implementation

The CONFINE DLEP implementation is made from a group of plugins that are attached to a lightweight link layer database. This plugin design allows to adapt the implementation to different use-cases without modifying the code base.

The DLEP implementation has several plugins that gather link layer data and enter it into the database, either by querying the operating system about data of a local network interface (Wi-Fi, Ethernet) or by importing static data from a configuration file.

The core of the DLEP protocol itself is a pair of plugins (DLEP service and DLEP client) which can mirror the content of this database over an IP network between two devices. This can allow a research node to gather and process link layer data of hardware that is part of a different device, e.g. the attached Community Mesh node or an external radio connected by Ethernet. The client plugin also has the capability to automatically detect the presence of service devices on the local network, which should allow plug-and-play extension of Community Mesh Nodes with external radio devices in the future.

The information gathered by the plugins, directly from the local hardware or over DLEP from remote hardware can then be either processed on the application itself (e.g. for a routing metric) or can be supplied to other processes by different interfaces. The DLEP implementation has a plugin to export the content of the link layer database in text or JSON form for this.

8. Conclusions

This document presents an update of the architecture, design and development of the CONFINE testbed software system done during the second year of the project. It builds on the work reported in D2.1 during the first year of the project.

The improvements and extensions affect diverse aspects:

- The testbed controller: a ticket system for tracking issues with the testbed, monitoring of network connectivity, execution of operations in multiple nodes at once, improvements in the REST API for resource list filtering and updates in its architecture, structure and modularization, compliance with standards
- Design and development of a Common NodeDB for planning common, shared resources in a community network. The system can also import data from the current FunkFeuer database and also from the Guifi CNML data.
- Design and development of a new testbed monitoring system that collects node, sliver, slice and experiment metrics and accumulates a metrics history. This allows to assess individual and aggregated resource usage to select resources for experiments, analyse and detect anomalies in the operation of the testbed.
- Design and prototyping of a network (L2) topology virtualization prototype for experiments and thus present to the researchers a set of abstract and manageable L2 resources.
- Further steps in the integration of OMF with the testbed controller.
- Extensions and reimplementation of part of the node software following the controller API, with tools for monitoring and upgrading nodes.
- Evolution of the DLEP implementation for the testbed, providing researchers access to WiFi hardware and its configuration, according to the IETF Manet WG.
- Evolution of the virtual testbed (VCT) with containers that can run native node images, which provides a more faithful environment for testing and quality assurance.

This set of software developments with new and redesigned features can be found in the project repository at http://redmine.confine-project.eu. This new software combined with the continuous effort of integration, testing and maintenance has contributed to increase the stability, usability and functionality of the Community-Lab and the virtual (VCT) testbeds for research experiments.

9. References

[cryptopan] *Crypto-PAn Cryptography-based Prefix-preserving Anonymization*: http://www.cc.gatech.edu/computing/Telecomm/projects/cryptopan/

[FEDERICA] FEDERICA, Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures: http://www.fp7-federica.eu/

[FFM] *Common Node Database*, 2012-13: https://github.com/FFM/FFM

[OMF] The cOntrol and Management Framework: http://mytestbed.net/

[OpenFlow] OpenFlow: http://www.openflow.org/

[OpenWrt] OpenWrt: https://openwrt.org/

[PlanetLab] PlanetLab, an open platform for developing, deploying, and accessing planetary-scale services: https://www.planet-lab.org/

[PrivIP] “Address Allocation for Private Internets”, IETF RFC 1918: https://www.ietf.org/rfc/rfc1918.txt

[pycryptopan] *A python implementation of Crypto-PAn a IP anonymization ¡algorithm*: https://pypi.python.org/pypi/pycryptopan

[pyspkac] *Support for Netscape / HTML5 SPKAC client certificate request:* https://pypi.python.org/pypi/pyspkac

[rsclib] *rsclib: Utility Routines*, Ralf Schlatterbeck 2004-13; *rsclib: Utility Routines*: http://rsclib.sourceforge.net/

[Tapyr] *Tapyr Framework*, 2008-13: https://github.com/Tapyr/tapyr

[topo-github] *Visualizing OLSRD topography using d3* Software distribution, 2013: https://github.com/FFM/d3topo

The CONFINE project

September 2013

This document is licensed under the following license:

CC Attribution-Share Alike 3.0 Unported

<http://creativecommons.org/licenses/by-sa/3.0/>

CONFINE-201309-D2.3-1.0