Project title: Clommunity: A Community networking Cloud in a box

Pilot study evaluation

Deliverable number: D4.5

Version 1.1

This project has received funding from the European Union’s Seventh Programme for research, technological development and demonstration under grant agreement No 317879
Abstract

This document comprises the pilot study evaluation of the CLOMMUNITY project. Several services were deployed as pilots in the community network cloud and their performance was evaluated.
# Contents

1. Introduction 4  
   1.1. Contents of this deliverable 4  
   1.2. Relations of this document with other CLOMMUNITY tasks and deliverables 4  

2. Consolidated pilots 6  
   2.1. Distributed announcement and discovery of services (DADS) 6  
   2.2. Internet of Things (IoT) 14  
   2.3. Syncting web backup and file sync (SwebFS) 20  
   2.4. Tahoe Storage service 25  
   2.5. Peerstreamer video streamer 28  
   2.6. Decentralized video sharing 34  

3. Other pilot deployments 38  
   3.1. Infrastructure as a service (IaaS) for Community Clouds 38  
   3.2. MODAClouds Decisions Support System 45  

4. Evaluation 49  
   4.1. Introduction 49  
   4.2. IaaS 49  
   4.3. NaaS 49  
   4.4. PaaS 50  
   4.5. SaaS 51  
   4.6. Non-Technical 52  

5. Conclusions 53  

A. Pilots template 54  

Acronyms 56  

Bibliography 57  

Licence 58
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Search service scenario.</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>CAP triangle.</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>First results from deployment in testbed</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Results from deployment with user input</td>
<td>12</td>
</tr>
<tr>
<td>2.5</td>
<td>IoT integration</td>
<td>14</td>
</tr>
<tr>
<td>2.6</td>
<td>IoT structure</td>
<td>15</td>
</tr>
<tr>
<td>2.7</td>
<td>SCK Board Ambient</td>
<td>17</td>
</tr>
<tr>
<td>2.8</td>
<td>SCK Layer 1</td>
<td>18</td>
</tr>
<tr>
<td>2.9</td>
<td>SCK Layer 2</td>
<td>18</td>
</tr>
<tr>
<td>2.10</td>
<td>ThingSpeak on the Cloud</td>
<td>19</td>
</tr>
<tr>
<td>2.11</td>
<td>Tahoe-LAFS deployed in the CONFINE testbed.</td>
<td>25</td>
</tr>
<tr>
<td>2.12</td>
<td>Tahoe-LAFS deployed on heterogeneous devices.</td>
<td>27</td>
</tr>
<tr>
<td>2.13</td>
<td>Tahoe-LAFS and XtreemFS write and read performance without replication.</td>
<td>28</td>
</tr>
<tr>
<td>2.14</td>
<td>Tahoe-LAFS and XtreemFS write and read performance with replication.</td>
<td>28</td>
</tr>
<tr>
<td>2.15</td>
<td>Peerstreamer scenario used.</td>
<td>30</td>
</tr>
<tr>
<td>2.16</td>
<td>Results for UPC and ALL scenarios.</td>
<td>32</td>
</tr>
<tr>
<td>2.17</td>
<td>Comparison between averaged Chunks received and Chunks on time and not dupli-</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>cated (ALL).</td>
<td></td>
</tr>
<tr>
<td>2.18</td>
<td>Average Chunk rate received with each time frame (ALL).</td>
<td>33</td>
</tr>
<tr>
<td>2.19</td>
<td>Screenshot from the live event taken from one of the peers.</td>
<td>34</td>
</tr>
<tr>
<td>3.1</td>
<td>Open Web Panel interface screenshot.</td>
<td>41</td>
</tr>
<tr>
<td>3.2</td>
<td>LXC Web Panel interface screenshot.</td>
<td>42</td>
</tr>
<tr>
<td>3.3</td>
<td>phpVirtualBox interface screenshot.</td>
<td>42</td>
</tr>
<tr>
<td>3.4</td>
<td>Kimchi interface screenshot.</td>
<td>42</td>
</tr>
<tr>
<td>3.5</td>
<td>OpenVZ Web Panel in Cloudy's web interface</td>
<td>43</td>
</tr>
<tr>
<td>3.6</td>
<td>OpenVZ is announced to the CN</td>
<td>44</td>
</tr>
<tr>
<td>3.7</td>
<td>OpenVZ Web Panel screenshot showing the list of deployed virtual machines</td>
<td>44</td>
</tr>
<tr>
<td>3.8</td>
<td>OpenVZ Web Panel screenshot showing the details of a running virtual machine</td>
<td>44</td>
</tr>
<tr>
<td>3.9</td>
<td>Prototype DSS for supporting service selection in collaborative multi-clouds</td>
<td>47</td>
</tr>
<tr>
<td>A.1</td>
<td>Search service scenario.</td>
<td>54</td>
</tr>
</tbody>
</table>
## List of Tables

2.1. Different ThingSpeak middleware platforms running .................................. 16  
2.2. Comparison of SBCs used to run ThingSpeak ........................................... 20  
2.3. Estimated costs per user (monthly and yearly) of the Backup Syncthing (BST) service for different number of subscribed users ........................................... 25  
4.1. IaaS cases metrics and evaluation .............................................................. 50  
4.2. NaaS cases metrics and evaluation .............................................................. 50  
4.3. PaaS cases metrics and evaluation .............................................................. 51  
4.4. Non-technical cases metrics and evaluation ................................................ 52
1. Introduction

This deliverable belongs to Work Package 4 (WP4) (Experimentation and evaluation), in which the framework for experimentation and evaluation based on experimental large-scale deployments over the CLOMMUNITY community-networking testbed (cases, performance objectives, evaluation criteria, metrics) is defined, the operation of an experimental deployment of a community cloud system is supported and experiments required to validate research results implemented in a prototype software system (closing the RTD loop), based on i) research experiments, and ii) pilot experiments with real end-users from existing communities, are performed.

1.1. Contents of this deliverable

This document reports on the results of task T4.4 (Pilot experiments), where a set of pilot studies aiming at involving end-user groups from community networks were developed, based upon the results of T4.1 and the initial work of T4.2. The documentation of the development of the pilots was centralised in the project’s wiki\(^1\) following a common template. Nine pilots were initially proposed. At the time of writing of this deliverable, six of them have been classified as consolidated because they have involved end-users or are ready to do so. Of the remaining, two of them have been frozen at design state, and in one further development has not proceeded.

The remainder of this document is structured as follows. Chapter 2 integrates the documentation of the consolidated pilots and Chapter 3 integrates the documentation of the unfinished ones. Chapter 4 evaluates the results according to the metrics identified during the first reporting period (D4.1). The conclusions are presented in Chapter 5. Finally, the pilot’s template is presented in Appendix A.

1.2. Relations of this document with other CLOMMUNITY tasks and deliverables

This deliverable belongs to WP4 (Experimentation and evaluation) and receives inputs from the following deliverables:

- **D2.1** System software (year 1)
- **D3.1** Requirements for a holistic abstract network and service architecture
- **D3.2** Experimental research on community clouds (year 1)
- **D3.3** Experimental research on community clouds (year 2)
- **D4.1** Case definition and evaluation criteria
- **D4.2** Experimental community cloud testbed
- **D4.3** Experimental research evaluation (initial)
- **D4.4** Experimental research evaluation (final)

\(^1\)http://wiki.clommunity-project.eu/pilots:start
At the same time, it provides input to the following This deliverable is included in the context of WP4, which receives inputs from the following deliverables:

- **D2.1** System requirements and software architecture (initial)
- **D2.2** System software (year 1)
- **D3.1** Requirements for a holistic abstract network and service architecture
- **D3.2** Experimental research on community clouds (year 1)
- **D3.3** Experimental research on community clouds (year 2)
- **D5.1** IT and knowledge management tool
- **D5.2** Dissemination activities in year 1
2. Consolidated pilots

For the pilot studies reported in this chapter, we ran close to end-user services in order to assess them for usage in community network clouds.

2.1. Distributed announcement and discovery of services (DADS)

2.1.1. Introduction

The distributed, Decentralised and automatic Announcement and Discovery of services (DADS) is a key service in community networks (CN), which, among their main characteristics, are distributed and decentralised (Figure 2.1). With this service activated, users can publish information about the services they are offering to the community (e.g. type of service and characteristics) and also find out in real time what other users have published.

![Figure 2.1: Search service scenario.](image)

This pilot aims at studying the technical feasibility of deploying a distributed mechanism to announce and discover services in several CN nodes, which run Cloudy, and which are distributed all over the network. The Cloudy nodes will update with a certain periodicity the status of their services to a common data pool (i.e. a distributed database). At the same time, they are aware of the updates from other nodes in the CN in [quasi-]real time. This happens according to the microcloud/cloud topology of the CN. This mechanism eases the development of federation mechanisms and protocols for intra- and inter-cloud sharing of resources.

The information of the services is presented to the users on the web interface of the Cloudy nodes. Additionally, [non-]Cloudy applications are principally able to read this common data pool and feed a publicly accessible web server with updated information of the cloud services available through all the CN.

2.1.2. Partners

Guifi.net, UPC
2.1.3. State

During the initial development and deployment of Cloudy, each instance uses Avahi over a virtual layer-2 network built with TincVPN to announce and discover services in real time (like if all the Cloudy devices were in the same collision domain). While this implementation is suitable for small deployments, scalability problems appeared when the number of instances grew (from tens of nodes on), causing network loops and a high bandwidth usage. Additionally, in case of short network partitions (i.e. a few seconds), isolated instances would not retain the previously collected information (to do so, they should be saved in a local database) and, on recovery, a huge amount of traffic was generated.

The limitations imposed by the layer-2 network solution required an alternative solution for the DADS, based on a mechanism that automatically spreads the information over several distributed nodes that belong to different network collision domains.

After the experiments and the evaluation of the alternatives, Serf has been chosen as the best solution for DADS in the Cloudy environment and is currently in production.

2.1.4. Goals

The main goal of the pilot is to deploy and evaluate a distributed, decentralised and automatic announcement and discovery of services in a Community Network.

The pilot-specific goals are to:

- Study the compatibility of a distributed database-like solution with the current Avahi-over-Tinc implementation, as well as with the micro-cloud architecture, and current or envisioned federation mechanisms
- Examine the available distributed database solutions
- Find candidates for pilot experiments according to:
  - Hardware, software and network requirements (CPU, memory, OS, libraries, network bandwidth, RTT latency)
  - Tolerance and resilience to network splitting, data corruption, etc.
- Evaluate the candidates in a CN-like testbed
- Measure the performance of the selected solution under typical user usage

Regarding the contribution to the Clommunity project, the goals are to:

- Find a suitable distributed database solution for the announcement and discovery of services
- Integrate the database in the Cloudy distribution

2.1.5. Work plan

Work Plan Summary:

- Measure Avahi-Tinc performance ✓
- Examine available distributed database solutions ✓
- Select distributed database candidates for experiments ✓
- Evaluate distributed database candidates in a CN-like testbed ✓
- Evaluate the selected distributed database solution with CN users ✓
Distributed announcement and discovery of services (DADS)

- Compatibilize the DADS with the Avahi-over-Tinc implementation ✓
- Integrate the selected solution in the Cloudy distribution ✓

Detailed Work Plan (numbering across subsections):

2.1.5.1. Measurements of Avahi-Tinc performance

As mentioned above, the developed Avahi over a virtual layer-2 network built with the TincVPN solution to announce and discover services in real time has proven difficult to scale. CPU and bandwidth usage issues have appeared after the number of nodes has increased beyond tenths of nodes.

2.1.5.2. Examination of available distributed database solutions

There are several open source no-SQL databases that can be deployed in a distributed manner, allowing replication of data between different machines. To name a few of them:

- CouchDB
- MongoDB
- CaracalDB
- CouchBase
- Voldemort
- Cassandra
- Redis

Figure 2.2 (extracted from Scott Logic’s blog\(^1\)) is a visual guide to NoSQL databases positioning several well known solutions on the vertices of the CAP triangle (which stands for Consistency - Availability - Partition tolerance). The two desirable characteristics, from the CN network point of view and the type of usage foreseen, availability and partition tolerance are preferred over data consistency. In other terms, all the nodes need to have the DB information be available, even if it is not 100% consistent.

During the course of the Clommunity project, CoreOS has gained a certain notoriety (see references). Among the software provided, it ships etcd, an open-source distributed key value store (see references) that provides very interesting features like TTL-based key expiration.

Additionally, a decentralized solution for cluster membership, failure detection and orchestration called Serf, which claims to be lightweight and highly available, based on a gossip protocol, has appeared.

2.1.5.3. Selection of candidates for pilot experiments

The following software has been preliminary selected according to the criteria for experimentation defined in subsection 4 of this section:

- CouchDB
- MongoDB
- Etcd

\(^1\)http://www.scottlogic.com/blog/2014/08/04/mongodb-vs-couchdb.html
These four candidates provide, a priori, a convenient solution for deploying a common data pool to announce and discover services that can be integrated into Cloudy.

2.1.5.4. Evaluate distributed database candidates in a CN-like testbed

The criteria and metrics to be applied to the experiments are the following:

*Ease of integration with Debian 7 Wheezy and Cloudy*
- Existence of .deb package (Yes/No)
- Init.d integration (Yes/No)
- Existence of an automated installation process (Yes/No)
- Software running out of the box after installation without further configuration (Yes/No)
- Need for extra manual configuration (Yes/No)

*Hardware requirements and resources usage in a production-like scenario*
- CPU usage
- Memory footprint (avg, max)

*Tolerance to network-induced latency and bandwidth shortage*
- Bandwidth usage (avg, max)
- Data propagation timing (avg, max)
- Network delay tolerance (max)

*Data consistency*
- Recovery time in case of network failure, network splitting, etc.

2.1.5.5. Experiments design

Several experiments will be performed in order to determine for a suitable distributed DB solution:
2.1. Distributed announcement and discovery of services (DADS) 2. Consolidated pilots

- Ease of integration with Debian 7 Wheezy and Cloudy
  - .deb packages
  - binary releases
  - source code
- Hardware requirements and resources usage in a production-like scenario
  - low CPU and memory footprint
  - support for multiple architectures (i386, amd64, arm)
- Special software requirements/dependencies
  - required dependencies like Java, etc.
- Tolerance to network-induced latency and bandwidth shortage
- Recovery time in case of network failure, network splitting, etc.
- Other

A first round of experiments will address the issues stated above in a controlled environment, like Proxmox-based virtual machines running Cloudy in the lab premises.

After that, and considering the preliminary results obtained from the first round of experiments, a second set will be performed in a controlled testbed environment with virtualized Cloudy instances distributed on several locations of the community network.

2.1.5.6. Experimentation and results

The network between nodes will be solely used for service discovery communication. In order to automatically configure the service network, software developed by the Avahi project will be used.

In the experiments we will focus on the responsiveness of the discovery mechanism. We will consider responsiveness as the probability of successful operation within deadlines, which applied to our case refers to successful service discovery within time limits.

We will run the discovery requests from a single dedicated discovering node - or service client that will search and locate the service instances. The service client will be located in our UPC campus at Barcelona. All other nodes will act as service providers responding to discovery requests. All service providers will be spread between two community networks. Discovery times will be measured on the client directly before the request will be sent and directly after responses will be received to measure user-perceived responsiveness.

No nodes will join or leave the network, so no configuration on the network layers will occur during measurements which would interfere with discovery operation. In our case discovery of a service will be considered successful when all instances of a service will be discovered. Discoveries will considered failed (aborted) after 1 minute waiting time. After service discovery a client will have enough information in order to contact a service instance. Hence discovery in our case will be meaning resolving the IP address and port for every service instance. We will use different services for our experiments. Some of them will be Tahoe-LAFS distributed storage service, TincVPN service etc.

This table in figure 2.3 qualitatively summarizes the results of the experimental deployment of the four candidate softwares in a testbed environment.

The two first candidates (CouchDB and MongoDB) are designed for environments very different to community networks and for completely different usage scenarios. They offer lots of options and
2. Consolidated pilots

2.1. Distributed announcement and discovery of services (DADS)

### Table: Tested version, Integration with Debian/Cloudy, Hardware requirements, Software requirements, Tolerance, Recovery, Other

<table>
<thead>
<tr>
<th></th>
<th>Tested version</th>
<th>Integration with Debian/Cloudy</th>
<th>Hardware requirements</th>
<th>Software requirements</th>
<th>Tolerance</th>
<th>Recovery</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>CouchDB</td>
<td>1.5.1</td>
<td>No Debian packages (download and compile source code), Detailed instructions available</td>
<td>Moderate</td>
<td>None</td>
<td>Good</td>
<td>Average</td>
<td>Too sophisticated for Cloudy’s needs</td>
</tr>
<tr>
<td>MongoDB</td>
<td>2.2.7</td>
<td>i386 and amd64, deb packages, no arm, Source code, Detailed instructions available</td>
<td>Moderate</td>
<td>None</td>
<td>Average</td>
<td>Good</td>
<td>Too sophisticated for Cloudy’s needs</td>
</tr>
<tr>
<td>Etcd</td>
<td>0.4.6</td>
<td>Zip file with i386 and amd64 executables, no arm, Source code</td>
<td>Low</td>
<td>None</td>
<td>Good</td>
<td>Very long recovery times in case of a node exiting and joining back a cluster</td>
<td>Interesting features like data expiration. Not mature enough yet</td>
</tr>
<tr>
<td>Serf</td>
<td>0.6.1</td>
<td>Zip file with i386, amd64, arm, Source code</td>
<td>Low</td>
<td>None</td>
<td>Very good</td>
<td>Good</td>
<td>Very lightweight, Limited storage space (512 B)</td>
</tr>
</tbody>
</table>

**Figure 2.3:** First results from deployment in testbed
[source: http://wiki.clommunity-project.eu/pilots:distributed_announcement_and_discovery]

features that are far beyond the needs of the pilot. Therefore, they are not efficient for the desired use case.

Etcd (which is described as a highly-available key-value store for shared configuration and service discovery) has performed very well in the testbed experiments. Synchronization between nodes is fast and efficient, and new nodes can join and leave the cluster easily. However, when network partitions occur, the recovery times are too high. Etcd has a very interesting feature, the data expiration field, that allows setting a certain duration to a key-value stored in the database. This is very convenient for real-time services announcement: if the service goes off-line, after a certain timeout the information disappears from the common data pool. The very long recovery time, however, makes it unsuitable for using it in Cloudy. The software is very recent and under active development, but it is not mature enough yet.

Serf (which is described as a decentralized solution for cluster membership, failure detection, and orchestration, lightweight and highly available) has also performed very well in the testbed experiments. Synchronization between nodes is also fast and efficient, using approximately 1kB/s of download+upload bandwidth per node with 10 nodes active. Compared to Etcd, it does not have the data expiration feature but, if a node goes offline, the information it published is removed from the rest of the nodes in the cluster after a certain timeout. Therefore, in terms of user experience, the result is very similar. Additionally, recovery times in case of network partition are almost immediate and very efficient, since Serf uses a distributed gossip protocol to communicate between nodes. Serf is very lightweight and is binary-available for i386, amd64 and arm. However, the amount of data each node can publish is limited to 512 bytes. The software is very recent and under active development, but it seems to be mature enough for the pilot’s requirement.

### 2.1.5.7. Evaluate the selected distributed database solution with CN users

Five early adopters of Cloudy inside Guifi.net have been asked to try out a development version of the DADS service built with Serf and compare it with Avahi-over-Tinc current solution. The goal of
this experiment is, more than characterizing Serf’s performance, to receive feedback and suggestions from the users regarding their experience.

Usage differences between using Avahi-over-Tinc and Serf

- Number and quality of the results
- Time to retrieve results
- Other feedback and suggestions

Results: The table shown in figure 2.4 qualitatively summarizes the results of the experiments involving community network users in Guifi.net:

![Figure 2.4: Results from deployment with user input](source: http://wiki.cloommunity-project.eu/pilots:distributed_announcement_and_discovery)

### 2.1.5.8. Compatibilize the DADS with the Avahi-over-Tinc implementation

The Cloudy distribution, including the web interface, has been built using a modular design that eases the addition of new software pieces. Shipped with the distribution comes a tool, which includes a series of scheduled scripts to manage the announcement of services. This tool checks periodically the status of the installed services and, according to their configuration, publishes them to the community network. The publication can be simultaneously performed using more than one mechanism, which integrate with the management tool in the form of plug-ins.

The DADS tool, based on Serf, has been built as a plug-in for Cloudy that integrates both with the web interface and with the services manager tool. This means that, once is installed, it can be activated or deactivated from its specific page on the web interface and works in parallel with other announcement and discovery mechanisms (like Avahi-over-Tinc) or any other which could be added later.

### 2.1.6. Dependencies

In the case that other partners would like to have their software included in the pilot (e.g. CaracalDB), they should provide it ready to install as a regular Debian (.deb) package.
2. Consolidated pilots  

2.1. Distributed announcement and discovery of services (DADS)

2.1.7. Results

The following papers have been published covering (among other topics) the results of the described evaluations:


- **Cloud Services in the Guifi.net Community Network**. Mennan Selimi, Amin M. Khan, Emmanouil Dimogerontakis, Felix Freitag, Roger Pueyo Centelles. Special Issue on Community Networks - Elsevier Computer Networks Journal. Accepted for publication.

2.1.8. Conclusions

Pilot-specific conclusions:

From the technical point of view, given the requirements of the service announcement and discovery in terms of information volume (size), number of read/write requests per second, etc. and the volatility of the underlying community network, the DADS pilot has shown that most of the distributed database solutions are not suitable. In many cases, these applications are designed to handle a huge amount of requests per second. Others are designed to operate over distributed resources but with a centralized control.

Related to the users of the Guifi.net community network experience with the DADS, the automation of the services publishing and discovering process has been very positive. First, many of the tedious tasks like manually registering the Cloudy node on www.guifi.net and then registering the active services one by one has been simplified to (literally) a few clicks from within the Cloudy web interface. Second, the ability do discover services in real time, allowed to have an up-to-date information about their status and some simple metrics about the service quality (at least about the network round trip time between the user and the service).

Contribution of the pilot to the Clommunity project:

Most of the available open source cloud-oriented software products are designed to operate in data center-like cloud environments, where the underlying network performance provides an excellent connectivity (e.g. Gigabit connectivity with low and constant delay). Therefore, when they are moved to the more challenging community network environment, their performance is poor or they are unable to operate reliably. This behaviour was already experienced during the deployment of distributed storage solutions in the community network cloud environment, and should be considered when analysing or deploying new software.

2.1.9. References

CoreOS is a Linux distribution that has been rearchitected to provide features needed to run modern infrastructure stacks. The strategies and architectures that influence CoreOS allow companies like Google, Facebook and Twitter to run their services at scale with high resilience.

etcd is an open-source distributed key value store that provides the backbone of CoreOS clusters and the etcd client runs on each machine in a cluster. etcd gracefully handles master election during network partitions and the loss of the current master.

Deliverable D4.5
2.2. Internet of Things (IoT)

2.2.1. Introduction

The Internet of Things (IoT) is a fast growing technology that allows users to gather data from the physical environment. The deployment of IoT nodes presents issues in terms of data privacy and access to the data. A new solution is presented in this work where an IoT platform was deployed to measure air quality in the framework of the Clommunity project. The nodes were installed in a community network, using the existing network infrastructure. Services were deployed using the Cloudy system based on open software and hardware. The data measured in the community network were made available to members of the community. The IoT integration among the real physical world, digital world and the virtual cyber world is illustrated in fig. 2.5.

![IoT integration](image_url)

**Figure 2.5: IoT integration**

This pilot aims at studying the technical feasibility of deploying wireless sensors to measure air quality and weather conditions in a community and sharing data in the cloud in a controlled way. IoT offers the possibility of accessing data whenever it is required and from anywhere in the community. In this project we incorporate IoT to community networks (CNs) making use of open software and hardware. That is, code and devices can be freely modified by the end user, since their designs are publicly accessible. The layout of the sensors networks and related cloud infrastructure is shown in fig. 2.6.
IoT in community networks (CNs) offers the possibility of accessing data whenever it is required from inside the community and from anywhere in the world by users with the proper credentials. Data from sensors measuring the physical environment in a microcloud are stored and displayed directly on an on-line system installed in the local Microcloud or on another part of the Community Network.

2.2.2. Partner

- ICTP - The Abdus Salam International Centre for Theoretical Physics.

2.2.3. State

- Status: finished

The IoT project is being tested in different places. Final users are pushing data from devices to the server in the cloud using GuiFi.net. There are two sensing boards at the ICTP laboratory in Trieste. At Universita politecnica de Catalunya (UPC) there is a board which pushes data to the server. Wireless devices (SCK) with sensors purchased from SmartCitizen are programmed to send data to a server installed in the Cloud. SCK has different embedded sensors able to measure CO and NO2 concentrations, temperature, humidity, light and sound.

Currently a MiddleWare (MW) system called ThingSpeak (TS) is installed in a machine in the Cloud. Its step by step installation is detailed here[[howto:configurets—Install ThingSpeak on clean install of Ubuntu 12.04 LTS]]. This MW stores offers the possibility to the community users to have their own account by means of which they can store their own data or access somebody’s else, provided they have the required credentials. This allows to store, share and show data from sensors making data public or private as desired.

Three different servers were installed, two in single board computers at ICTP Trieste, Italy and another one in a computer at UPC Barcelona Spain as seen on Table 2.1.
2.2. Internet of Things (IoT)

<table>
<thead>
<tr>
<th>Ubication</th>
<th>Name</th>
<th>Address</th>
<th>Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPC</td>
<td>TS</td>
<td>84.88.85.11:3000</td>
<td>desktop server (Public IP address)</td>
</tr>
<tr>
<td>ICTP</td>
<td>TSI</td>
<td>10.95.00.24:3000</td>
<td>RaspBerry Pi (Private IP address)</td>
</tr>
<tr>
<td>ICTP</td>
<td>TSI2</td>
<td>10.95.00.25:3000</td>
<td>BeagleBone Balck (Private IP address)</td>
</tr>
</tbody>
</table>

2.2.4. Goals

CNs participants used to have problems creating and sharing applications and services of common interest. This was due to the lack of cost effective solutions. Cloudy fulfills this gap, and the capability to integrate IoT further empowers the users with information about air quality and other environmental data of interest. This information can also in the future be used to predict weather condition. The quality of the air is a major concern in many cities and the availability of hard data in this regard is very important, since it can be used by the citizens to lobby for improvements. The platform can also be extended with additional sensors for other applications.

Deploying an IoT open infrastructure inside CNs let users manage stored information on-line, with password controlled access, adding privacy and speed to the process.

There are 21 weather station in Barcelona (according to www.aemet.es) using IoT and wireless sensors connected to the cloud there would be more information about weather and air quality. This allows:

- More precise weather prediction in the future.
- Locate the pollution "hot Spots” in the community.
- Share data among users
- Assess the air quality in any place of the community.
- Control who has access to which kind of information collected.

It is important to have privacy in the information because it could affect users. For example a contaminated area could decrease the value of a property.

2.2.5. Work plan

A deployment was done in Barcelona to test sensors and IoT platform installed in Clommunity. This will allow to check connection among different sensors and the Cloud, storing data in the cloud.

The device developed by SmartCitizen is not calibrated. A calibration has to be done in order to get useful measurements.

2.2.6. Progress

Sensors are ready to send data to the middleware ThingSpeak. ThingSpeak is installed in different platforms, desktop PC and two single board computers (BeageBone Black and Raspberry Pi Model A) to assess its performance.
2.2.7. Experiments design

The Smart Citizen Kit (SCK) is an electronic board based on Arduino, equipped with the following sensors:

- Air Quality (CO & NO2 concentration)
- Temperature
- Sound
- Humidity
- Light Quantity

The board is equipped with a WiFi radio that allows to upload data from the sensors in real time to an online platform. In addition, the board contains a voltage regulator that allows it to be connected to a photovoltaic panel, so it can be installed without depending on the electrical grid.

The hardware is composed of two stacked printed-circuit boards: an interchangeable daughterboard or shield, and an arduino-compatible data-processing board. We have nicknamed the shield developed for this campaign ‘The Ambient Board.’ As the name suggests, it carries sensors that measure air composition (CO and NO2), temperature, light intensity, sound levels, and humidity. Once it’s set up, the ambient board is able to stream data measured by the sensors over WiFi module on the data-processing board. The device’s low power consumption allows for placing it on balconies and windowsills. Power to the device can be provided by a solar panel and/or battery.

![SCK Board Ambient](image)

**Figure 2.7: SCK Board Ambient**

2.2.8. ThinkSpeak Middleware

ThingSpeak allows developers to interact with devices using standard Web technologies. ThingSpeak can be run via its free hosted service or on personal servers.

Features of the ThingSpeak platform include data logging, data processing, data distribution, location-based services, status updates, social network integration, apps, and plugins.
2.2.9. Experimentation

ThinkSpeak server running on the cloud in Spain.
For this project different SBCs were used, Raspberry Pi (models A and B) and BeagleBone Black. A description of SBCs used for the project is shown in Table 2.2.

2.2.10. Results

Data online from sensors connected to the cloud can be accessed in this links:
- http://84.88.85.11:3000/channels/8
2. Consolidated pilots

2.2. Internet of Things (IoT)

Figure 2.10: ThingSpeak on the Cloud

- http://84.88.85.11:3000/channels/9
- http://84.88.85.11:3000/channels/18

The free and open source Middleware installed in the CN using the Cloudy system achieves the integration of physical environment data with a high level of data privacy. It also increases speed and increases efficiency to access to data stored and makes IoT on CNs less dependent on external sources.
Table 2.2: Comparison of SBCs used to run ThingSpeak

<table>
<thead>
<tr>
<th>SBCs</th>
<th>BeagleBone</th>
<th>RaspBerry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Black</td>
<td>A+</td>
</tr>
<tr>
<td>Microproc.</td>
<td>1GHz</td>
<td>700MHz</td>
</tr>
<tr>
<td></td>
<td>ARM</td>
<td>ARM</td>
</tr>
<tr>
<td></td>
<td>CORTEX A8</td>
<td>1176JZPS</td>
</tr>
<tr>
<td>RAM</td>
<td>512MB</td>
<td>512MB</td>
</tr>
</tbody>
</table>

resources.

2.2.11. Conclusion

Sensors were tested using Guifi.net CN in the Marconi Laboratory at ICTP, Italy and also in the Department of Computer Architecture at UPC, Spain. They were later deployed by end users in Barcelona. There are two different TS servers deployed in Guifi.net CN, one in Italy and the other one in Spain. The user has a choice about which one to leverage in a particular application.

2.3. Syncthing web backup and file sync (SwebFS)

2.3.1. Introduction

SwebFS is a pilot that analyses the provision of a Backup and File Sync (BFS) service via a web interface for Community Networks (CNs). File synchronisation between different devices is nowadays a common service, since Internet connections become ubiquitous and most of the desktop computers and portable devices are on-line by default.

Syncthing replaces proprietary sync and cloud services with something open, trustworthy and decentralized. Your data is your data alone and you deserve to choose where it is stored, if it is shared with some third party and how it's transmitted over the Internet.\(^2\)

The BFS service is provided by means of BST\(^3\), a web application that manages the provision of remote, private Syncthing instances for end users with Docker containers.

2.3.2. Partners

This pilot has been conducted by Guifi.net in collaboration with Routek S.L, an SME participating in the Guifi CN that has contributed hardware resources.

2.3.3. State

The pilot has been completed with the participation of 10 users from the Guifi.net CN.

\(^2\)Syncthing: [http://www.syncthing.net](http://www.syncthing.net)

\(^3\)BST: [https://github.com/Clommunity/bst-mux](https://github.com/Clommunity/bst-mux)
2.3.4. Goals

The main goal of the pilot is to *deploy and evaluate* a BFS service via a web interface for CNs involving *real end users* to assess its technical and economic viability. The following lists contain a number of more specific objectives:

The *pilot-specific* goals are to:

- Identify the user requirements related to backup and file synchronization between devices
- Examine the available [commercial] closed source solutions
- Analyse current open-source software on which to base the SwefBS service
- Involve real CN users in the pilot
- Evaluate the performance of the solution proposed in terms of hardware resources (CPU, RAM, bandwidth, etc.)

Regarding the contribution to the Community project, the goals are to:

- Integrate the solution in the Cloudy distribution
- Evaluate the economic requirements and viability of the service
  - Hardware and maintenance costs (bootstrapping costs, per user costs, etc.)
  - Business opportunities

2.3.5. Work plan and progress

The pilot consists of several sequential tasks, listed below:

- Service definition ✓
- BST development (web application for ST instances provision) ✓
- Hardware allocation ✓
- Pilot environment deployment ✓
- User engagement ✓
- Usage monitoring ✓
- Economic approach in different scenarios ✓

All the tasks have been completed.

2.3.6. State of the art

There are several cloud-based, commercial providers that offer a BFS services, like Dropbox\(^4\), Google Drive\(^5\), SpiderOak\(^6\), Box\(^7\), SugarSync\(^8\), etc. Most of them offer a *freemium* service, with a limited amount of storage space available, to engage customers into one of the payed subscription plans offering more space. These services are based, however, on closed-source proprietary solutions (e.g. the Dropbox client) and the data is hosted in countries where the applicable legislation may differ from that of the user.

\(^4\) Dropbox: [http://www.dropbox.com](http://www.dropbox.com)
\(^5\) Google Drive: [http://drive.google.com](http://drive.google.com)
\(^6\) SpiderOak: [http://spideroak.com](http://spideroak.com)
\(^7\) Box: [http://www.box.com](http://www.box.com)
\(^8\) SugarSync: [http://www.sugarsync.com](http://www.sugarsync.com)
As an alternative, few open-source software solutions provide files synchronization between devices (e.g. rsync\(^9\)). To our knowledge, only one open source application provides easy multi-device file synchronization without the need for a central server (Syncthing), and a few open source solutions that are based on a central server (e.g. OwnCloud\(^{10}\), Seafile\(^{11}\), SparkleShare\(^{12}\)).

### 2.3.7. Experiments setup

An experimental SwebFS server with BST has been deployed to perform the experiments with the following hardware specifications:

- **CPU:** Intel Core i7-3770 @ 3.40GHz (8 cores)
- **RAM:** 8 GB
- **HDD:** 1 TB
- **Network:** 100base T/FD connection to Guifi.net fiber backbone

Regarding the number of users involved and the application instances required:

- **Participating users:** 10
- **Syncthing instances:** 10 (running in isolated Linux Containers (LXC)\(^{13}\) instances provided by Docker\(^{14}\))

The following performance parameters have been studied:

- **CPU usage**
- **RAM consumption**
- **Network traffic**

for these cases:

- **On idle**
- **Active file synchronisation**
- **Daily average**
- **Weekly average**

as a per user average.

### 2.3.8. Experimentation

The experimentation phase consisted in monitoring during a period of two weeks the required resources for providing the SwebFS service to the 10 participating users. To do so, the users were required to sign up to the BST service, install Syncthing on at least one of their devices (desktop computer, laptop, mobile device, etc.), link them and share at least a new repository between them.

Users were asked to copy to the repository shared between their devices and the BST service at least 1 GB of data of any kind during the first week and use this service to synchronise and backup their files as they would do with any other commercial service they may be using. Since no specific measures

\(^{9}\)rsync: https://rsync.samba.org/

\(^{10}\)OwnCloud: http://owncloud.org/

\(^{11}\)Seafile: http://seafile.com

\(^{12}\)SparkleShare: http://sparkleshare.org/

\(^{13}\)LXC: http://www.linuxcontainers.org

\(^{14}\)Docker: http://www.docker.com
were taken for this pilot to ensure privacy and security, they were recommended not to provide files with personal information or that they wouldn’t like to be leaked.

2.3.9. Results

These are the measurements obtained during the experimentation phase.

2.3.9.1. Resources utilisation on idle

Resources utilisation on idle:\[\]

**CPU:** < 0.01 \times CPU_{core}

**RAM:** 12.5MB

**Network:** < 0.25kbps

2.3.9.2. Resources utilisation on active synchronisation

Resources utilisation on active synchronisation:\[\]

**CPU max:** 0.21 \times core

**CPU avg:** 0.11 \times core

**RAM:** 27.6MB

**Network:** N/A

2.3.9.3. Resources utilisation on daily average

Resources utilisation on daily average:\[\]

**CPU:** 0.023 \times core

**RAM:** 17.2MB

**Network:** N/A

2.3.9.4. Resources utilisation on weekly average

Resources utilisation on weekly average:\[\]

**CPU:** 0.022 \times core

**RAM:** 17.6MB

**Network:** N/A

---

\[15\] Averaged over 10 LXC Syncthing instances

\[16\] Averaged over 10 LXC Syncthing instances

\[17\] Average over 10\textsuperscript{th} day of the experiment

\[18\] Average over 2\textsuperscript{nd} week of the experiment

Deliverable D4.5
2.3. Syncthing web backup and file sync (SwebFS)

2.3.10. Economic approach

2.3.10.1. Hardware requirements

According to the experimental results, a server with similar hardware specifications to that used in the experiment could provide service to at least 300 users. This rough estimation is obtained from the RAM memory requirements during active sync, in case all the users were simultaneously using the service at full load: $\frac{8 \text{ GB}}{27.6 \text{ MB/instance}} = 296.8 \text{ instances} \geq 300 \text{ users}$. RAM memory would be in this case the most restrictive resource while the CPU could be overloaded (leading only to longer response times).

The worst scenario considered above, however, is very unlikely to occur at any time. The service resources could probably be oversubscribed with a 2:1 ratio without affecting the quality of the service.

In terms of storage capacity, it has been estimated that an average user would store 10 GB of files. Therefore, a total of 6 TB would be needed to host the users’ data.

2.3.10.2. Setup costs

The price for setting up a server with similar specifications to the one used in the experiments in a datacenter facility, including the required storage capacity (including disks redundancy) and other specific hardware, plus the personnel costs to maintain it on-line during three years, is estimated of 3300 €.

2.3.10.3. Operation costs

The average electricity consumption of the server is estimated to be around 120 W, leading to a yearly consumption of $135 \text{ W} \times 24 \text{ h} \times 365 \text{ days} = 1182.6 \text{ kWh}$. With an estimated cost of 0.15€/kWh, the yearly electricity bill is expected to be around 177.39 €.

Bandwidth usage inside the CN is provided for free. Given the fact that the service is run for profit and that it generates large amounts of traffic, it is considered convenient to contribute a yearly fee to the network maintenance and improvement. We estimate this contribution to be of 250 €.

In terms of personnel costs, the service is expected to demand 2 hours/week, resulting in a yearly cost of 3640 €\textsuperscript{19}.

Finally, 400 €are added yearly to cover unexpected issues and replace failing hardware.

The total yearly running cost is estimated to be $177.39 + 250 + 3640 + 400 = 4467.39€$.

2.3.10.4. Pricing

Considering the hardware setup cost of 3300 €, a yearly operation cost of 4467.39 € and an amortization period of three years, running the service is estimated to cost 5567.39 € per year.

Table 2.3 summarizes the pricing for the service to make it sustainable depending on the number of users subscribed:

\textsuperscript{19}Work is charged at 35€/h
### Table 2.3: Estimated costs per user (monthly and yearly) of the BST service for different number of subscribed users

<table>
<thead>
<tr>
<th>Subscribed Users</th>
<th>Monthly Cost (€)</th>
<th>Yearly Cost (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 users</td>
<td>9.28</td>
<td>111.35</td>
</tr>
<tr>
<td>100 users</td>
<td>4.64</td>
<td>55.67</td>
</tr>
<tr>
<td>200 users</td>
<td>2.32</td>
<td>27.84</td>
</tr>
<tr>
<td>300 users</td>
<td>1.55</td>
<td>18.56</td>
</tr>
<tr>
<td>600 users</td>
<td>0.77</td>
<td>9.28</td>
</tr>
</tbody>
</table>

2.3.11. Conclusions

The technical measurements allowed to obtain data for estimating costs and prices for the commercial approach. A commercial service becomes depends on the number of users. If the service is used by more than 100 users, the monthly fee seems to become reasonable to be attractive for end users.

### 2.4. Tahoe Storage service

#### 2.4.1. Introduction

Tahoe-LAFS is a decentralized storage system with provider-independent security. This means that the user is the only one who can view or modify the data. Since the CLOMMUNITY project uses the CONFINE testbed, we can use these nodes to implement Tahoe-LAFS services. Clients, introducers, gateways (optional) and servers are inside the slivers/VMs of the testbed. For more about the Tahoe-LAFS architecture, please refer to (https://wiki.clommunity-project.eu/tahoe:research).

#### 2.4.2. Partner

UPC  
Guifi.net

#### 2.4.3. State

CLOMMUNITY uses the CONFINE testbed. Therefore, we can use its nodes to implement services of Tahoe-LAFS. Figure 2.11 illustrates the deployment of Tahoe-LAFS in the CONFINE testbed.

**Figure 2.11:** Tahoe-LAFS deployed in the CONFINE testbed.
Tahoe-LAFS provides to the project a distributed file system with encryption and privacy. It is easy to use at the client side. Privacy is assured since the storage server operator can not see the stored data nor knows about a file name or content. That only can be known by using an URI.

In order to use Tahoe-LAFS in Clommunity, first we need to setup our own storage servers and a gateway. The users can store securely files. It works internally as a RAID, distributing the data across the various storage servers, and the user only knows a directory mounted via sshfs in the simplest use case. Clients, introducers, and servers can be inside the slivers of the testbed. We also need a gateway which can be on the client side. Storage servers can be added or removed at any time.

The simplest use case requires a client (with local gateway), a introducer and storage servers (2 or more, since we want test data split). The client can be behind NAT. The introducer and the storage servers have to accept incoming connections from clients and servers, respectively.

In the case of the testbed, the introducer can also be an added service in the place where the confine-controller software is installed.

Tahoe clients can be placed inside the slivers and behind NAT if the network is IPv4.

The gateway is local to the client. It can be placed inside the sliver. If it is shared, it can also be placed in the same place as the confine-controller software. Remember that placing the gateway outside the client allows to share URIs, but the data can be seen “clear” (plain text) by the gateway operators. If the client wants to share local files, external access to gateway have to be provided by the user (no NAT, port redirect is not needed).

### 2.4.4. Goals

The proposal is to assess Tahoe-LAFS for inclusion in the services provided by Cloudy.

### 2.4.5. State of the art

The widely used alternative is Freenet, which differs from Tahoe-LAFS. Freenet and Tahoe grew together. Today, the main difference is that Tahoe does not rely on strange storage servers as does Freenet. There are other alternatives like XtremFS, GlusterFS, Ceph etc. The difference between Tahoe-LAFS and Freenet is explained at Zooko’s post in the Tahoe Web site).

### 2.4.6. Experiment design

Allowing users in a community network to share and use the storage of other users in a reliable, secure, and privacy-preserving way, is of a great importance. For this reason, we use Tahoe-LAFS as a main storage service in Cloudy. Our objective is to evaluate the Tahoe-LAFS storage service in community network clouds. The community network cloud is deployed in Guifi.net. The experiments are performed in two ways: in the first way the workloads that are used in the experiments are generated manually from our side. In the second way, they are generated by a popular storage benchmark tool called IOzone.

### 2.4.7. Experimentation

For our first experiment, the main configuration includes 43 nodes of the Guifi.net community network. The nodes of our experiments are real nodes with Guifi.net IPs, connected though wireless IEEE 802.11 a/b/g/n technology, using equipment from various manufacturers, while different routing
protocols are used on some zones of the network where our nodes are located. We use 10 nodes from each three different local clouds located in the Catalonia region of Spain as illustrated in figure 2.12 (UPC local cloud, HAN local cloud and TAR local cloud). We use 4 nodes from the KTH local cloud located in Stockholm, Sweden, and 2 nodes from the ICTP local cloud located in Trieste, Italy. We use 4 physical nodes from Community-Lab (testbed for experimentation with network technologies and services for community networks) and three Intel Galileo boards from the Internet-of-Things (IoT) domain. We observe that network characteristics are not symmetric between the local clouds and devices. Our Tahoe-LAFS and XtreemFS clusters therefore consist of 43 nodes. For Tahoe-LAFS, the introducer node and client node are located in the UPC local cloud (in Barcelona). For XtreemFS the client node, MRC node and DIR node (central registry for all services in XtreemFS), are also located in the UPC local cloud. Each VM in every machine shares 10GB of its storage to the Tahoe-LAFS storage cluster and 10GB to the XtreemFS storage cluster. In the first experiment we use 3 different workloads of 100K, 500K and 1M file sizes. We measure the sequential write and read performance of Tahoe-LAFS and XtreemFS in community networks production environment.

Figure 2.12: Tahoe-LAFS deployed on heterogeneous devices.

2.4.8. Results

Figure 2.13 shows the sequential write and read performance of Tahoe-LAFS and XtreemFS in Volume0 setting with no replication. It can be seen that Tahoe-LAFS write performance is lower than XtreemFS. This comes from the fact that Tahoe-LAFS performs expensive cryptographic operations during write operation (the file is encrypted at the client side, erasure coded and distributed across storage nodes). One of the reasons that XtreemFS performs better might be that since it does not use encryption at the client side, it encrypts the data only on the network (files are still stored in clear-text on the nodes). It is interesting to note that when writing 100K files, Tahoe-LAFS performs better than XtreemFS, and this can be attributed to the fact that the default stripe size of Tahoe-LAFS is well optimized for writing small objects (the stripe size determines the granularity at which data is being encrypted and erasure coded). Tahoe-LAFS stores shares of files on a number of storage nodes chosen in a kind of round-robin way; read access uses fastest servers (Tahoe-LAFS is unaware of the node “location”). XtreemFS provides replica selection policies and this feature reflects in read performance of XtreemFS.

As we increase the number of replicas to 2 and 5, the write performance of Tahoe-LAFS and XtreemFS decreases drastically (Figure 2.14). Since we deal with heterogeneous hardware devices...
2.4.9. Conclusions

The purpose of this pilot was to evaluate the performance and feasibility of Tahoe-LAFS and XtreemFS on our cloud infrastructure. The distributed file systems for the experiment were selected according to their potential relevance for community network users, covering solutions for different requirements regarding fault-tolerance, storage performance, and privacy. The study aimed to provide an understanding on using distributed file systems in distributed heterogeneous cloud-based infrastructures in a production community network.

2.4.10. References


2.5. Peerstreamer video streamer

2.5.1. Introduction

Peerstreamer is an application which allows users to stream a video in a P2P network overlay. When a peer connects to a video stream (i.e: another peer), a mesh network is created between the different...
peers and video chunks are exchanged. Peerstreamer is running in the Cloudy distro, which is installed in the slivers or VM of the testbed.

Peerstreamer was originally developed by the University of Trento, Italy. For more information about the original project, please refer to http://peerstreamer.org/.

2.5.2. Partner

UPC, Guifi

2.5.3. State

For the time being, users can share video streams in a very easy way, since Peerstreamer instances are announced within a microcloud by Avahi. While Peerstreamer is running well, but is has shown however some small issues that should be fixed.

2.5.4. Goals

The goal of Peerstreamer is to allow users to live-stream video in a comfortable way in a community network.

The pilot-specific goals are:

- Understand how a video is streamed in P2P fashion without using the client server model
- Simplify possible complications that end-user may face when streaming a video
- Observe hardware and CPU usage
- Fix any bug or non-desirable behaviour of the application

Contribution to the Clommunity project:

- Provide to the users of Cloudy the tools to stream video with PeerStreamer
- Finalize the integration of Peerstreamer to work well in the Cloudy distribution.

2.5.5. Work plan

The steps for the integration of Peerstreamer in Cloudy are the following:

- How is Peerstreamer organized
- Understand the Peerstreamer infrastructure
- How does Peerstreamer work
- How is a video streamed in a P2P network
- Different ways of streaming a video
- Peerstreamer core and essentials
- Involving Avahi in Peerstreamer service publication
- Integration in Cloudy

Deliverable D4.5
2.5.6. Progress

PeerStreamer is already included in the Cloudy distribution. Installing, publishing video streams, connecting to a peers are functionalities that are implemented and working.

2.5.7. State of the art

In terms of evaluating the performance of PeerStreamer in unreliable networks, the work of Baldesi et al.\textsuperscript{20} is the most relevant to our work. The authors evaluate PeerStreamer, a P2P video streaming platform, on the Community-Lab, the wireless community network (WCN) testbed of the EU FIRE project CONFINE. Their experiments highlight the feasibility of P2P video streaming, but they also show that the streaming platform must be tailored ad-hoc for the WCN itself to be able to fully adapt and exploit its features and overcome its limitations. However, the authors evaluated with a limited number of nodes (16 Guifi.net nodes), which were located in the city of Barcelona.

2.5.8. Experiments design

For our experiments, the main configuration includes nodes of two geographically distant CNs: Guifi.net in Spain and AWMN (Athens Wireless Metropolitan Network) in Greece. For our scenarios, we construct our topology as shown in figure 2.15, to easily manage the connectivity of the nodes where the camera streams to a local PeerStreamer service, acting as the peer source, converting the stream to be used on the P2P PeerStreamer network, and thus each of the nodes only need to connect as a peers to this source. Also, it is defined that each chunk contains one forth of a frame of the video, and that the source can generate and send around 104 chunks per second.

![Figure 2.15: Peerstreamer scenario used.](image)

2.5.9. Evaluation metrics

For our metrics, we choose RTT for time and location, chunk rate for data on network behaviour, chunks received for data coming from the P2P network, played out ratio for quality of image, and

\hspace{\textsuperscript{20}}http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6849101
neighbourhood size for quality of the P2P network.

2.5.10. Dependencies

**Avahi**: Avahi is a framework for service discovery on local networks, which uses the same specifications as Apple’s Bonjour, Multicast DNS (mDNS) and DNS Service Discovery (DNSSD) from the Zero Configuration Networking (Zeroconf) working group. Avahi allows programs to publish and discover services that are available on other machines. As an example, a user can find local printers without the need to know their IP address. However, the multicast-based design does not allow the Avahi service to reach beyond the local link which is the case in CNs where services are spread over different nodes that belong to different broadcast domains. In this environment, Avahi packets can not be directly exchanged from one node to another.

**TincVPN**: To solve the problem of Avahi packets, we use TincVPN, a Virtual Private Network (VPN) daemon that uses tunnelling and encryption to create a secure private layer 2 network between hosts of different domains. TincVPN is automatically installed and configured on every Cloudy node, ready to be activated (for privacy reasons, this option is left to the user’s choice). After its activation, a VPN daemon is started in order to reach other Cloudy instances via a layer 2 network. Thus Avahi can communicate transparently with other nodes.

2.5.11. Experimentation

**Scenario UPC**

In this first scenario, we connect a node from UPC, called best node, that we previously measured with the best round-trip time to the streaming publisher (PeerStreamer source service), and the rest of the nodes to this node, in order for us to manage our own network topology. With our topology in mind, we gather statistical data about the current state of each node and how these nodes would manage their data (chunk) exchange. Three experiments were conducted with different time frames of 30 minutes, 1 hour and 12 hours of continuous running of the PeerStreamer service, respectively. This was done in order to gather information about the chunks received and/or played-out by each node, and also to analyse its progression within different time frames. These chunks contain the data used to watch the actual streaming of the camera images, and therefore gathering this kind of information is essential to know how PeerStreamer is handling the network and image quality.

**Scenario ALL**

For the second scenario, we use three groups of nodes comprising UPC, Guifi.net, and AWMN. We select one node of each group, the one which has the best RTT to the PeerStreamer source service. These nodes connect to the PeerStreamer service while the rest connect to the group’s best node, respecting the group each node belongs to. This way, we can control the topology network, which allows us to analyze in more detail the node behaviours of each group. Having nodes located in different geographical areas of the community networks enriches our experiments, since we can better understand the effect of distance on the degradation of the image quality. For a better comparison with the first scenario, the three experiments are also executed within a time frame of 30 minutes, 1 hour and 12 hours of continuous live streaming, respectively, with an addition of a 6 hours experiment.

**Scenario RTMP**

The main purpose of this experiment was to determine whether our platform supports or not the streaming of rtmp streams. The scenario was basically based on one node with a good connection and relative good resources and the whole Guifi network. The stream we selected was the continuous...
stream of Guifi-TV. The first mentioned node was responsible of getting the rtmp stream and republish it in Guifi through the peerstreamer infrastructure so other nodes could watch the stream using Cloudy. Since the main goal was to check if Cloudy could be used in this field as a user friendly video streamer, the only thing we wanted to check was if there was a constant video and audio flow with a rather low package loss. Besides the Guifi network, there were also some nodes from UPC watching the streaming.

**Scenario RTP instead of UDP**

Recently the Peerstreamer development team released a new version of Peerstreamer which uses RTP instead of UDP for its internal communication. RTP provides a more secure communication between the peers since it has control messages besides the data packages.

### 2.5.12. Results

#### For UPC and ALL Scenarios

The following figure 2.16 depicts a comparison for each node, between all the chunks the node received and the chunks that were on time to be displayed and not duplicated, which gives us the number of chunks that were discarded. However, all the nodes do have mostly the same number in each experiment, meaning that there were very few extra chunks flooding the network, within each time frame. We can also see that, each node received a different amount of chunks, within the same experiment. The reason is that each node is unique, and therefore performs the tasks of receiving and sending chunks differently from each other.

![Figure 2.16: Results for UPC and ALL scenarios.](image)

Fig. 2.17 depicts the average chunks received for each group of nodes within each time frame (for 30 minutes, 1 hour and 12 hours). We can see that each group receives a different amount of chunks on average. For each group the amount of discarded chunks on average is very low. It means that all chunks received were viable to be played out, and thus the network was not flooded with extra chunks, even when we have distant groups. In Fig 2.18 it can be seen that on average each group has a receive chunk rate of mostly the same ratio. Also, the time frame of each stream seems to affect the number of received chunks with longer time frames. This can happen because of the constant change of the network behaviour. While some of the nodes can loose temporally their connection to the network, others may became unavailable for short periods of time. Therefore, we see that in the additional 6 hours experiment, the chunk rate is higher on average, and on the 12 hours it averages along the 1 hour and the 6 hours results.

#### For RTMP Scenario

During the first 20 minutes, the stream was running pretty good: the image flow was constant (with some package loss, but almost null) and the audio was constant; but at a certain time a node from Guifi which was a little far away from the UPC joined the stream, and afterwards it got stuck.
2. Consolidated pilots

2.5. Peerstreamer video streamer

Figure 2.17: Comparison between averaged Chunks received and Chunks on time and not duplicated (ALL)

Figure 2.18: Average Chunk rate received with each time frame (ALL)

this happened, the stream was reset, but since the previous instance ended under unexpected circumstances, the ports it was using were blocked and they could not be used again. So finally the new stream was bound to different ports. To see what was happening, all the process were invoked manually from the node, being these a vlc and a Peerstreamer instance in one peer (source), a Peerstreamer instance with UDP output in another peer, and finally a machine which watched the UDP final video using vlc from the command line. The results when the first source was an RTSP stream were clean and good, there was not any unusual error. However, when the original stream is RTMP, a huge bunch of errors from the final machine’s VLC instance could be appreciated. From there, the hypothesis that errors are carried out when transcoding the RTMP errors came up.

2.5.13. PeerStreamer Live Streaming at Guifi.net event

We used PeerStreamer to stream live the Guifi.net event in Barcelona on February 2015. The duration of the event was two hours. PeerStreamer service of Cloudy was used in order to stream the live event. The source node was located in the UPC office E101, and there were 5-7 peers located in different places in Barcelona watching the stream. Figure 2.19 depicts a screenshot from the live event taken from one of the peers.

2.5.14. Conclusions

A heterogeneous distributed community cloud was presented as environment for the experimental evaluation of PeerStreamer, a P2P live streaming platform. The Cloudy distribution for community network clouds was presented, which integrates PeerStreamer and provides to the end user additional functionalities to join easily the streaming system.

Deliverable D4.5
2.6. Decentralized video sharing

2.5.15. References


2.6. Decentralized video sharing

A key service for any community of users is the possibility to easily share and search for content, be it videos, photos, news, blogs, forums. In order to provide such a service in a decentralized p2p manner, we identified three necessary services:

- a decentralized full text search service - Sweep
- a p2p content sharing service - GVoD (video sharing)
- a bootstrapping service - CaracalDB

2.6.1. Sweep

Full-text search is the de-facto standard for finding information on the Internet. From Google to Piratebay, people are used to writing words in natural language, pressing return, and finding what they are looking for. Currently, there is no decentralised, full-text search service available on the Internet or in wireless community networks.

Sweep is a decentralised search system, where clients can, reliably and with low latency, search for resources in the system.
2.6.2. GVoD

Video on Demand (VoD) is a digital service that enables users to find and watch video content on-demand, at any time of their convenience. Peer-to-Peer (P2P) technology has been used to build VoD systems, by enabling nodes to collaborate in the downloading of video files, helping the system to scale in order to handle a large number of concurrent downloaders.

GVoD is a VoD service that has been adapted for CLOMMUNITY, based on previous work on video on-demand carried out at SICS.

2.6.3. CaracalDB

A key-value store is a simple, but scalable way of storing unstructured, relatively small (1kB or less) data values identified by a globally unique key. Common usage scenarios include storage of preferences or configurations, storage of third-party service meta-data, temporary data or time-series data (i.e. logs). In addition to these general features our system, CaracalDB has support for queries over data-ranges, secondary indexes and ORM mapping (with JPA). CaracalDB is a stand-alone server application written in Java, but provides a REST API and a simple Web-interface to allow different modes of interaction.

2.6.4. Partners

- KTH - Royal Institute of Technology
- SICS - The Swedish Institute of Computer Science

2.6.5. State

The system as a whole has a stable release that is tested and contains the core features.

- Sweep core features: searching and adding new content entries, with scalability support through sharing and replication of shards on multiple peers.
- GVoD core features: creation and management of overlays to allow easy discovery of content providing peers; streaming content to the video player as the content is being downloaded as well as adjusting download position based on playing position.
- CaracalDB core features: storing and retrieving data through an extended key-value api, including powerful range queries. CaracalDB also provides flexible storage options: in-memory or on disk(levelDB), as well as load balancing.

All three components (Sweep, GVoD and CaracalDB) are still under development with new features under design or development.

An AngularJS web-ui provides access to the system and allows searching and sharing videos.

2.6.6. Goals

Pilot specific goals:

- integration of the three separate services
- evaluate the system’s properties in relation to wireless community networks
- fix non desired behaviour
Contribution to Clommunity project:

- provide search and content sharing service to Clommunity users. As a first case the content is restricted to videos in mp4 format.
- integration of services in the Cloudy distribution

2.6.7. State of the art

There are many successful video on demand systems available on the Internet, such as Netflix and YouTube. What they have in common is that they are typically delivered by content distribution networks, with multilevel caching infrastructures that span the globe.

Although there are some P2P-enabled VoD systems, such as Popcorn, there is no dominant, widely deployed P2P-VoD platform for the Internet. The main reasons for this are both technical and legal.

Technically, many users on the Internet have asymmetric upload and download bandwidths and in a P2P network, the aggregate download bandwidth is equal to the bandwidth provided by all nodes in the system. Since upload speeds are lower, on average, than download speeds for nodes in P2P networks on the Internet, a centralised service, typically backed by a content-delivery network (CDN), can provide better quality videos using its higher available bandwidth. Examples of such services are Netflix and YouTube.

Asymmetric network bandwidths are not the only factor reducing available upload bandwidth. The high proportion of nodes behind NATs (between 80-90% of all nodes) and the low success rate for NAT hole-punching protocols (that establish a direct connection to a node behind a NAT) has meant that existing P2P systems only provide a small fraction of their potential available upload bandwidth.

In addition to the technical challenges, there are also legal impediments to P2P-VoD systems. Existing P2P-VoD systems typically lack copyright control, leading to the widespread illegal distribution of copyrighted material. There is also a large number of commercial and open source database systems available today, among them Cassandra, MongoDB, CouchDB, FoundationDB and Riak. These systems provide a wide range of capabilities and applications, many of which CaracalDB will be able offer as well when it is finished, while allowing for flexibility to pick what is best for the service’s application.

2.6.8. Workplan

Project tasks included:

- integrating the search engine (Sweep) and the video downloading service (GVoD) (Done)
- building a web-ui for the system (Done)
- integrating CaracalDB for speeding up system response (system bootstrap, video overlay bootstrap) (Done)
- integrating the system into Cloudy (Done)
- preliminary experiments on the Clommunity infrastructure (Done)

2.6.9. Progress

All three services: CaracalDB, GVoD and Sweep are integrated into Cloudy and have been deployed and tested on Clommunity infrastructure.
2.6.10. Experimentation

The system contains three different services: Sweep, GVoD and CaracalDB that can run on the same machine or on different machines. Simulations show that each of the systems perform correctly, but a small deployment was required to verify through a small preliminary experimentation if the deployed system behaves as expected.

2.6.11. Experimental setup

Two different experimental setups are used:
- 10 Clommunity nodes, with 1GB RAM, Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz.
- In the first setup, all nodes are within the same cluster and the latency between nodes is 20ms.
- In the second setup, we have two clusters of 7 nodes. Within the same cluster we have 20ms latency, and 330ms latency between clusters.

2.6.12. View video scenario

In both setups, we have 3 nodes running CaracalDB and 7 nodes running Sweep, GVoD and the application web-ui. The scenario is simple:
- upload video
- search for video through Sweep
- download and view video
- measure playback latency and playback continuity

The scenario is run in two configurations:
1. all GVoD and Sweep nodes are in the same cluster
2. the GVoD and Sweep nodes are split in two clusters of 3 and 4

2.6.13. Results

In both cases there is no interruption in the playback of the video.

The measured playback latency is:
1. 2.8s with 0.3 standard deviation.
2. 3s with 0.25 standard deviation
3. Other pilot deployments

The pilots gathered in this chapter neither involve end-user nor are ready to do so. Nonetheless they are included in the deliverable because they include valuable information for future developments.

3.1. Infrastructure as a service (IaaS) for Community Clouds

3.1.1. Introduction

Virtualization is the main enabling technology for cloud computing. Therefore, for the development of a cloud environment in the context of CNs, provision of Infrastructure as a Services (IaaSs) in the form of virtual machines (VMs) is a key aspect to take into account.

In the CLOMMUNITY project, IaaS in the CNs cloud has been proposed to be renamed as Infrastructure Services for Community Clouds (ISCC), to stress the relevance of proximity in the cloud infrastructure being deployed inside the CN premises rather than on the Internet cloud.

This pilot aims at studying and deploying a web-based ISCC management platform integrated in Cloudy to facilitate users deploying VMs in their CN nodes. This way, they will be able not only to rapidly launch machines with any kind of service pre-installed (a blog, a file repository, a LAMP server, etc.), but also to share the hardware resources with other users, letting them deploy their own VMs in a neighbouring node.

3.1.2. Partners

Guifi.net

3.1.3. State

The development phase and the integration with Cloudy have been finished, but no experimentation with real CN users has been conducted.

The tools integrated in Cloudy’s web interface have been made available for the CN users.

3.1.4. Goals

The primary goals of the pilot are to integrate virtualization capabilities in Cloudy and evaluate their utilization within CN environment and its users.

The pilot-specific goals are to:

- Analyse the specifications and capabilities of available open-source virtualization technologies and their web-based management interfaces
- Integrate VMs provision with Cloudy’s web interface
- Provide CN users an experimental testbed for deploying VMs to bootstrap CN services

With regard the contribution to the CLOMMUNITY project, the goals are to:
3. Other pilot deployments

3.1. Infrastructure as a service (IaaS) for Community Clouds

- Contribute to the IaaS provision in the context of Community Clouds
- Analyse the pros [and cons] of empowering users with easy, click-based interfaces to deploy VMs

3.1.5. Work plan

- Available open-source virtualization technologies analysis ✔
- Web-based virtualization management interface tests ✔
- VMs provision integration with Cloudy’s web interface ✔
- Virtualization testbed provision ✗
- VMs deployment user empowerment analysis ✗

3.1.6. Available virtualization technologies

The following are the virtualization technologies that were analysed and considered for integration in Cloudy for ISCC provision.

3.1.6.1. OpenVZ

OpenVZ (Open Virtuozzo)\(^1\) is an operating system-level virtualization technology based on the Linux kernel and operating system. OpenVZ allows a physical server to run multiple isolated operating system instances, called containers, virtual private servers (VPSs), or virtual environments (VEs).

OpenVZ uses a single patched Linux kernel and therefore can run only Linux. All OpenVZ containers share the same architecture and kernel version. This can be a disadvantage in situations where guests require different kernel versions than that of the host. However, as it does not have the overhead of a true hypervisor, it is very fast and efficient.

Memory allocation with OpenVZ is soft in that memory not used in one virtual environment can be used by others or for disk caching. While old versions of OpenVZ used a common file system (where each virtual environment is just a directory of files that is isolated using chroot), current versions of OpenVZ allow each container to have its own file system. OpenVZ is one of the virtualization technologies available in Proxmox VE.

3.1.6.2. KVM

KVM (Kernel-based Virtual Machine)\(^2\) is a virtualization infrastructure for the Linux kernel that turns it into a hypervisor. KVM requires a processor with hardware virtualization extension.

A wide variety of guest operating systems work with KVM, including many flavours and versions of Linux, BSD, Solaris, Windows, Haiku, ReactOS, Plan 9, AROS Research Operating System and OS X. In addition, Android 2.2,

\(^1\)OpenVZ: [http://openvz.org](http://openvz.org)
\(^2\)KVM: [http://www.linux-kvm.org/page/Main_Page](http://www.linux-kvm.org/page/Main_Page)
3.1. Infrastructure as a service (IaaS) for Community Clouds

3. Other pilot deployments

GNU/Hurd, Minix 3.1.2a, Solaris 10 U3 and Darwin 8.0.1, together with other operating systems and some newer versions of these listed, are known to work with certain limitations.

3.1.6.3. LXC

LXC (Linux Containers)\(^3\) is an operating-system-level virtualization environment for running multiple isolated Linux systems (containers) on a single Linux host machine. LXC provides operating system-level virtualization through a virtual environment that has its own process and network space, instead of creating a full-fledged virtual machine.

LXC relies on the Linux kernel cgroups functionality that allows limitation and prioritization of resources (CPU, memory, block I/O, network, etc.) without the need for starting any virtual machines, and namespace isolation functionality that allows complete isolation of an applications’ view of the operating environment, including process trees, networking, user IDs and mounted file systems.

LXC is similar to other OS-level virtualization technologies on Linux such as the earlier OpenVZ.

3.1.6.4. VirtualBox

Oracle VM VirtualBox (formerly Sun VirtualBox)\(^4\) is a hypervisor for x86 computers from Oracle Corporation. VirtualBox may be installed on an existing host operating system; it can create and manage guest virtual machines, each with a guest operating system and its own virtual environment. Supported host operating systems include Linux, OS X, Windows XP and later, Solaris, and OpenSolaris. Supported guest operating systems include versions and derivations of Windows, Linux, BSD, OS/2, Solaris, Haiku and others.

Guest Additions should be installed on the guest operating system in order to achieve the best possible experience. It consist of device drivers and system applications that optimize the guest operating system for better performance and usability.

3.1.6.5. Analysis

After testing the different virtualization technologies mentioned above, the following considerations have been reached:

- **OpenVZ** seems to be the a good option for low-end machines (e.g. Intel Atom CPU-based embedded devices), since its lack of a hypervisor makes it very fast and efficient. It needs, however, to replace the running kernel with the OpenVZ kernel, which is based on an old version of the Linux kernel. This might pose incompatibility issues with modern hardware.

- **KVM** is a full virtualization technology that allows running almost every kind of guest OS, it is mature and well supported in the running Linux kernel. Since it requires CPU virtualization extensions and the host acts as a hypervisor, it requires more powerful hardware, making it unsuitable for certain low-end machines.

\(^3\)LXC: https://linuxcontainers.org/
\(^4\)VirtualBox: https://www.virtualbox.org/
3. Other pilot deployments

3.1. Infrastructure as a service (IaaS) for Community Clouds

- **LXC** is one of the main cloud-enabling technology nowadays. It relies on the Linux kernel functionalities and, since it provides system-level virtualization instead of a full virtual machine, it is very fast and efficient. It is a very good option for low-end machines, but also for devices with more resources.

- **VirtualBox** is a very popular hypervisor-like virtualization technology that is more oriented for the graphical desktop interface usage, for which it provides features like drag’n’drop, copy&paste integration, etc. that are not completely useful in the CN cloud environment.

### 3.1.7. Web-based virtualization management

The following are the web-based virtualization management platforms analysed and considered for integration in Cloudy for ISCC provision.

#### 3.1.7.1. OpenVZ Web Panel (OWP)

OpenVZ web panel screenshot

OpenVZ Web Panel is a GUI web-based frontend for controlling of the hardware and virtual servers with the OpenVZ virtualization technology. It provides an automatic installer for replacing the running Linux kernel, ability to backup, restore and clone containers, network addressing management, multi-user/roles/language support, etc.

![Figure 3.1: Open Web Panel interface screenshot.](image)

#### 3.1.7.2. LXC Web Panel

LXC web panel screenshot

LXC Web Panel is an easy-to-install and user-friendly web panel for LXC on Ubuntu. It allows managing containers (CPU, memory, network) from a simple web interface based on Flatstrap/Bootstrap, making the process of deploying VMs a matter of a few clicks.

#### 3.1.7.3. phpVirtualBox

phpVirtualBox screenshot

phpVirtualBox is a web-based front-end to VirtualBox written in PHP. It is an open source, AJAX implementation of the VirtualBox user interface written in PHP. It allows accessing and controlling remote VirtualBox instances in a headless environment, mimicking the VirtualBox GUI on a web interface.
3.1.7.4. Kimchi

Kimchi is a web management tool to manage Kernel-based Virtual Machine (KVM) infrastructure. It is developed with HTML5 to intuitively manage KVM guests, create storage pools, manage network interfaces (bridges, vlans, NAT), and perform other related tasks. It is designed to make it as easy as possible to get started with KVM and create the first guest. Kimchi runs as a daemon on the hypervisor host and manages KVM guests through libvirt.

Figure 3.2: LXC Web Panel interface screenshot.

Figure 3.3: phpVirutalBox interface screenshot.

Figure 3.4: Kimchi interface screenshot.
3.1.7.5. Analysis

After testing the different web-based virtualization management platforms, the following results have been obtained:

- **OpenVZ Web Panel** can be integrated with relative simplicity into a Debian 7.x Wheezy-based Cloudy, but it has a dependency on a somehow outdated Linux kernel that may not support newest hardware. It allows to easily obtain a large number of well known and up-to-date guest templates like bare OS images or TurnKey Linux application-specific images.

- **LXC Web Panel** is a very promising tool but the fact that it does not work on Debian makes it unsuitable for Cloudy. Efforts could be dedicated to make it Debian-compatible in the future.

- **phpVirtualBox** can be easily integrated into Cloudy, but VirtualBox does not appear to be the reference virtualization technology for Community Clouds.

- **Kimchi** can be integrated with relative simplicity into a Debian 8.x Jessie-based Cloudy, but a number of additional packages need to be installed. It is really intuitive to manage and it allows running virtually every x86-targeted operating system available as an ISO image. Kimchi is the newest project among these four, it is in active development and - up to an extent - supported by IBM.

3.1.8. Virtual machines provision integration in Cloudy

This section reports the integration efforts of virtualization technologies in Cloudy.

3.1.8.1. OpenVZ Web Panel

OpenVZ Web Panel has been integrated in Cloudy, by adding a plug in to the web interface.

If the host device is running on a compatible hardware architecture (x86), the OpenVZ Web Panel plugin is shown in Cloudy’s web interface. The user can start the auto-installation process in a couple of clicks and, after a few minutes, get an OpenVZ-enabled Cloudy instance.

![Figure 3.5: OpenVZ Web Panel in Cloudy’s web interface](image)

Using the web interface in Cloudy, the user can announce the VMs service to the CN, facilitating the share of resources between users of the network.

The OpenVZ Web Panel provides full virtualization support to CN users, letting them to deploy different readily available images for popular operating systems and customised applications like almost any version of Debian, Ubuntu, Fedora, SuSe, etc.
3.1. Infrastructure as a service (IaaS) for Community Clouds

3.1.8.2. Kimchi

Kimchi integration in Cloudy has also started and it is expected to be released in the next future.
3.2. MODAClouds Decisions Support System

With large number of cloud service providers in the community network clouds, it is necessary to support service selection mechanisms for ensuring quality of experience. We need to support mechanisms that provide assistance in cloud service selection while taking into account different aspects pertaining to associated risks in community clouds, quality concerns of the users and cost limitations specifically in multi-clouds ecosystems. This work proposes a risk-cost-quality based decision support system [1] to assist the community cloud users to select the most appropriate cloud services meeting their needs. The proposed framework not only increases the ease of adoption of community clouds by providing assistance to users in cloud service selection, but also provides insights into the improvement of community clouds based on user behaviour.

3.2.1. Introduction

The successful functioning of the community cloud relies on the active participation of the community members that in turn is highly dependent on the level of satisfaction experienced by the users of the community cloud services. The user experience can be maximised if the offered cloud service accurately matches the user requirements. A decision support system (DSS) is therefore required that can recommend the users to select the accurate cloud services from the services offered by the network satisfying their requirements. Such DSS is particularly crucial in multi-cloud ecosystem where cloud service selection becomes exceedingly difficult for users. Another crucial challenge in cloud service selection is translating the user requirements from a highly pragmatic perspective into technical properties that should be possessed by the cloud service. The DSS should take into account all these challenges and provide the user a recommendation system to assist the cloud service selection.

DSS is based on the solution to a multi-criteria decision making problem (MCDM), and there are many ways to solve this like analytical hierarchy process, condition based optimization, etc [2]. In multi-cloud environments, the DSS should take into account multiple factors that are specific to the characteristics of such ecosystems. For instance, DSS adopting three dimensional approach of satisfying risk, cost and quality based aspects in such multi-cloud environments are particularly effective. In terms of cloud service selection, a novel DSS proposed in [3] is based on such approach. This DSS implements a risk-driven methodology to translate the user requirements into technical characteristics of the cloud services, and recommends the most accurate service selection that minimises the risk and maximises the quality at an appropriate cost.

3.2.1.1. Key Characteristics

Our proposed DSS is based on a number of characteristics that allows it to be agile enough to accommodate the crucial requirements derived from the problem of service selection in community clouds. We now describe these key characteristics. Firstly, the DSS framework allows the specification of the pragmatic needs of the users from different perspectives by allowing them to specify the assets that they intend to protect. These assets could be business oriented or technical oriented assets. Hence, the community cloud users from varied background can specify their main concerns with the use of the cloud services. Secondly, in the next step, the DSS using a pre-defined background mapping produces the list of risks associated with the assets and provides relevant treatments for these risks. The users are allowed to choose the risks and treatments. The set of treatments eventually form the necessary technical and non-technical features that the desired cloud services should have. Hence
the proposed DSS provides a risk-analysis based methodology to translate user needs into desired characteristics of cloud services. Thirdly, each cloud service is scored from 1 to 10, inspired from SMI scores, in order to estimate the desired level of capabilities it possesses to mitigate a particular risk. These scores are assigned by careful analysis and are heuristically improved with repeated use of DSS. Fourthly, gathering the data from the service providers forms an integral part of the proposed framework and not a separate module that allows advanced implementation in complex environments. Fifthly, the multi-cloud environment is particularly accommodated in this framework by allowing the prevention of specific risks associated with the multi-cloud systems in community clouds. For example, the problem of obtaining the services from the same provider in community cloud, translated otherwise as vendor lock-in, is tackled by discarding such recommendations. The ease of migration from one provider to another is also considered while providing a final score and ranking to each of the recommendations provided by the DSS.

3.2.1.2. Implementation of DSS

We have developed a prototype DSS for supporting service selection in collaborative manner for multi-cloud applications. The prototype uses distributed graph database ArangoDB\(^5\) as a persistent layer data store with graph exploration capabilities and AngularJS\(^6\) based front-end which provides wizard based approach to identify best match of selection criteria. The prototype supports saving and sharing sessions capability in order to allow participation from multiple users in the definition of the requirements. In Figure 3.9, we have shown the generic MODAClouds DSS for cloud service selection that provides recommendations of the cloud providers as proposed in [3]. The final screen provides recommendation in multi-cloud environment for the possible cloud service providers matching the requirements of the user along with a score indicating how close a recommendation is to the desired properties provided by the user. The score is calculated based on what percentage of risks that were specified by the users have been mitigated.

3.2.1.3. Service Selection with DSS

For evaluating the use cases for community cloud services, we need to identify both the low-level and high-level metrics relevant for the community cloud, and take into account the data provided by the service providers in the community cloud testbed. Low-level data can be collected from the monitoring systems in testbed, and for high-level data we can survey the providers and users of the services. Some useful metrics particular to the collaborative nature of community clouds include providers’ reputation among the users, users’ as well as providers’ geographical location, and mutual trust and standing among the providers and users based on the social status and cumulative history of past transactions. Considering the different applications already deployed in community cloud, we plan to collect relevant data from service providers.

3.2.2. Partner(s)

- Guifi.net
- Universitat Politècnica de Catalunya BarcelonaTech

\(^5\)http://www.arangodb.com
\(^6\)http://angularjs.org
3.2. MODAClouds Decisions Support System

3.2.3. State

We have deployed a prototype from the MODAClouds project for supporting service selection in collaborative multi-cloud settings of community network clouds. The prototype takes as input from the user different requirements about the service, and matches the specified criteria to available cloud services.

3.2.4. Goals

The goals of this work are the following:

- Presenting the use cases of storage applications in collaborative multi-cloud environment.
- Extending the idea of DSS to collaborative multi-cloud environments, which differ from public and enterprise cloud systems because consumers can also act as providers.
- Identifying metrics specific to collaborative multi-clouds, which are normally not considered in traditional cloud systems.

3.2.5. Work Plan

- The initial step is to get the first prototype of the DSS system, and deploy in Clommunity testbed.
- Subsequently, we need to collect data from Clommunity use case from the testbed for configuring and customising the updated DSS system.

---

Figure 3.9: Prototype DSS for supporting service selection in collaborative multi-clouds

- CA Labs

7http://www.modaclouds.eu/
3.2. MODAClouds Decisions Support System

- Taking the Clommunity use case into account, CA Labs will update the DSS system.
- The updated second prototype has to be installed in Clommunity testbed.

3.2.6. Progress

- The first prototype of the DSS system has been deployed in the Clommunity testbed in M20.
- The updated prototype of the DSS system, customised for the Clommunity use case, is to be deployed in M30.

3.2.7. Evaluation metrics

We evaluate the use cases that reflect how the collaborative cloud services can be used in a multi-cloud environment, and how DSS can assist different actors – developers, providers, and end-users – in service selection, as consumer or prosumers. We focus mainly on storage service, where factors like vendor lock-in and migration costs can be considered when comparing different providers in the community cloud.

3.2.8. Dependencies

The work is in collaboration with and dependant on the DSS system being developed at CA Labs as part of MODAClouds European FP-7 project.

3.2.9. Conclusion

We have deployed useful applications on community cloud infrastructures in the Guifi.net community network to assess the applications’ performance, and have set up a prototype DSS in Clommunity testbed\(^8\), that helps in selecting the appropriate services given the criteria specified by the users of these community cloud services. Such a DSS that can facilitate selection between service providers with very different resource profiles depending upon given criteria can be very useful for the users of community clouds.

We observe that the risk-analysis based DSS framework holds a high potential to meet the requirements of the community cloud users. By assisting the users to make cloud service selection, the DSS not only increases the potential to attract more active users, but also provides efficient usage of the cloud services. The agility of DSS framework makes it possible to tailor it to the community cloud, and we envision the adaptation of the DSS to more specific needs of the community cloud users in future.

---

\(^8\)[http://dss.clommunity-project.eu/](http://dss.clommunity-project.eu/)
4. Evaluation

4.1. Introduction

We apply in the following sections the metrics and criteria proposed in D4.1 to evaluate the performance objectives integrating the results of the different cases pilots. We follow the order of the use cases as to D4.1. It must be noticed that the results are limited to the cases we are aware of either by DADS, in the case of those clouds we have access, or by indirect means like comments on mailing-lists, private e-mails, etc.

4.2. IaaS for Community Clouds

In the IaaS domain, two cases were identified: 1) Virtual machines for development and testing, and to attend special requirements, and, 2) tailored software distributions that are effective to reduce the level of skills and the learning curve needed to install and maintain software applications.

The case of VM provision for community clouds was part of almost any of the experiments of the pilots. Proxmox has been used for the management of VMs in the cluster formed by desktop machines. For low-capacity machines, containers were created, mainly by using the Confine controller.

The Cloudy distribution has been involved in many pilots and experiments. Thus, it has been indirectly tested to great extend. Major pilots and experiments which used the VMs and Cloudy were DADS, IoT, Syncthing, Tahoe, Peerstreamer, Video, ModaClouds.

Table 4.1 indicates the results on IaaS for community clouds. We observe that the current Cloudy hosts are a mix of project-contributed hosts and third-party hosts. This fits the expected evolution, since the project contributed start the deployment of the community cloud and now we are in a transition towards the community up-take. The short-term perspective is that Cloudy will gradually replace the software used for the host in Guifi for network management, while being the first choice for installation in all new hosts that are added. Thus, we expect an increase of Cloudy distributions over time, strongly correlated to the growth of the network.

4.3. NaaS for Community Clouds

Contrary to the standard commercial cloud solutions, in CNs uncoordinated contributions resulting in sub-optimal placement of nodes and services must be expected. Moreover, connection to all contributed nodes cannot be assumed. Thus, node discovery strategies based on layer two are not suitable. The DADS pilot studied and validated solution based on Serf which resolves these challenges satisfactorily with an insignificant overhead. In addition, Serf allows to gossip additional content among the nodes. This feature is used to exchange additional node information for higher-level services, e.g. for service selection. Examples for such service is the ModaClouds DSS, but also the current ”scan service” information obtained through Serf.
4.4. PaaS

Table 4.1: Metrics and evaluation of IaaS cases.

<table>
<thead>
<tr>
<th>Case</th>
<th>Metric</th>
<th>Eval. crit.</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM</td>
<td>No. of CLOMMUNITY-project contributed hosts</td>
<td>5/10/100</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>No. of third-party contributed hosts</td>
<td>5/10/100</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>No. of third-parties participating</td>
<td>3/5/10</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>No. of experiments</td>
<td>5/10/20</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>No. of VMs per experiment</td>
<td>–</td>
<td>up to 15</td>
</tr>
<tr>
<td>Distro</td>
<td>No. of installations</td>
<td>10/25/80</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Installation and management GUI</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>UX (Satisfaction survey)</td>
<td>Positive</td>
<td>WiP (July 2015)</td>
</tr>
</tbody>
</table>

Table 4.2: Metrics and evaluation of NaaS cases.

<table>
<thead>
<tr>
<th>Case</th>
<th>Metric</th>
<th>Eval. crit.</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2-L3</td>
<td>Self-discovery</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Restricted to management traffic</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Average Mbps of traffic of middle-size cloud server</td>
<td>0.5/1</td>
<td>Insignificant for Serf</td>
</tr>
</tbody>
</table>

4.4. PaaS for Community Clouds

Platform services provide basic services and resources to enable the end-user software services on top. Platform services are aimed at offering transparent and uniform procedures to the software services layer, to avoid duplication of functions and simplification of common mechanisms. Service monitoring is useful to verify the correct operation of the applications and, when needed, perform any required actions. Service monitoring is part of the service discovery and announcement service, materialized by Serf.

Table 4.3 indicates the results on PaaS for community clouds. The monitoring service dynamically informs about the status of each service in the community cloud (which has been declared appropriately with the owner’s consent). Serf allows to exchange additional information, which eases to develop extensions to support additional functionalities. User authentication has been integrated and is applied, for instance, when the user integrates through the Cloudy GUI its node in the Guifi Web. Service announcement is operational and service discovery too.

The results of these pilot are successful and they are relevant since the studied PaaS provide basic services to keep a decentralized community cloud operational.
4. Evaluation

4.5. SaaS

<table>
<thead>
<tr>
<th>Case</th>
<th>Metric</th>
<th>Eval. crit. [min/normal/outstand.]</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Monitoring</td>
<td>Interfaces Traffic</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Nodes RTT</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Amount of supported services</td>
<td>3/5/10</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>Modular design (to allow future additions)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>User Authentication</td>
<td>Autonomous service</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>No. of services using it</td>
<td>3/5/10</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Average queries (daily)</td>
<td>0.1/1/10k</td>
<td>~5k</td>
</tr>
<tr>
<td></td>
<td>Basic credentials</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Free-Schema</td>
<td>Desirable</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>HA</td>
<td>1/1/2</td>
<td>4 (2x2)</td>
</tr>
<tr>
<td>Service Announcement</td>
<td>Service implemented</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Modular design</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>No. of services using it</td>
<td>3/5/10</td>
<td>7</td>
</tr>
<tr>
<td>Service Discovery</td>
<td>Recognition of announced services</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Publication of discovered services</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DDBB</td>
<td>No. of services using PaaS DDBB</td>
<td>1/2/5</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.3: Metrics and evaluation of PaaS cases.

4.5. SaaS for Community Clouds

Differently to the rest of the sections, we evaluate the up-take of SaaS qualitatively, because during the development of the solutions and the execution of the pilots, we realised that the many of the metrics identified in D4.1 are impossible to quantify (e.g. there is no way to known the total space available in syncthing) and others are not suitable (e.g. how to identify new consumers of video on demand if there is no registration process?).

**Data storage** Two services have been put in place and are integrated in Cloudy: Tahoe-LAFS, for highly secure storage, and syncthing, for massive storage. In addition, a personalised and secured application has been developed on top of syncthing to test a business case. Massive storage, the “commercial” option included, has become popular rather fast. Contrary, the usage of Tahoe-LAFS is almost reduced to the experiments performed as part of the pilots. We think that the ease of use in the first case and the difficulty of the second, despite the WEB interface developed on purpose, are the main reasons to explain these results.

**Video on demand** GV oD has just been integrated and there has not been time to involve community users. Nonetheless, we expect a fast up-take because it is a very demanded service.
**Video streaming** Mainly thought to stream community events, during the pilot one of these events was already streamed and was very welcome, both by the audience as well as by the technical team in charge of the recording and broadcasting. Thus we expect a fast appropriation by the community.

### 4.6. Non-Technical

This section considers all non-technical related issues, specially the social ones. Active involvement of the users was the key challenge. Table 4.4 summarises the results. We can observe that a large number of activities were conducted to inform users and to address particular groups for participation. These activities were taken-up by the community members. The tools put into place by the consortium to support users are expected to be permanent, going beyond the end of the project. Openness and transparency of the developed system have been taken care of. The developed software is FLOSS and tools have been put in place such that development continues by the community.

<table>
<thead>
<tr>
<th>Case</th>
<th>Metric</th>
<th>Eval. crit.</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Live streaming [min /normal/outstand.]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Engagement</td>
<td>Presentations and workshops in CNs forums (total)</td>
<td>5/10/20</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Presentations and workshops in academic and industrial forums (total)</td>
<td>5/10/20</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Support to Guifi.net adopters</td>
<td>10/20/-</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Support to third-party adopters</td>
<td>5/10/-</td>
<td>2</td>
</tr>
<tr>
<td>Awareness</td>
<td>Presentations and workshops in CNs forums (total)</td>
<td>5/10/20</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>No. of secure applications integrated</td>
<td>2/3/5</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>% of FLOSS applications integrated</td>
<td>90/95/100</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 4.4: Metrics and evaluation of non-technical cases.
5. Conclusions

This deliverable reported the pilots that were deployed in the community network cloud. These pilots refer to services used by end-user, which were run in order to assess them for usage in community network clouds.

As result of the pilot studies we can say that overall, end-user services can be run with suitable performance in the community network clouds. The distributed announcement and discovery service is a basic services needed in community network clouds. Other services relay on it to be discovered. Tahoe-LAFS showed to perform correctly and fulfilled the feature of fault-tolerance. Peerstreamer was tested in several experiments including for live transmission of a Guifi meeting over the community network infrastructure. GVod video-on-demand service was tested with existing videos from the Guifi TV channel. An IoT proof-of-concept deployment was run during several months demonstrating the capability of community networks to be used for such types of services. Third-party services such as the decision support system were deployed to explore tools that assist users in service selection. Finally, the Syncthing service was tested and the results have encouraged an SME to further explore a commercial exploitation of this service.

The results overall confirm that services offered through Cloudy in the community network cloud in Guifi.net are stable and a performance is obtained, which is suitable to encourages users to join the cloud. The numerical metrics applied in the evaluation of the services support quantitatively these results.
A. Pilots template

Introduction

Abstract of the pilot. This is a template for the Clommunity project pilots. All pilots must cover the following sections. You can use this one as an example: Distributed announcement and discovery. Nice pictures, like Figure A.1 by Naddsy, always help.

![Search service scenario.](image)

Figure A.1: Search service scenario.

Partners

The partner or partners of the consortium participating.

State

A brief description of the pilot-related work, the status, timing, etc. For instance:

- Status: not started / work in progress / finished, etc.
- Observations: short description of the requirements or any needs, etc.
- …

Goals

A list, with a short description, of the goals of the pilot. They can be classified in these non-exhaustive categories:

- Pilot-specific goals
- Contribution to Clommunity

Work plan

Breakdown of tasks, timing, dependencies among them, milestones, etc. (Ganttt chart)

Risk analysis and contingency plan.
A. Pilots template

Progress
Progress according to the work plan.

State of the art
A text describing the state of the art of the pilot (similar technologies, related experiments, etc.)

Experiments design
A global description of how the experiments will be, the resources needed to perform then, etc.

Evaluation metrics
The criteria to evaluate the goals of the pilot.

Dependencies
If the pilot is dependent on another pilot or organization it should be stated here.

Experimentation
The details of the actual experimentation process.

Results
As results from experimentation are gathered, they have to be reported here in a suitable way.

Conclusions
The conclusions of the pilot. These two categories, at least, have to be clearly considered:

- Pilot-specific conclusions
- Contribution of the pilot to Clommunity
Acronyms

**BFS**  Backup and File Sync
**BST**  Backup Syncthing
**CN**  Community Network
**DDBB**  Database
**FLOSS**  Free, Libre, Open Source Software
**GUI**  Graphical User Interface
**HA**  High Availability
**IaaS**  Infrastructure as a Service
**LXC**  Linux Containers
**NaaS**  Network as a Service
**PaaS**  Platform as a Service
**RTT**  Round Trip Time
**SwebFS**  Syncthing web backup and file sync
**UX**  User eXperience
**VM**  Virtual Machine
**WP**  Work Package (*EC projects specific*)
Bibliography


Licence

The CLOMUNITY project, June 2015, CLOMUNITY-201506-D4.5:

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 Unported License.