2015 IEEE 29th International Conference on Advanced Information Networking and Applications

TROBADOR: Service Discovery for Distributed Community Network Micro-Clouds

Mennan Selimi, Felix Freitag
Department of Computer Architecture
Universitat Politècnica de Catalunya
Barcelona, Spain
{mselimi, felix}@ac.upc.edu

Roger Pueyo Centelles, Agustí Moll
Fundació Privada per a la Xarxa
Lliure, Oberta i Neural Guifi.net
Barcelona, Spain
{roger.pueyo, agusti.moll}@guifi.net

Luís Veiga
Instituto Superior Técnico (IST)
INESC-ID Lisboa
Lisbon, Portugal
luis.veiga@inesc-id.pt

***Abstract*—Community networks are decentralized communication networks built and operated by citizens, for citizens. We consider service discovery for clouds in such community networks. The currently employed solutions for service discovery are static and are not able to follow the dynamics in cloud service provision. We propose a novel service discovery mechanism based on the common Linux tool Avahi combined with TincVPN, in which the VPN allows Avahi to reach nodes beyond the local link. The concept of the micro-cloud is introduced which contains the nodes reached with the extended broadcast domain of Avahi. We evaluate the performance of the proposed service discovery mechanism in a real community cloud deployment consisting of 25 geographically distributed nodes. In experiments with different settings and number of services, we measure the number of services discovered as a function of the discovery time. Our results show that while a client eventually discovers all the services, a significant time is needed to achieve a complete service discovery. The proposed mechanism therefore seems appropriate for the case of community clouds with many replicas of a sought service, where the fast discovery of just a few providers satisfies the requirements of a client. By applying standard Linux tools, an advantage of our solution is that it is by default available on most Linux distribution, which allows our solution to be easily used for real deployments.**

***Index Terms*—community networks; community cloud; service discovery**

I. Introduction

Wireless community networks are large scale, self-organized and decentralized networks which are deployed and maintained by their users. Different from the traditional (business-focused) model used by the telecommunication operators, in which the infrastructure is owned by the operator, in community networks the users contribute the infrastructure. These networks are normally open, free and often involve non-profit organizations which can cooperate with local stakeholders to develop community services, including local networking, voice connections and Internet access [1].

Hundreds of community networks operate across the globe, in rural and urban, rich and poor areas. There are several large community networks in Europe having from 500 to 25000 nodes, such as Guifi.net[1], FunkFeuer[2], AWMN[3], Freifunk[4]

[1]http://guifi.net
[2]http://www.funkfeuer.at
[3]http://www.awmn.gr
[4]http://freifunk.net/

Figure 1. Guifi.net nodes and links in the area around Barcelona

and many more worldwide. Most of them are based on Wi-Fi technology (ad-hoc networks, IEEE 802.11a/b/g/n access points in the first hop, long-distance point-to-point WiFi links for the trunk network), because WiFi technology seems to be the easiest and cheapest way to deploy an outdoor network. Figure 1 shows as an example the wireless nodes, links and outdoor devices used by Guifi.net community network in the area of Barcelona, Spain. Guifi.net is considered one of the largest community networks worldwide having more than 25000 operational nodes [2].

The concept of community clouds has been introduced in its generic form before, e.g. [3], [4], as a cloud deployment model in which a cloud infrastructure is built and provisioned for an exclusive use by a specific community of consumers with shared concerns and interests. We refer here to a specific kind of a community cloud, i.e. clouds in community networks, in which sharing of computing resources is from within community networks, while using the service models of cloud computing in general [5]–[7].

Cloud service discovery is essential for allowing cloud usage and user participation. Service discovery involves service providers publishing services and clients being able to search and locate service instances. Since community cloud nodes are distributed all over the network and administrated by the their owners, different to datacentre type clouds a mechanism is needed that allows the cloud users to discover services offered by other community cloud nodes and announce their own services.

The contribution of this paper is an experimental perfor-

1550-445X/15 $31.00 © 2015 IEEE
DOI 10.1109/AINA.2015.248

mance assessment of a practical service discovery solution named TROBADOR. TROBADOR is a system based on creating a layer-2 TincVPN among the cloud nodes that are distributed over the network, while using the standard Linux tool Avahi for service announcement and discovery.

Our approach for the performance assessment of the service discovery in community clouds is to reproduce in the experiments the conditions as seen from the end user: to experiment in production community networks, focus on metrics that are of interest for end users and assess them with applications on real nodes integrated in community networks. We leverage on the cloud infrastructure provided by an on-going cloud deployment in the Guifi community network and a testbed deployed in community networks [8] [9].

The rest of the paper is organized as follows. Section II defines the community clouds and discusses the requirements for such clouds. Section III explains some of the service discovery protocols. In section IV we show how we publish and discover services in community clouds and explain our Avahi-TincVPN implementation. Section V describes the experimental setup. In Section VI we analyze our results. Section VII describes related work and section VIII concludes and discusses future research directions.

II. Clouds in Community Networks

A. Community cloud scenario

In this section we elaborate the conditions and scenarios for community networks. It provides the context for the the service discovery solution we study in the remaining sections.

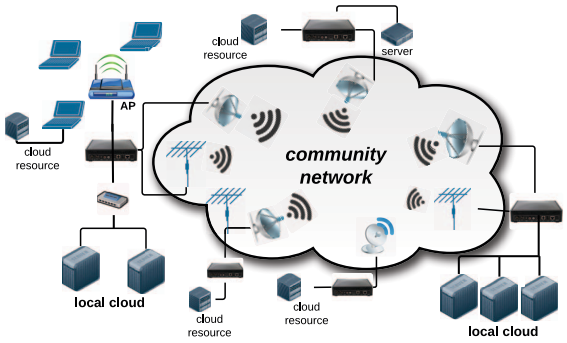

Figure 2. A community network with local clouds

The community network generally has two different types of nodes, super nodes (SNs) and client nodes (CNs) and each type plays a different role in the network. Each client node has a unique connection to a super node that routes traffic, and super nodes can have many connected client nodes.

We consider Figure 2 to derive the scenarios of clouds in community networks. The picture shows some typical community nodes with a router and some server or clients attached to it. Some CNs are shown that are connected to the access point (AP) of a SN. In addition, however, these community nodes have hosts, cloud resources and even small cloud data centers attached to them, illustrating the vision of the *community cloud*. In such local community cloud, a super node is responsible for the management of a set of attached nodes that as hosts contribute cloud resources. From the perspective of the attached nodes, this super node acts as a centralized unit to manage the cloud services in that vicinity. A local community cloud is not necessarily built with high-end machines, we rather see small data centres built with commodity desktop machines and managed by popular cloud operating systems such as Proxmox[5], OpenNebula[6] or OpenStack[7]. A set of local community clouds may form federated clouds. Cloud federation may enable the allocation of resources on several local clouds. Distributed applications for community networks will need to be deployed over federated clouds [10] [11]. Similar to content-distribution networks, requests from client nodes could be allocated to the most appropriate cloud resource.

B. Cloud-based services in community networks

Services in the community cloud are provided through instances of the Cloudy[8] distribution deployed on the cloud nodes. *Cloudy* is a Debian-based distribution which has been equipped with a set of basic platform services and applications. Figure 3 indicates some applications already integrated in Cloudy. It can be seen that the applications which we consider in this paper, Avahi[9] and Tinc[10], are available in the Cloudy distribution. Figure 3 also shows that the community services provided by the distribution aim to run on a combination of cloud resources. On IaaS level, these can be provided by cloud management platforms such as OpenStack or OpenNebula, and even include low-resource devices such as those provided by CONFINE's Community-Lab[11] testbed.

1) Micro-Clouds: The concept of micro-clouds is introduced in order to split deployed community cloud nodes into different groups. A micro-cloud refers to these cloud nodes which are within the same service announcement and discovery domain. Different criteria can be applied to determine to which micro-cloud a node belongs to. Applying technical criteria (e.g. RTT, bandwidth, number of hops, resource characteristics) for micro-cloud assignment is a possibility to optimize the performance of several applications. But also social criteria may be used, e.g. bringing in a micro-cloud cloud resources together from users which are socially close may improve acceptance, the willingness to share resources and to maintain the infrastructure.

2) Cloudy architecture: The internal architecture of the Cloudy distribution is depicted in Figure 4, inside the central rectangle. On the bottom part, the virtual L2 over L3 network provides the overlay to interconnect all the nodes within a micro-cloud. This overlay network (created by the TincVPN) is used in the service announcement and discovery processes,

[5]http://www.proxmox.com/
[6]http://www.opennebula.org/
[7]http://www.openstack.org/
[8]http://cloudy.community/
[9]http://avahi.org/
[10]http://tinc-vpn.org/
[11]http://community-lab.net/

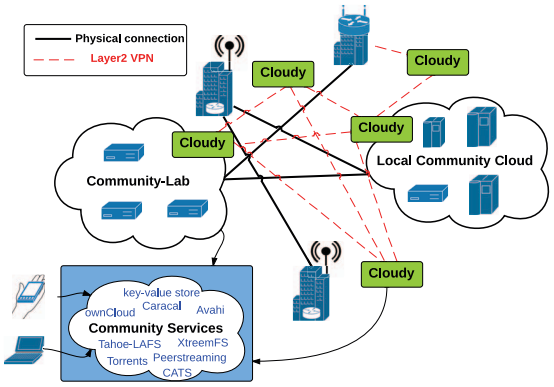

Figure 3. Cloud-based services in the distributed community cloud

that respectively publish local information to the cloud and receive data from other cloud nodes.

The service announcement daemon in the implementation periodically checks the status of the available services by means of their corresponding plug-in and publishes the information. The nodes in the micro-cloud receive this information and update their list of services. When a service in a node is stopped or removed, this information is also announced to the rest of the cloud servers. The service discovery daemon continuously listens for announcements from other peers, and keeps an updated list of the available services in the micro-cloud. This list can be requested directly by the services or by the end user. Operations can be performed either by the command-line interface (console) or using the Cloudy web-based management interface.

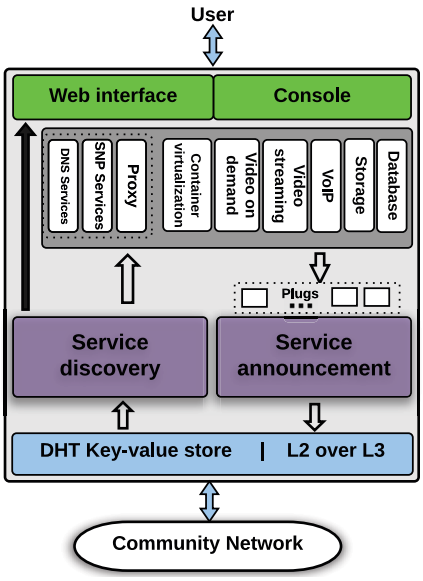

Figure 4. Cloudy architecture

III. Discovery protocols

In order to perform service discovery there are two architectural approaches to consider: directory based and non-directory based approaches. A directory based approach is centralized and all operations rely on a single point of failure. But service discovery operations are not possible if the central server (registry) or routes to it are not available. Therefore, for the case of dynamic infrastructures such as wireless mesh networks it is not suggested. In the case of non-directory based approach, broadcasting or multicasting is used to disseminate messages. The main advantage of this architecture is that no administration is needed and that it is not dependent on the infrastructure component of a central server. We focus in the following on non-directory based approaches for service discovery.

A. DNS-SD, mDNS and Zeroconf

In the simplest terms DNS-based Service Discovery (DNS-SD)[12] is a method of service discovery which offers to clients the opportunity to discover a list of named instances of the desired service using only standard DNS-messages. DNS-SD can be used in combination with multicast, which is then called multicast DNS (mDNS)[13]. mDNS is an IETF protocol which allows DNS operations on the local link without a unicast DNS server. For this purpose, mDNS uses a part of the DNS namespace, which is available for local use. DNS-SD is not dependent on the mDNS technology, meaning service discovery with DNS-SD can be done with a traditional DNS server setup. However, we must use the mDNS technology in order to eliminate the need for a DNS server in our quest for decentralization. Apple's Bonjour Protocol is the implementation of mDNS and DNS-SD protocols. As Bonjour became widespread, the term Zeroconf[14] became synonymous with the abstraction that Bonjour implements, namely the mDNS and DNS-SD protocols.

B. Avahi

Avahi is a framework for service discovery on local networks, which uses the same specifications as Apple's Bonjour, multicast DNS and DNS Service Discovery from the Zero Configuration Networking (Zeroconf) working group. Avahi allows programs to publish services and to discover services that are available on other machines in the local link. As an example, a user could find local printers without needing to know their IP address, or which computers have shared folders. However Avahi's multicast design makes it difficult to reach beyond the local link. As a solution to this problem, we apply TincVPN, a Virtual Private Network (VPN) daemon that uses tunneling and encryption to create a secure private network between hosts of different domains.

[12]https://tools.ietf.org/html/rfc6763
[13]http://www.multicastdns.org/
[14]http://www.zeroconf.org/

IV. Service Discovery in Community Clouds

Cloud services in community networks need a common place for both providers and users to respectively, publish their services and learn about their availability.

A. Need for service discovery mechanism

Guifi.net previously declared the available network services on a web page, by manually submitting the details (type of service and specific characteristics, location, IP address, terms of usage, etc.). The lack of automated methods for publishing services, and also for conveniently finding out which are the ones closest to the user, has lead to a couple of drawbacks: not all the services are declared on the website (although they are announced by other means, like users mailing lists) and when a service is temporarily or permanently unavailable, it still appears on the website as *offline* until it is manually removed from the list.

Providing an automated method for publishing and discovering services can help in three aspects within community networks:

- Minimum manual intervention, or none at all, is required to publish services and update their status by the providers;
- Users can retrieve an updated list of the services available in their vicinity;
- Distributed services and applications that need to cooperate with remote instances can offer and demand resources in real time.

While the first and the second aspects are more tied to the management and the user experience of the cloud services, the third can facilitate the process of deploying services that spread over several nodes, like a distributed file system.

B. Avahi and Tinc for service discovery

The service publishing and discovery mechanism included in the Cloudy distribution is based on Avahi, a free Zeroconf implementation. By means of a series of scripts that are executed periodically on a per-service basis, the list of available services is subsequently published using the Avahi daemon broadcast mechanism. Then, the rest of nodes (which have the Avahi daemon running too) receive the information and store it locally until the next update is received, or a timeout is reached. This information can then be used internally by the node for operational purposes or be shown to the end user on the node's web interface.

Avahi works as a service publishing and discovery system for the broadcast domain of the nodes (this is in a network segment of the data link layer to which all the nodes are connected). However, most commonly in community networks, devices are spread over different nodes that belong to different broadcast domains. Therefore, Avahi packets can not be directly exchanged between one node and another. To solve this limitation, a virtual network connecting all the cloud devices is created. Using this layer-2 network, all the devices *appear* to in the same broadcast domain, and Avahi packets can be exchanged between different nodes beyond the local link.

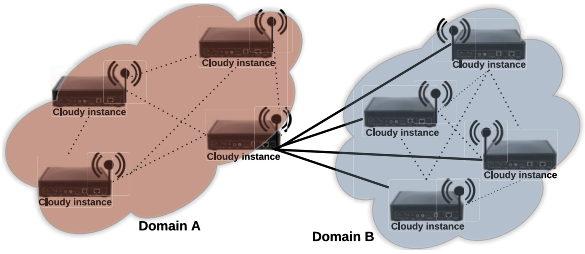

Figure 5. TincVPN over nodes from two different domains. A point-to-point connection is established between all nodes of Domain A and B. The picture highlights how one node from Domain A connects to all nodes of the Domain B.

The software chosen to create this virtual network between the Cloudy devices is TincVPN, a Virtual Private Network (VPN) daemon that uses tunnelling and encryption to create a secure private network between hosts on the Internet. Figure 5 shows how TincVPN makes devices appear as they are in the same broadcast domain (dotted lines indicate the connection of nodes inside the same domain, solid lines indicate the connection of nodes between two collision domains). The encrypted tunnels allows VPN sites to share information with each other over the Internet without exposing any information to others. Tinc also allows more than two sites to connect to each other and form a single VPN. Traditionally VPNs are created by making tunnels, which only have two endpoints. Larger VPNs with more sites are created by adding more tunnels. Tinc takes another approach: only endpoints are specified. Tinc itself takes care of creating the tunnels. This allows for easier configuration and improved scalability .

Tinc is automatically installed and configured on every Cloudy device, ready to be activated (for privacy reasons, this option is left to the user's choice). Right after activation, a VPN is started to reach the rest of Cloudy nodes and Avahi can communicate transparently with the other nodes.

C. Real use case example

Tahoe-LAFS[15] is a free and open cloud storage system. It distributes the data across multiple servers. Even if some of the servers fail, the entire filesystem continues to function correctly, preserving privacy and security. A Tahoe-LAFS cluster consists of a set of storage nodes, client nodes and a single coordinator node called the Introducer. The storage nodes connect to the Introducer and announce their presence, and the client nodes connect to the Introducer to get the list of all connected storage nodes. Tahoe-LAFS is a service provided by several community cloud nodes [12] .

The configuration of Tahoe-LAFS in a Cloudy node can be done almost automatically using the web interface, where the user only needs to introduce some basic information.

The process of deploying a whole storage grid is assisted by the Avahi service discovery mechanism. To create a Tahoe-LAFS Introducer on a node, basic configuration parameters are introduced (the grid name, the node name and optionally

[15]https://www.tahoe-lafs.org/trac/tahoe-lafs

Figure 6. Partial screenshot of the Cloudy web interface showing the service discovery section. Four different Tahoe-LAFS storage grids are shown.

if the default value is inadequate for some reasons the port the service will be listening to). As soon as the service is started, if it is marked as public, it is announced to the rest of Cloudy nodes. After that, adding a storage node to join the grid is just a matter of three steps:

- Listing the available services and browse to the Tahoe-LAFS storage section to find the desired Introducer
- Clicking on the "Join grid" button for the desired grid (see Figure 6)
- Choose a name for the storage node and apply changes to deploy it (see Figure 7)

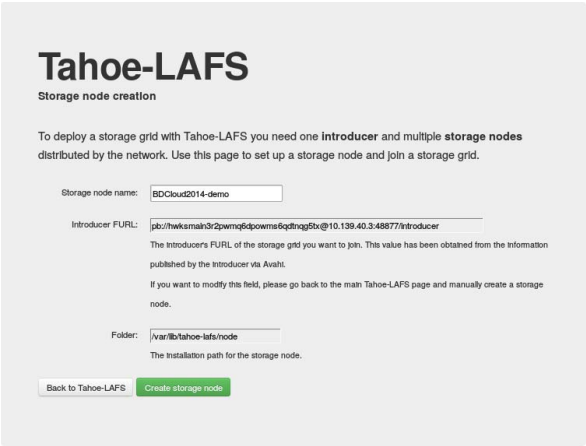

Figure 7. Partial screenshot of the Cloudy web interface showing the form for joining a Tahoe-LAFS grid with a storage node.

V. Experimental Setup

A. Selection of cloud nodes

We perform some experiments to evaluate the described service discovery mechanism. The main configuration for the experiments includes nodes of two geographically distant community networks: Guifi.net in Spain and AWMN (Athens Wireless Metropolitan Network) in Greece. The nodes of our experiments are real nodes with Guifi.net and AWMN IP addresses. The connectivity between community network nodes varies significantly. We observe that network characteristics are not symmetric. Both community networks (Guifi.net and AWMN) are connected through a tunnel over Internet to enable network federation at IP level.

In order to deploy service discovery experiments in a realistic community cloud setting, we use the Community-Lab testbed, a distributed infrastructure provided by the CONFINE[16] project, where researchers can deploy experimental services, perform experiments or access open data traces. We use some of the available nodes of that testbed for deploying the applications.

Table I
Nodes, their location and RTT from the client node

Nr. of nodes	Community Network	Location	RTT
13	Guifi.net	Barcelona, Spain	1-7 ms
7	Guifi.net	Catalonia, Spain	10-20 ms
5	AWMN	Athens, Greece	90-100 ms

We use in total 25 nodes spread between two community networks (see Table I). We use 20 nodes from Guifi.net community network, where 13 of the nodes are located in the city of Barcelona and 7 of them are located in the Catalonia region of Spain. From AWMN we use 5 nodes which are distributed in Athens, Greece. Figure 8 shows the average aggregated bandwidth of nodes from Guifi.net and AWMN during the experiment time, measured for a time period of one week (7 hours a day). Most Community-Lab nodes are built with a Jetway device that is equipped with an Intel Atom N2600 CPU, 4GB of RAM and 120GB SSD. Nodes (i.e. research devices in CONFINE terminology) in the Community-Lab testbed run a custom firmware (based on OpenWRT[17]) provided by CONFINE, which allows running on one node several slivers simultaneously implemented as Linux containers (LXC). A sliver is defined as the partition of the resources of a node assigned to a specific slice (group of slivers). We can think of slivers as virtual machines running inside of a node. The Community-Lab nodes deploy the Cloudy distribution in slivers.

B. Metrics

We focus in our experiments on the responsiveness of the discovery mechanism. We consider responsiveness as the probability of successful operation within deadlines. Applied to our case, it refers to successful service discovery within given time limits.

We run the discovery requests from a single dedicated discovering node or service client that searches and locates the service instances. The service client is located in the UPC campus at Barcelona (Guifi.net node). All other nodes act as service providers responding to discovery requests. The service providers are spread between two community networks. Discovery times are measured at the client when the request is sent till the response is received to obtain the user-perceived responsiveness. During the experiments no other nodes join or leave the network so no configuration on the network layers occur during measurements which could interfere with the discovery operation. Service discovery is considered successful

[16]https://confine-project.eu
[17]https://openwrt.org

when all instances have been discovered. Discoveries are considered failed (aborted) after a 1 minute waiting time (max time). We repeat the experiments several times to avoid the effects of particular network conditions in the results.

After service discovery, a client should have enough information in order to contact a service instance. Hence, discovery in our case means resolving the IP address and port of every service instance. We use different services to be discovered in our experiments, among them Tahoe-LAFS distributed storage service, print service, etc.

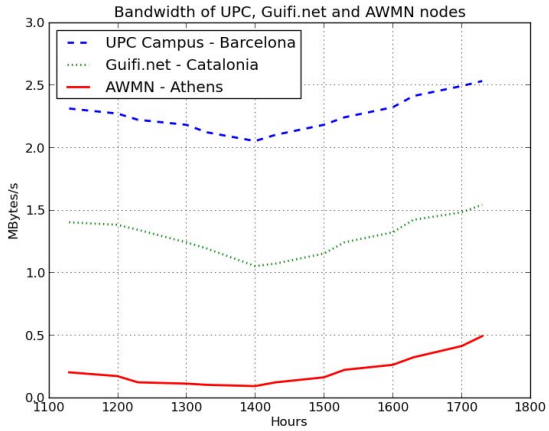

Figure 8. Average aggregated bandwidth of different group of nodes in our cluster

C. Experiment scenarios

To assess the applicability of decentralized discovery mechanisms in community networks, three scenarios are chosen that reflect common use cases of service discovery. The parameters used in the scenarios are summarized in Table II.

Table II
SUMMARY OF OUR SCENARIO PARAMETERS

Scenario	1	2	3
Number of service providers	1	25	25
Maximum discovery time	10 sec	30 sec	1 min

1) *Scenario 1: Single service discovery*
Our first goal is to measure the responsiveness of single service discovery. In this scenario, the service network consists of one client and one provider. The client is allowed to wait up to 10 seconds for a positive response. This is a common scenario for service discovery and can be considered as the baseline. Only one answer needs to be received and there is enough time to wait for it (we consider 10 seconds is a reasonable time for our community network settings). In this case, the client discovers a Tahoe-LAFS distributed storage service.

2) *Scenario 2: Timely service discovery of the same service type*
Service networks are populated with multiple instances of the same service type (e.g. Tahoe-LAFS service). The clients needs to discover as many instances as possible and will then choose one that optimally fits its requirements. In this scenario we have one service client and 25 service providers. The discovery is successful if all 25 provided service instances have been discovered. We measure how responsiveness increases with time given. The client waits 30 seconds to receive responses.

3) *Scenario 3: Timely service discovery of all services in the network*
In this scenario we have one service client and 25 service providers that offer services of different types (Tahoe-LAFS, print service etc). The client needs to discover all instances of the different service types. Here the service providers offer more than one service. The discovery is successful if all services of the different types published from 25 providers are discovered. The client waits 1 minute to receive the responses from the service providers.

VI. EXPERIMENTAL RESULTS

In this section the results for the three scenarios described in Section V are presented and discussed.

A. Scenario 1 results: Single service discovery

Discovery of a single service instance within ten seconds proved to be reasonable responsive. Our client node is located in the Guifi community network (UPC Campus Barcelona). When the service provider is located also in the Guifi.net community network, it took approximately 4 seconds for the client to discover the service. On the other side, it took 9 seconds for the client to discover a service from the AMWN community network service provider.

B. Scenario 2 results: Timely service discovery of the same service type

Figure 9 shows that discovery of services increases rapidly with the time where the size of the circle is proportional to the number of services discovered. In this scenario the client node waits 30 seconds to receive the responses from service publishers. In the first 5 seconds, the client receives 5 service responses from the publishers. The service published in this case is the Tahoe-LAFS distributed storage service. These 5 services received in the first 5 seconds are close to the client (nodes in the UPC campus). After 20 seconds the last 5 services are discovered. These are the five service providers from AWMN in Athens, Greece. In total, the client discovers 25 services published from 25 service providers spread across two community networks.

C. Scenario 3 results: Timely service discovery of all services in the network

Figure 10 shows the number of different types of services discovered in a time window of 1 minute. Different types

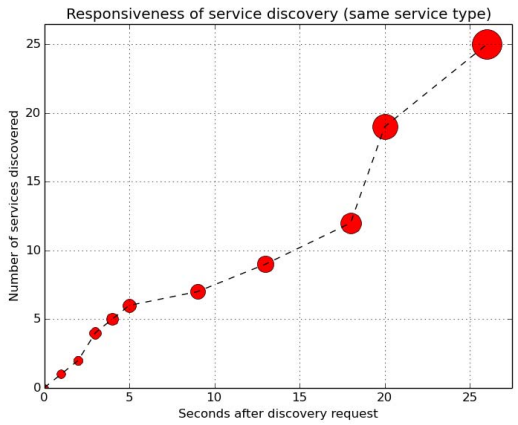

Figure 9. Responsiveness of service discovery (same service type)

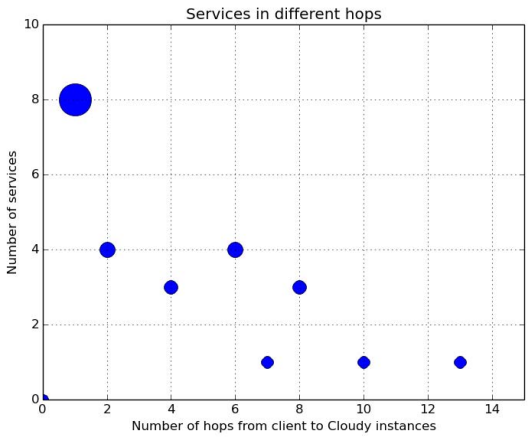

Figure 11. Services discovered in different hops

of services present in the network, such as Tahoe-LAFS and print service are discovered. However, the location of nodes in Greece and the high diversity of the quality of wireless links in Guifi.net [13] and AWMN leads to an increase of service discovery time. Around 10 services are discovered between the fortieth and the fiftieth seconds. Figure 11 shows the number of hops from the client to other Cloudy instances. Also it shows the number of services discovered at each hop.

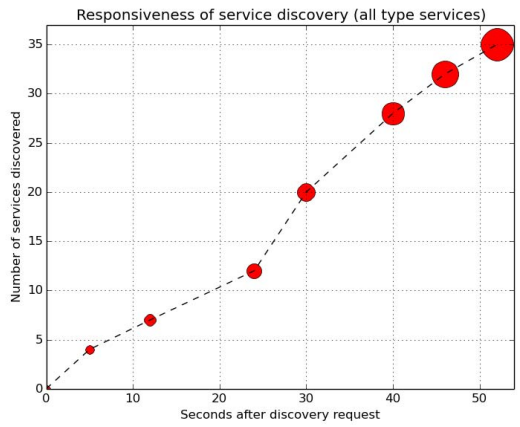

Figure 10. Responsiveness of service discovery (all type services)

VII. Related Work

In terms of examining the dependability aspects of decentralized service discovery concepts in unreliable networks, Dittrich's and Salfner's paper [14] is the most relevant to our work. The authors evaluate the responsiveness of domain name system (DNS) based service discovery under influence of packet loss and with up to 50 service instances. Their empirical results show that the responsiveness of the used service discovery mechanisms decreases dramatically with moderate packet loss of around 20 percent. However their experimental evaluation is based on simulations and has not been applied to wireless scenarios.

The research described in [15] presents an alternative approach to extend the limit of Zeroconf beyond the local link. Here, the robustness of existing discovery mechanism is evaluated under increasing failure intensity. However, responsiveness is not covered in particular. The z2z toolkit [16] combines the Zeroconf with the scalability of DHT-based peer-to-peer networks allowing services to reach beyond local links. z2z connects multiple Zeroconf subnets using OpenDHT.

Robustness of service discovery with respect to discovery delay times is addressed in [17]. The work of [18] describes and compares through simulation the performance of service discovery of two IETF proposals of distributed DNS: Multicast DNS and LLMNR (Link-Local Multicast Name Resolution). The authors propose simple improvements that reduce the traffic generated, and so the power consumption.

In wireless mesh network settings, the work of Wirtz [19] proposes DLSD (DHT-based Localized Service Discovery), a hierarchy of localized DHT address spaces that enable localized provision and discovery of services and data. Another work [20] proposes a stochastic model family to evaluate the user-perceived responsiveness of service discovery, the probability to find providers within a deadline, even in the presence of faults.

From the review of the related work it can be seen that the reviewed experimental studies related to service discovery were not conducted in the context of the community networks, which we address as scenario. Also, we proposed a novel usage of Avahi in combination with Tinc, which was not investigated before. Finally, our work conducted the service discovery evaluation in a real deployment scenario to understand its performance and operation feasibility under real network conditions.

VIII. Conclusion and Outlook

A novel combination of the common Linux tool Avahi with TincVPN as a mechanism to achieve service discovery in community network clouds was presented. Tinc establishes a VPN among the cloud nodes and this way creates a virtual layer 2 extension beyond the local link. As a consequence, Avahi's publication of the services of a cloud node reaches even the other nodes beyond the local link. This allows users and cloud nodes to publish services and find services published by others even when they are on other network segments, a common case in distributed clouds.

The concept of micro-clouds was introduced as the set of these cloud nodes which are interconnected by the TincVPN and which thus are part of the broadcast domain of Avahi, i.e. can publish services and receive service announcements from the other nodes.

Finding out about the size, which the proposed micro-clouds should have, is the research question to answer. It is clear that the number of nodes of a micro-cloud will need to be limited, since Avahi and Tinc are not scalable, due to employing broadcast within their protocols. We therefore conducted experimental studies with different sizes of micro-clouds and number of services to understand better the limits of the proposed Avahi-Tinc solution.

For the experimental evaluation of the proposed service discovery mechanism, a real system was used consisting of up to 25 heterogeneous geographically distributed community cloud nodes. We measured experimentally the relation of the number of discovered services as a function of discovery time.

We observed that for a full discovery in larger micro-clouds, a significant time is needed until all services are found. If such a complete discovery is sought by a machine process, this time may be tolerable for several applications. However, if a human conducts a complete discovery, the quality of experience may be insufficient. The results of our evaluations suggest that for popular services, which are available on many cloud nodes, a time-limited search should be applied and will return a sufficiently large number of discovered service instances. For rare services, however, additional mechanisms may be needed to improve the user experience, especially if a large micro-cloud cannot be split into smaller ones and if it includes several nodes distant to each other.

Future work will look at combining the studied Avahi-Tinc discovery with other mechanism, such as DHT-based search or gossiping, to address for several micro-clouds the inter micro-cloud service discovery.

Acknowledgement

This work was supported by the European Framework Programme 7 FIRE Initiative projects CONFINE (FP7-288535), CLOMMUNITY (FP7-317879), Universitat Politècnica de Catalunya-BarcelonaTECH and by the Spanish government under contract TIN2013-47245-C2-1-R.

References

[1] B. Braem *et al.*, "A case for research with and on community networks," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 3, pp. 68–73, Jul. 2013.

[2] L. Cerdà-Alabern, A. Neumann, and P. Escrich, "Experimental evaluation of a wireless community mesh network," in *Proceedings of the 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, ser. MSWiM '13. New York, NY, USA: ACM, 2013, pp. 23–30.

[3] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *NIST Special Publication*, vol. 800, no. 145, 2011.

[4] A. Marinos and G. Briscoe, "Community Cloud Computing," *Cloud Computing*, vol. 5931, pp. 472–484, 2009.

[5] A. M. Khan, U. C. Büyükşahin, and F. Freitag, "Prototyping Incentive-based Resource Assignment for Clouds in Community Networks," in *28th International Conference on Advanced Information Networking and Applications (AINA 2014)*. Victoria, Canada: IEEE, May 2014.

[6] J. Jiménez, R. Baig, F. Freitag, L. Navarro, and P. Escrich, "Deploying PaaS for Accelerating Cloud Uptake in the Guifi.net Community Network," in *International Workshop on the Future of PaaS 2014, within IEEE IC2E*. Boston, Massachusetts, USA: IEEE, Mar. 2014.

[7] M. Selimi, F. Freitag, R. Centelles, and A. Moll, "Distributed Storage and Service Discovery for Heterogeneous Community Network Clouds," in *7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC'14)*. London, UK: IEEE/ACM, Dec. 2014.

[8] J. Jimenez *et al.*, "Supporting Cloud Deployment in the Guifi.net Community Network," in *5th Global Information Infrastructure and Networking Symposium (GIIS 2013)*, Trento, Italy, Oct. 2013.

[9] M. Selimi, F. Freitag, R. Pueyo, P. Escrich, D. Marti, and R. Baig, "Experiences with Distributed Heterogeneous Clouds over Community Networks," in *SIGCOMM Workshop on Distributed Cloud Computing (DCC'14), within ACM SIGCOMM*. Chicago, USA: ACM, Aug. 2014.

[10] R. Buyya, R. Ranjan, and R. N. Calheiros, "InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services," *Algorithms and Architectures for Parallel Processing*, vol. 6081, pp. 20–31, Mar. 2010.

[11] M. Gall *et al.*, "An Architecture for Community Clouds Using Concepts of the Intercloud," in *27th International Conference on Advanced Information Networking and Applications (AINA'13)*. Barcelona, Spain: IEEE, Mar. 2013, pp. 74–81.

[12] M. Selimi and F. Freitag, "Tahoe-LAFS Distributed Storage Service in Community Network Clouds," in *4th IEEE International Conference on Big Data and Cloud Computing (BDCloud 2014)*. Sydney, Australia: IEEE, Dec. 2014.

[13] D. Vega, L. Cerda-Alabern, L. Navarro, and R. Meseguer, "Topology patterns of a community network: Guifi.net," in *1st International Workshop on Community Networks and Bottom-up-Broadband (CNBuB 2012), within IEEE WiMob*, Barcelona, Spain, Oct. 2012, pp. 612–619.

[14] A. Dittrich and F. Salfner, "Experimental responsiveness evaluation of decentralized service discovery," in *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, April 2010, pp. 1–7.

[15] C. Dabrowski, K. Mills, and S. Quirolgico, "Understanding failure response in service discovery systems," *Journal of Systems and Software*, vol. 80, no. 6, pp. 896 – 917, 2007.

[16] J. W. Lee *et al.*, "z2z: Discovering zeroconf services beyond local link," in *Globecom Workshops, 2007 IEEE*, Nov 2007, pp. 1–7.

[17] C.-S. Oh *et al.*, "A hybrid service discovery for improving robustness in mobile ad hoc networks," in *IEEE International Conference on Dependable Systems and Networks (DSN 2004)*, 2004.

[18] C. Campo and C. García-Rubio, "Dns-based service discovery in ad hoc networks: Evaluation and improvements," in *Personal Wireless Communications*, ser. Lecture Notes in Computer Science, P. Cuenca and L. Orozco-Barbosa, Eds. Springer Berlin Heidelberg, 2006, vol. 4217, pp. 111–122.

[19] H. Wirtz, T. Heer, M. Serror, and K. Wehrle, "Dht-based localized service discovery in wireless mesh networks." in *MASS*, 2012, pp. 19–28.

[20] A. Dittrich, B. Lichtblau, R. Rezende, and M. Malek, "Modeling responsiveness of decentralized service discovery in wireless mesh networks," in *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, ser. Lecture Notes in Computer Science, K. Fischbach and U. Krieger, Eds. Springer International Publishing, 2014, vol. 8376, pp. 88–102.