KTH Royal Institute of Technology

Bachelor Thesis

Comparison of Virtual Networks Solutions for Community Clouds

Author:
Albert Avellana

Examiner:
Vladimir Vlassov

Supervisors:
Paris Carbone, Hooman Peiro

Information and Communication Technology School

February 2014

KTH Royal Institute of Technology

Abstract

Information and Communication Technology School

Bachelor Thesis

Comparison of Virtual Networks Solutions for Community Clouds

by Albert Avellana

Cloud computing has a huge importance and big impact nowadays on the IT world. The idea of community clouds has emerged recently in order to satisfy several user expectations. Clommunity is a European project that aims to provide a design and implementation of a self-configured, fully distributed, decentralized, scalable and robust cloud for a community of users across a commmunity network. One of the aspects to analyze in this design is which kind of Virtual Private Network (VPN) is going to be used to interconnect the nodes of the community members interested in access cloud services. In this thesis we will study, compare and analyze the possibility of using Tinc, IPOP or SDN-based solutions such as OpenFlow to establish such a VPN.

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to do this thesis in KTH. Firstly, I would like to thank Vlad for the opportunity he gave me to do this thesis and for his support. Secondly, thanks to my thesis supervisors: Paris Carbone and Hooman Peiro, who guided me through the research, helped me and gave me recommendations during this period.

Also, I would like to thank Félix Freitag and Leandro Navarro from Universitat Politècnica de Catalunya for supporting me from Barcelona and make this stay in Stockholm possible. Furthermore, I would also like to thank Pau, Roger, Jorge, Javi and all the members of Clommunity and Confine Project for their assistance and guidance on some aspects of my thesis.

Lastly, thanks to my friends and family for encouraging me and motivating me during the development of this thesis.

Contents

List of Figures

Abbreviations

ARP	**A**ddress **R**esolution **P**rotocol
CN	**C**ommunity **N**etwork
DHCP	**D**ynamic **H**ost **C**onfiguration **P**rotocol
DHT	**D**istributed **H**ash **T**able
ICT	**I**information & **C**ommunications **T**echnology
IPOP	**IP** **O**ver **P**2P
LAN	**L**ocal **A**rea **N**etwork
NAT	**N**etwork **A**ddress **T**ranslator
P2P	**P**eer **T**o **P**eer
QoS	**Q**uality **o**f **S**ervice
RTT	**R**ound **T**rip **D**elay
SDN	**S**oftware **D**efined **N**etwork
STUN	**S**ession**T**ransversal **U**tilities for NAT
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
UDP	**U**ser **D**atagram **P**rotocol
VLAN	**V**irtual **L**ocal**A**rea **N**etwork
VM	**V**irtual **M**achine
VNI	**V**irtual **N**etwork **I**nterface
VPN	**V**irtual **P**rivate **N**etwork

Chapter 1

Introduction

In the last years cloud computing has been rising incredible popularity and becoming a routinary working tool for many people and companies. Many businesses use cloud services offered by big companies such as Amazon or Google. Meanwhile, another model of network infrastructure has been slowly emerging: community networks (CN). This kind of network focus on satisfy a community's demand for ICT services and Internet access. A community cloud is based on a community network. Existing CNs offer free services to community members such as e-mail or web space, nonetheless, they lack many other features and services like storage or media streaming applications. Also, most of existing services lack elasticity, scalability and reliability needs. The need of these services and features evolved to the idea of creating community clouds using the already proven and tested infrastructure of community networks. An example of a community network is guifi.net : a WiFi community mesh network mainly deployed in Catalonia with more than 17.000 operational nodes and more than 30.000 km of links [1].

The main difference between community clouds and public or private clouds is that community clouds are distributed among the participating users/entities without the need to rely on any other third parties [2]. Furthermore, community clouds have many benefits including increased privacy and security while also introducing challenges to face such as the heterogeneity and dynamism of the network itself. The design and creation of the community cloud is a big step because many things have to be changed to get the idea of a "cloud" working. One of the problems that have to be solved is the way nodes communicate among each other and offer services discovery and sharing, therefore, which virtual network software we can use to do so. This private network has to be decentralized and it has three main requirements: scalable, robust and self-configuring. It has to be scalable due to the unknown number of members of the community; a community can be formed by a few members such as in a building or it can have many more like in a small town. Also, it has to be robust and be able to manage join/leave/fail of nodes because the topology is very dynamic due to the continuous change of the number of users. Lastly the virtual private network has to be self-configuring or easy to configure. In many cases the end-users of community clouds will not have the knowledge to establish difficult connections, therefore having an easy to configure with the minimum participation of the user it is also an important feature. Furthermore, we have to take into consideration the dynamical behavior of the network and how it can be constantly changing with new links joining the set of nodes and other leaving/failing. In this project we will study and analyze different solutions to find the best way to automate neighbor discovery, enable self-configuration of the nodes in a scalable and robust virtual network.

There are many VPN solutions that could accomplish this requirements such as: OpenVPN [3], FreeLan [4] or PeerVPN [5]. However, in this thesis, the analysis is done with three specific solutions: Tinc VPN, IPOP, and OpenFlow; and an evaluation and comparison is done between Tinc and IPOP. Firstly ,Tinc is chosen because it is already used in Community networks such as FreiFunk 2 [6]. Tinc is an overlay based VPN solution used to interconnect nodes in a private network. Tinc is a scalable, stable and reliable, with easiness of configuration, and flexible P2P VPN. Secondly, another overlay solution for VPNs would be to use IPOP virtual network [7] because it is a quite new solution which, not only fits all the requirements , but only has some advantages such as self-optimizing connections or decentralized NAT transversal. IPOP creates a virtual network which supports the deployment of IP-based protocols over a self-configuring and robust P2P overlay. Lastly, we consider the possibility of using the newest and more powerful technology: SDN, more precisely OpenFlow software but only in a conceptual way due to its complexity. In chapter 3 the study and analysis of these possibilities is done.

The goal of this thesis is to help in the decision of which network virtualization technology could be best used in Community Clouds and more specifically for the Clommunity Project [8]. The Clommunity project is a project funded by the EU and it aims at helping communities of people in bootstrapping, deploying, running and expanding community-owned networks in an easy way to provide community services and create a community cloud.

1.1 Problem statement

The idea of this thesis emerged when we were thinking of the network requirements and design for a community cloud and we got this problem:

Which is the best option to create a decentralized VPN among the nodes of a Community Network which is self-configuring, robust and scalable?

In this work three types of VPN solutions are considered as possible solutions to the aforementioned problem, Tinc, IPOP and SDNs. The purpose of this project is to enumerate and compare the features of each solution while also providing a performance evaluation of IPOP and Tinc which are currently considered the most effective VPN solutions.

1.2 Related Work

The concept of community clouds is relatively new and lacks sufficient background research. However, some researchers have studied the architecture of community clouds using concepts of the intercloud, which is the creation of a community cloud interconnecting already existing small clouds [2]. Also, related to community networks we can find papers suggesting different topologies for community networks, more precisely in Guifi.net [9].

One of the most interesting related projects is GroupVPN [10] . In this project they make use of IPOP VPN to create a user interface that lets a group of friends deploy its own VPN. It has a web interface where you can register and the user can create individual accounts or groups, as well as joining existing groups sharing the same overlay network.

Recently a project of the Vrije Universiteit called ConPaas which is an integrated cloud environment for elastic cloud applications has integrated IPOP as the virtual network technology. ConPaas aims at offering the full power of the developers of cloud application while hiding the complexity of the cloud. ConPaaS is designed to host two different applications: high performance scientific applications and online Web applications. It automates the whole life-cycle of an application. It includes collaborative development, performance monitoring, deployment, and, also, automatic scaling [11].

One of the motivators of this thesis is to aid the Clommunity Project in the decision of which VPN they can integrate in the clommunity cloud. Previously to Clommunity, a project named Confine that aimed at the creation of Community Networks Testbeds has been using Tinc as a VPN for the management network establishing connection among the nodes with an improvement based on a server package that maintains a list of all the nodes and helps on the neighbor discovery and initialization.

1.3 Thesis Outline

The remainder of this Thesis is structured as follows: Section 2 provides background information about cloud computing, community clouds, VPNs and SDNs. In section 3 the possibility of using Tinc, IPOP VPN or OpenFlow as the main VPN in community cloud is analyzed and Tinc and IPOP testbeds are detailed. In section 4 a set of tests an evaluation of Tinc and IPOP solutions are done and finally in section 5 conclusions and future work are detailed.

Chapter 2

Technical Background

This chapter gives the description of the technologies and projects that are used or mentioned on this thesis: cloud computing, community clouds, Clommunity Project, virtual private networks, Tinc VPN, IPOP and SDN.

2.1 Cloud Computing

To understand community clouds first the concept of cloud computing has to be explained. Cloud computing is a set of concepts and technologies that involve a larger set of computer connected over a private or public network (internet). with cloud computing we can run programs or applications on a distributed way and remotely. Cloud computing in many cases let the companies or users increase capacity or add capabilities on the fly without investing in new infrastructure, training new, or paying new new software licenses.

Nowadays, the term "cloud computing" is mostly used to provide hosted services on the internet. We can distinguish basically four different services that current cloud providers offer: PaaS (platform as a service), IaaS (infrastructure as a service), SaaS (software as a service) and finally HaaS (hardware as a service). Cloud users have access to cloud-based applications via a web browser, a thin client on their machine or an smartphone app because the software logic and user's data are stored on remote servers. Examples include Amazon web services and Google App engine which allocate space for a user to deploy and manage software "in the cloud". Usually, the term "cloud computing" refers to hosting services that run client server software at a remote location. Some examples include Google App Engine or Amazon web services which let the user allocate space for to deploy, run and manage all kinds of software in the cloud.

2.2 Community Clouds

There are three main architectures of clouds: public, hybrid and private. In public clouds everyone who is willing to pay for it or in some cases free can use them. In private clouds only the people from that domain or company can use the cloud, increasing the security of it regarding the access from the public network and making its management easier. When we merge or interconnect these two kinds of clouds then we get an hybrid cloud. In the last past years, people has been increasingly worrying about security, privacy, efficiency and environmental sustainability while using public cloud services of the biggest companies such as Google or Amazon [12]. Also, the high individual costs for an individual to access internet nowadays has increased the number of people who have been deploying community networks to share internet access. For these reasons the concept of community cloud emerged, allowing a community to create and manage its own cloud infrastructure in addition to optionally sharing internet connection. This also brought up more challenging issues regarding security, privacy and networking in such shared environments.

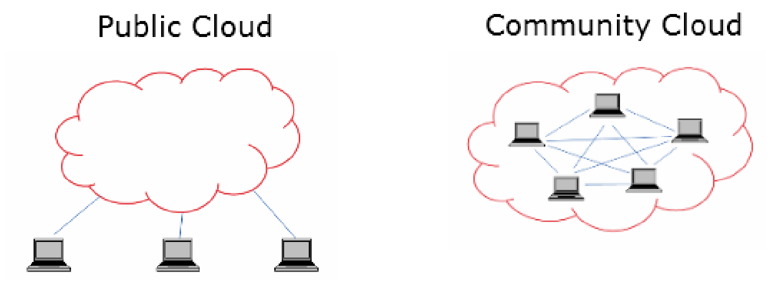


FIGURE 2.1: Architecture of Public Cloud vs Community Cloud.

In a more conceptual way, a community cloud has five main features that differentiate it from public/private clouds:

- *Openness*: It removes the dependence on vendors which makes the community cloud an open cloud, and therefore it is a dimension in the open versus proprietary discussion [13] that has emerged in many other fields such as code, standards, and data, but has not been explored in hosted services until now.

- *Community*: A community cloud is not only a technological structure,it is also a social structure made by the community of individuals.
- *Sustainability*: Because the community cloud makes use of user machines which are much more energy efficient than vendor clouds data centers it has less impact on the environment.
- *Graceful Failures*: A community cloud is not owned or controlled by any private or public organisation, and therefore it is not dependent on the failure or decisions of such. It is more robust, resilient to failures and immune to the whole-system failures compared to standard clouds providers because of the different kind of nodes and decentralized architecture. If it fails it will be non-destructively, and with low downtime because the other nodes compensate the nodes which fail.
- *Control and management*: Unlike vendor clouds, a community cloud has no problems on deciding which things can control the user and which it can not, because it is owned by the community, the decisions and management is done and decided on a democratic way.

A community cloud architecture could follow a fully distributed set of nodes where each one acts as consumer and producer, allowing the possibility of having a lot of services distributed among all the members of the community. In January 2013, a new project funded by the UE called Clommunity Project started to research about how we could create a low-powered devices community cloud.

2.2.1 Clommunity Project

The Clommunity Project is based on the idea of community networking, also known as bottom-up networking which is an emerging model for communities of citizens so they can build, operate and own their own open IP-based networks.

This project aims to build cloud infrastructure in community networks allowing groups of citizens to bootstrap, run and expand community-owned networks on the creation of a community cloud which offers services. To do this, some challenges appear to make possible the creation of a self-managing and scalable decentralized infrastructure which allows the the management and aggregation of a many and widespread low-cost networking devices, cheap storage solutions and home desktop computing resources. Also, it has to provide a distributed platform services that makes easier the design and operation of elastic, resilient and scalable services for end-users which provides good quality of experience at a low cost and sustainable way.

Furthermore, the cloud system developed by Clommunity is dynamic, therefore is resilient to the inestability of the community network, which is by nature erratic and uncertain.

2.3 Virtual Private Networks

A virtual private network establishes a secure private network across a public network. It uses Virtual Network Interfaces for the connection which are an abstract virtualized representation of a computer network interface that may or may not correspond directly to a physical network interface. Nowadays, common uses of VPNs include securing access to enterprise network resources from remote/insecure locations, connecting distributed resources from multiple sites, and establishing virtual LANs for uses such as multiplayer video games and media sharing over the Internet. In our case, in community clouds this is necessary because each node has its own public IP address and we need to create a private network to share resources and services between virtual nodes and to improve the security and privacy of the network creating tunnels among the nodes.

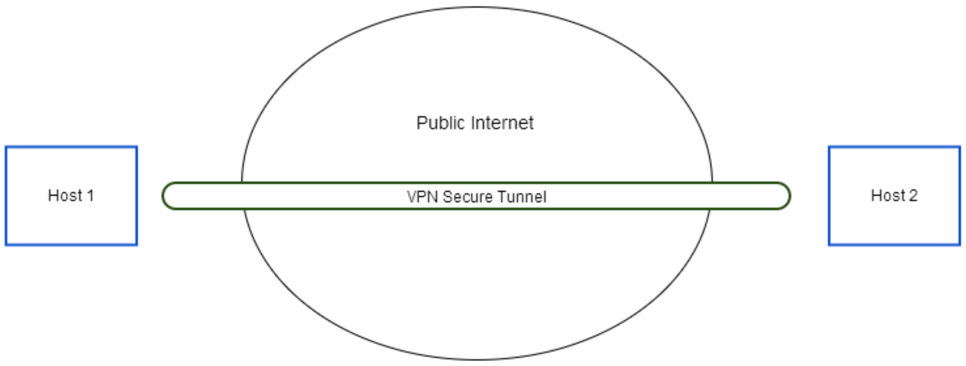


Figure 2.2: VPN Tunnel over Internet.

Usually, we separate mainly two different types of VPNs: LAN to LAN and Remote access. On LAN to LAN VPNs, a device such as router, firewall or concentrator terminates both ends of the connection. On Remote access, there is a piece of software installed on a PC/Laptop on one end and the other end would be established into a router, firewall or concentrator. Also VPNs can be distributed depending on which OSI layer they work on or whether they are centralized (e.g. OpenVPN) or decentralized (e.g. IPOP).

In general terms, a VPN works doing the following steps:

- Set up a Virtual Network Interface (VNI)
- Set up a tunnel to another computer/router
- Get any traffic coming from the VNI
- Encrypt the traffic and send it over the tunnel
- The receiver reads the information received

2.3.1 Tinc VPN

Tinc is a Virtual Private Network daemon that uses tunneling and optional encryption to create a secure private network between hosts on the Internet. Tinc is free open source software and licensed under the GNU General Public License version 2 or later. Since this VPN works at the IP level network (layer 3) as a normal network device, there is no need to adapt existing software. This allows VPN sites to share and send information with each other over the public Internet on a secure way without exposing the information to others. In addition, another features of Tinc are:

- Encryption, authentication and compression.
- Automatic mesh routing and fully decentralized management.
- Easily expandable VPN.
- Ability to bridge Ethernet segments.
- Runs on many operating systems and supports IPv6.

Tinc is currently used for the management network of Confine and in Community Networks such us FreiFunk. In the case of community clouds, the use of Tinc could be done in layer two. Some points in favor of layer two are that all the nodes are in the same collision domain and we can make use of broadcast and multicast packets, avoid NAT translation problems. Mind that service discovery software like avahi works on this layer. On the other hand, layer two VPN introduces more traffic on the network due to the Address Resolution Protocol (ARP) which continuously broadcasts packets that reach every node in the virtual network .

2.3.2 IPOP

IPOP open-source project provides a decentralized and distributed overlay virtual network. IPOP is architected as a peer-to-peer (P2P) overlay which allows tunneling and routing of encapsulated IP packets that are captured and injected into virtual network interfaces created on the nodes. This P2P networks is self-configurable and it makes use of fully decentralized DHCP requests and Distributed Hash Table, allowing user mobility, scalability and robustness.

The use of P2P routing to overlay virtual IP traffic differentiates IPOP from existing network virtualization solutions in the following properties:

- Resiliency: P2P networks are robust even under high node failure rate. IPOP overlay dynamically adapts routing of packets as nodes fail or leave the network meaning that autonomously deals with joins/leaves and failures.
- Accessibility: IPOP doesn't need a dedicated STUN or STUNT server to cross NAT or Firewalls. Each overlay node can provide the functionality to detect NATs and their subsequent transversal.
- Scalability: P2P network overlay allows a large number of nodes because routing information is naturally self-configured, decentralized, and dynamically adaptive in response to nodes joining/leaving.
- Self-optimizing: It autonomously forms 1-hop connections between nodes which frequently communicate at the virtual IP layer. It also trims on-demand edges no longer in use.

The IPOP architecture relies on a virtual network interface (tap) that captures and injects IP packets and a P2P routing substrate that encapsulates, tunnels and routes packets within the overlay. This is possible because IPOP makes use of the Brunet P2P library. Brunet library provides core services of routing, object storage and lookup, and overlay connection management. It also supports multiple protocols (UDP, TCP and tunnels) and NAT traversal.

The topology relies on a bi-directional ring ordered by 160-bit IPOP IDs with close and distant connections. See figure 2.2. some of the characteristics of this ring includes:

- Overlay edges: Multiple transports: UDP, TCP, TLS
- NAT traversal (UDP hole-punching): Greedy routing
- Deliver to peer closest to destination IPOPid: Constant number of edges per node
- On-demand edges: Created/trimmed down based on IP communication

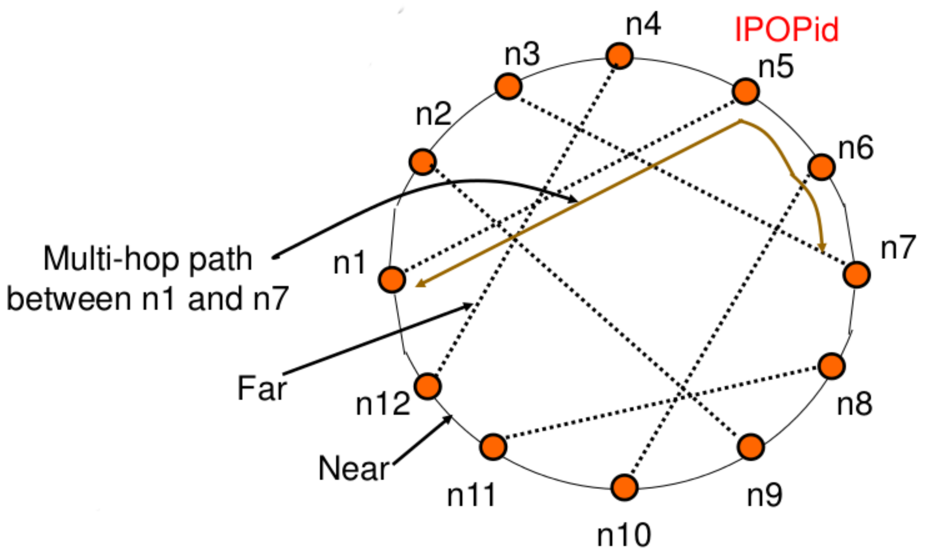


Figure 2.3: Ring topology of IPOP VPN. (source: slide 14 of "Peer-to-peer Virtual Private Networks and Applications" by Renato Figueiredo)

Another interesting property is that one P2P overlay can multiplex more than one VN by using different namespaces. This feature gives among others the community cloud the ability to create different VNs for different services or applications.

2.4 Software-Defined Networks

The installation and configurations of networks usually requires skilled people adept at configuring many network elements such as routers, switches, etc. If we have a very complex network context, an optimal configuration is difficult to achieve and changes on the network require a lot of work. To support this, a new network model was born with the idea of separate data and control plane and control the network behavior from a logically-centralized single high-level controller.

SDN focuses in 5 key features:

- Control and Data plane are decoupled
- A centralized controller that has a global view of the network which allows the creation of complex topologies (e.g. hierarchical, multi-tier architectures) that are impossible to achieve via VPN technology
- Open interfaces between the devices in the control plane (controllers) and those in the data plane
- External application can program and manage the underlying network
- In addition to providing connectivity between virtual nodes it allows full control of the network components such as routers, firewalls, subnets, dhcp server, etc.

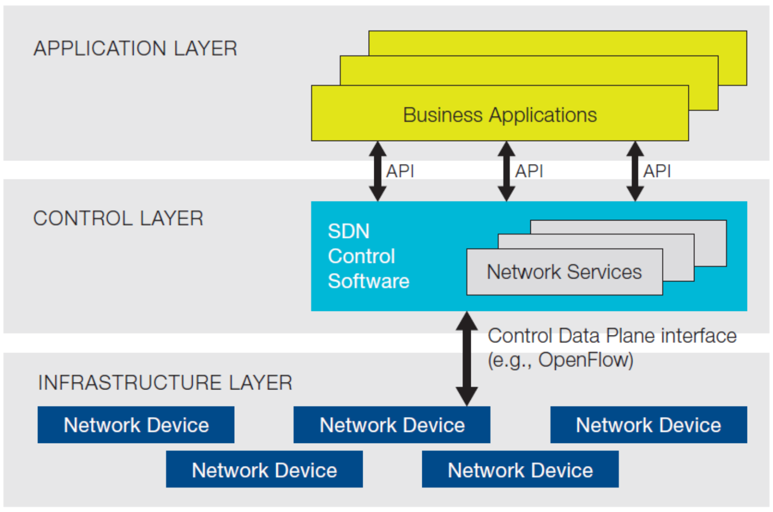


FIGURE 2.4: Software-Defined Network Architecture. (source: opennetworking.org)

The term software-defined networking (SDN) has been emerging in the last years. However, the concept behind SDN has been evolving since 1996, it started by the desire to provide user-controlled management of forwarding in network nodes [14]. Many implementations have been done by research groups and companies but the most interesting appeared in 2007: OpenFlow. SDN is not, however, limited to any one of these implementations, it is just a general term to refer to this kind of platform.

2.4.1 OpenFlow

OpenFlow enables entries in the Flow Table to be defined by a server external to the switch. It provides an API with an standard interface for programming the data plane switches and it requires an OpenFlow controller.

An advantage of using OpenFlow SDN for clouds is the topology optimization or ability to create more complex topologies, control logic is not tied to hardware and the network-wide view makes easier to infer network behavior.

Also, some challenges arise with the use of OpenFlow:

- Scalability: controller responsible for many nodes
- Security/reliability: Controller fails or its compromised
- Specialized/extra Hardware should, in some cases, be required

Chapter 3

Solutions Study

In this chapter three different approaches are shown for the creation of a Virtual Network among a set of nodes that simulate an small community cloud. First of all we analyze Tinc VPN Software and we prepare a testbed with three nodes to evaluate it. Secondly, we will use the IPOP project Virtual Network to create a testbed where we can do set of tests. Lastly, we will do a brief analysis of how we could use SDN in the scenario of community clouds.

3.1 Tinc VPN

One of the VPNs options for the community cloud is Tinc. Tinc is being used in some projects like Freifunk [6] or qMp [15]. Tinc can work either in layer 3 (router mode) or layer 2 (switch mode). Currently in the Clommunity Project it is under consideration to use a distribution service software that requires layer 2 connectivity so in this case Tinc should be configured in that layer.

Compared to other VPNs like IPOP, Tinc is more complex to configure, every node should contact their neighbors with configuration information for the public key exchange. Even if it is distributed, nodes need to know at least the public key and IP of another node to be able to connect to it. This is a barrier in the context of a community cloud where we focus on a self-configured system where the user does not have to do any, or almost any, manual configuration. To overpass this problem we can use syncTinc VPN, a client/server tool where the nodes can PULL to get the list of nodes they can connect with. A problem is that if this server fails, the network will continue working but no nodes will be able to join the existing VPN due to the server failure.

Although Tinc is supposed to add a latency of only around a millisecond, being a user space program means that the operating system's scheduler itself can add more latency if other programs are running simultaneously.

In the following subsection an scenario with three nodes interconnected with Tinc is explained in order to do a set of tests.

3.1.1 Implementation of Tinc testbed

To evaluate Tinc an scenario is defined simulating a very small community of only three nodes. To prepare this scenario we deploy 3 VMs on micro instances with Ubuntu Server 13.04 on Amazon EC2 and a VPN connection using Tinc between three hosts is established. Once virtual machines are deployed we execute a series of tests (view chapter 4) that measure the communication latency overhead of Tinc, fault tolerance and resources consumption.

First of all we launch three micro instances on Amazon EC2 where we will install and configure Tinc VPN. After, we install the Tinc software which can be currently found on Ubuntu repository, therefore, we can install it using "pat-get install Tinc".

Once we have Tinc installed on the three nodes we had to decide which node acts as a bootstrapping node so the other can connect to it the first time they join the P2P overlay network. If we name the virtual machines as VM1, VM2 and VM3; our scenario will look like the following figure:

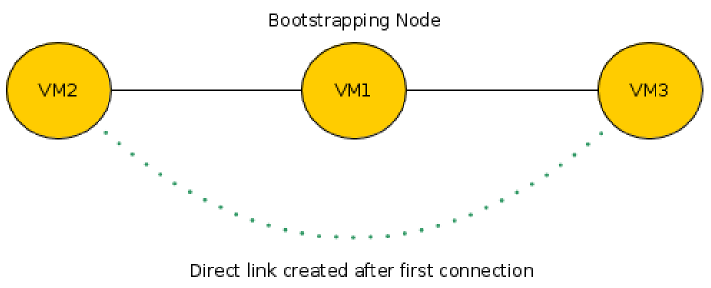


Figure 3.1: Schema of Tinc testing environment

For this testbed, synctinc server which helps keeping track of a list of nodes has not been used. For this reason we configured all nodes manually. To do so we followed these steps:

- Define a name for our vpn. In our case I called it ”myvpn”
- Create ”/etc/tinc/myvpn/hosts” and configuration file: ”/etc/tinc/myvpn/tinc.conf”. Note that the configuration is different for each node, indicating if it is the boostrapper node or a node that Connects to it. It also defines the name of the node.
- Generate private/public keys on each node
- Create scripts which configure the tun0 interface which a given private IP address assigned manually (10.0.0.0/32)
- Manual copy the public keys of VM1 and VM2 to VM3 and vice versa.
- Launch Tinc daemon. First on the bootstrapping node and then on the other two nodes.

If we don’t establish any kind of connection from VM1 to VM3 they don’t create a direct link, however the first time we ping each other they will establish a direct connection. Now that we have our testbed configured we proceed to do a set of tests that are explained in chapter 4.

3.2 IPOP

In this section we run some experiments with IPOP protocol. To simulate the scenario of a small community cloud we have defined two different cases. In the first case we have deployed four Ubuntu Server 13.04 virtual machines in Amazon EC2 [16] to simulate a small cloud. Once virtual machines are deployed we execute a series of tests that measure the communication latency overhead of IPOP, fault-tolerance, JOIN/LEAVE of nodes and resources consumption.

3.2.1 Implementation of IPOP testbed

The purposes of these tests is to find out if IPOP is a VPN solution that satisfies the requirements we have in community clouds mentioned in chapter 1.

To accomplish this, one scenario is defined to simulate a small community cloud geographically in the same location with four VMs which deploy 19 virtual nodes. To do so, we deploy 4 VMs on Amazon EC2. More precisely, we have three VMs with one virtual node, and 1 VM with 16 virtual nodes. IPOP lets you deploy more than one virtual node for testing purposes, therefore to simulate a bigger community we deploy 16 virtual extra virtual nodes that will join the overlay network, having a total of 19 simulated community members.

The first step is to deploy the virtual machines which will be connected by the IPOP virtual network. IPOP runs inside VMs and the main requirement is that VMs should have Internet access. A VM can have a public IP address, or a private one that connects to the Internet through NAT. Only the P2P network overlay bootstrap node should have a public IP address.

In our case have used Amazon EC2 to deploy four virtual machines with Ubuntu Server 13.04 distribution in Amazon micro-instances. These instances have low memory and CPU resources but it is enough to do the tests.

Once we have deployed the virtual machines we have to take into consideration that one VM (VM1) is going to act as the P2P overlay boostrapper and also have 16 virtual nodes, and the other three only as ”individual community cloud users”. To do so, first of all we installed the IPOP package to these virtual machines.

The second step was to bring up IPOP in different Virtual Machines. First of all we configured VM1 as the P2P overlay bootstrap machine. To do so we have a script that helps with the configuration. (see Appendix A)

- Install IPOP package
- Create configuration file bootstrap.conf
- Launch IPOP bootstrap node

In this case on the bootstrap node we can simulate 16 virtual nodes at the same time to have more links and edges. Therefore, these 16 nodes plus the other 3 nodes on the other VMs will make in total 19 virtual nodes deployed.

Next step was to configure the three nodes that were going to be connected to the P2P overlay.

- Install IPOP package
- Create configuration file bootstrap.conf
- Launch IPOP

Lastly we had our first scenario configured like this:

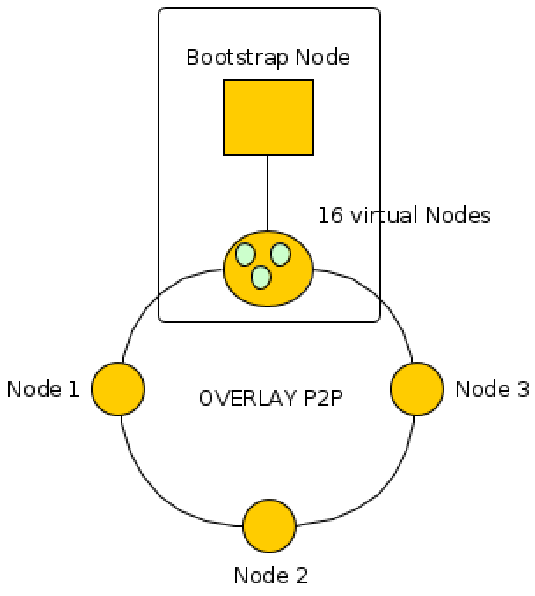
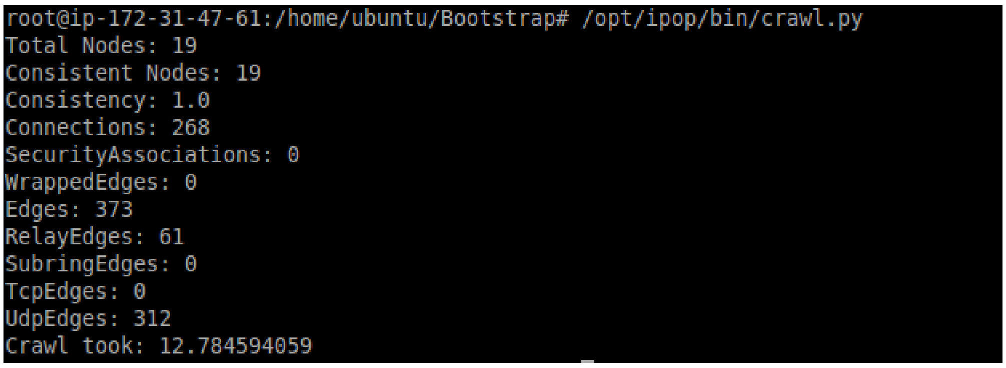


Figure 3.2: Schema of IPOP testing environment

IPOP comes with a tool to crawl the network via the Bootstrap node. If we use it we could see the following:



```
root@ip-172-31-47-61:/home/ubuntu/Bootstrap# /opt/ipop/bin/crawl.py
Total Nodes: 19
Consistent Nodes: 19
Consistency: 1.0
Connections: 268
SecurityAssociations: 0
WrappedEdges: 0
Edges: 373
RelayEdges: 61
SubringEdges: 0
TcpEdges: 0
UdpEdges: 312
Crawl took: 12.784594059
```

FIGURE 3.3: Screenshot of the network crawling tool included in IPOP.

In this figure we can see that the number of Nodes is 19, 16 virtualized nodes on the Bootstrap and 3 virtual nodes on each of the other three Amazon VMs. Also the consistency let us know when a node is failing or not responding. Another information that we can observe is SecurityAssociations which in this case is 0 because we haven't configured any kind of secure communication protocol like IPsec, providing privacy, integrity, and authentication at the IP layer. Also, we can see that we are using UDP and not TCP looking at the UdpEdges metric.

Now that we had our first scenario configured, we could proceed with the tests. In the next chapter we can find the tests detailed.

3.3 SDN

This approach is done only in a conceptual way due to lack of time and requirements about how to run SDN software.

In the first instance, the chosen software to run this would be OpenFlow. The main advantage of SDN as explained in chapter two is to separate Data plane and Control plane entities, doing this you don't just only control a networking device but an entire network. With OpenFlow you get an open API that provides standard interfaces for programming the data plane switches. One of the differences if we would try to implement OpenFlow in a community cloud would be the need of having extra hardware such as an Ethernet switch where we can modify the flow tables and a machine to install the main controller of the network [17].

After researching more SDN solutions another option would be an approach using Open vSwitch [14] which creates a virtual switch that enables massive network automation through programmatic extension, while supporting standard management interfaces and protocols (e.g. NetFlow, SPAN, CLI, 802.1ag, LACP). It can act as a virtual software switch in the virtual environment and provides different features for the network management such as VLAN, flow control or QoS .

3.4 Feature comparison table

Upon analyzing the features of all three solutions presented in this section we can provide a table mainly for feature comparison reasons.

	Tinc	IPOP	SDN
IPv6 Support	Yes	Under Development	Yes
Topology Optimization	No	Yes	Yes
Scalability	1000 nodes	+10000 nodes	Yes (via controller load balancing
Failure Recover	No	Yes	Yes
Encryption	Optional	Optional - with IPsec	Yes (supports any protocol)
Nat Transversal	Manual cfg	Yes - UDP hole punching	Yes (FWaaS support on SDN)
Single point of failure	No	No	No (if there is HA on controller)

On the next chapter, a detailed explanation of the tests done with Tinc and IPOP is found.

Chapter 4

Tinc & IPOP Evaluation

4.1 Tinc Tests

On this section the tests done with Tinc are explained. These tests include latency added, join/leave/fialure management and resource consumption. One thing that left unfeasable would be the test of traffic going through each node. This would provide more interesting data to evaluate.

4.1.0.1 Latency added

The first test consist of seeing the latency added of Tinc VPN software when pinging another node of the testbed. To compare the ping tool provided by Linux has been used. The ping was done every 1 second for a period of 30 seconds. The packet size is of 84 bytes (20 bytes of IP header + 8 bytes of ICMP header + 56 bytes of data). We will do three different test cases. First we will ping two nodes using the public IP address of the node, secondly using the Amazon private IP address and finally using the Tinc VPN private address. For each different case we get the minimum latency, the average, the maximum latency and the standard deviation,which is an average of how far each ping is from the mean RTT (round-trip time).

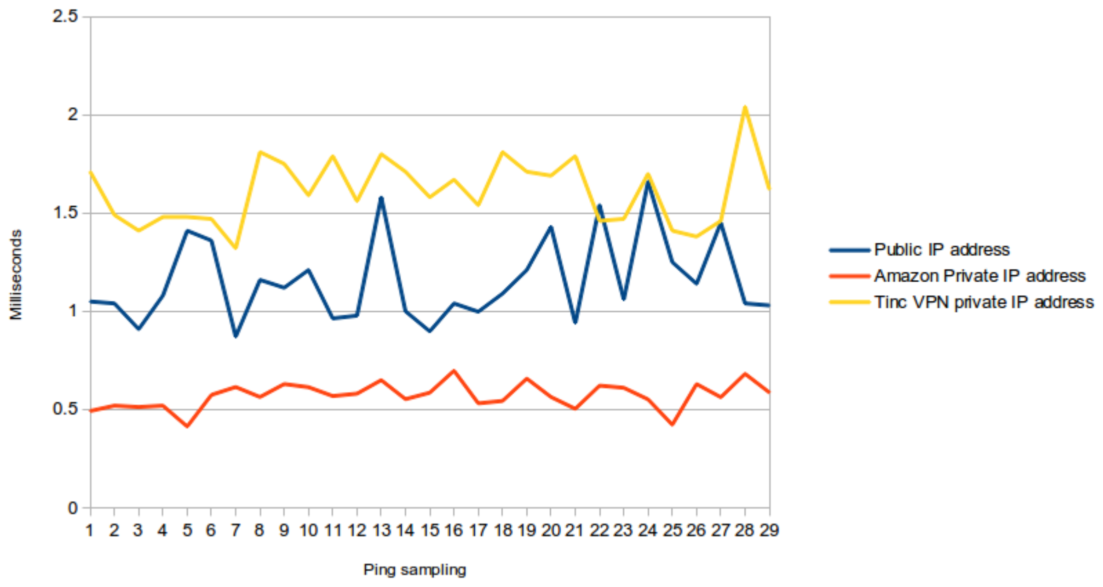


FIGURE 4.1: Tinc evaluation: Graph showing different ping times in milliseconds for a period of 30 seconds.

In the previous figure we can see a graph with the comparison of the ping times between the different IP addresses used. We can observe three different behaviors. First of all, the amazon private address is very stable and with the lowest latency. Secondly we can see how the public IP address ping has strange peaks which even if we don't know the cause, could be due to the network configuration on Amazon EC2 because the peaks looks like a cache refreshing every 3-5 seconds. Lastly the ping times of Tinc are the highest and not very stable. With the realization of these tests an average latency added of 1 ms has been observed using tinc vpn instead of the standard Amazon private network. The minimum latency, average latency, maximum latency and standard deviation can be observed in the following table:

Address	Latency min	Latency avg	Latency max	Deviation
Public IP	0.871 ms	1.154 ms	1.664 ms	0.214 ms
Amazon Private IP	0.414 ms	0.572 ms	0.698 ms	0.069 ms
Tinc Overlay Private IP	1.327 ms	1.617 ms	2.042 ms	0.169 ms

4.1.0.2 Fault tolerance

In Tinc fault-tolerance has to be managed manually on each node because we don't have a global view of the network. If a node fails, the rest of the overlay keeps working but this node has to be reseted/reconnected manually. In this test two things have been proven. The first one is to shut down VM2, and the second one to shutdown VM1, which is the bootstrapping node of this small overlay. The results shown that when shutting down a standard node everything still working while when shutting down the bootstrapping node makes the other two nodes loose connection.

4.1.0.3 Resource consumption

Having initialized Tinc in the three nodes and sending traffic to all of them using "ping" tool, the CPU usage was stable at 0.3% and the memory usage close to 0 Mb. Taking in consideration that the experiments are running in a low-resources machine this means that Tinc is lightweight and doesn't cause any heavy load on the CPU or high consumption of memory which means that could easily run on end-user devices without affecting other services running.

4.2 IPOP Tests

On this section the tests done with IPOP are explained. These tests include latency added, join/leave/fialure management and resource consumption.

4.2.0.4 Latency added

The idea of this test is to compare the latency between two nodes using their private IP address provided by Amazon, their virtual IP address deployed with IPOP and their public IP address. On the first instance we will use the public IP address assigned by Amazon. Secondly we will use the private IP of the Amazon network, and lastly we will use the private IP in the P2P overlay network created by IPOP. The tests have been done using Linux ping tool sending a total of 30 packets of 84 bytes (20 bytes of IP header + 8 bytes of ICMP header + 56 bytes of data) every 1 second during a period of 30 seconds. Of each test we get the minimum latency, the average, the maximum and the standard deviation, essentially an average of how far each ping RTT is from the mean RTT.

Ping test times comparison

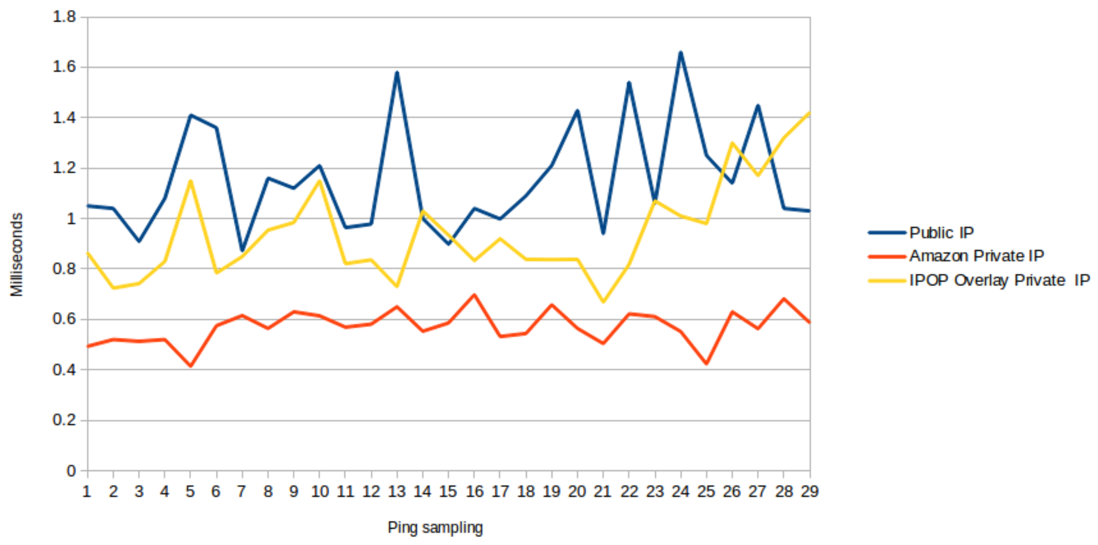


FIGURE 4.2: IPOP evaluation: Graph showing different ping times in milliseconds for a period of 30 seconds.

As we can see on the previous image ping times for the public IP address and IPOP overlay network addresses are not very stable, this could be due to VM location on Amazon Cloud which could cause this peaks behavior every 3-5 seconds.

If we put all the values together on a table and we calculate the average and the deviation:

Address	Latency min	Latency avg	Latency max	Deviation
Public IP	0.871 ms	1.154 ms	1.664 ms	0.214 ms
Amazon Private IP	0.414 ms	0.572 ms	0.698 ms	0.069 ms
IPOP Overlay Private IP	0.668 ms	0.943 ms	1.423 ms	0.188 ms

As we can see the overhead added by IPOP is quite low, IPOP overlay ping is an average of 0.4 ms slower if we compare it to the Amazon private IP address test. A factor that could affect this value is the reduced number of hops that we have between the nodes of this experiment. On an scenario with more nodes probably it would be a little higher but IPOP is configured to optimize links between nodes to reduce the number of hops.

Either way, this delay is deprecated for most services that we could be using/offering in a community cloud.

4.2.0.5 Fault tolerance

In this part the purpose is to check how IPOP manages failures of nodes and how is the response of them. In a real case scenario where users are connected to the community cloud many things can go wrong and we need to ensure that our VPN knows how to manage such failures. To try to emulate different failures we will kill the connection of the node in two ways:

- Shutting down the ipop interface
- Shutting down the machine

In the first case, if you shut down the tapipop interface simulating a failure, IPOP overlay detects that than node is failing but it does not drop it of the list, however it appears as off-line. On the other hand, if we shut down the computer the bootstrap node detects that that node is OFF and after a period of 10 seconds that node is dropped of the list of nodes.

The worst failure that may occur is the failure of the bootstrapping node shutting down the whole overlay network. If this happens all the overlay stop working and the nodes have to reconnect to the bootsrapping node as soon as this is up again. Assuming that the public IP address of this node has not changed the reconnection is done automatically. A possible option to avoid this failure is to replicate the bootstrapping server and have more than one across the network, but this option is not being studied on this thesis.

4.2.0.6 Join/Leave

Assuming that we have the bootstrapping node initialized the connection of one node to the overlay takes a few seconds. We need to know the public IP address and namespace of that overlay and establish connection.

If we want to leave the overlay we have just have to stop the tapipop interface manually or stopping it using the proper script.

4.2.0.7 Resources Consumption

An important aspect to take into account is the resource consumption of the VPN software we will use. In some cases users have a low resources device to be part of the community cloud, because of this we have to focus on a lightweight and low-resources demanding VPN software. We should check approximately the amount of memory it uses and see if the CPU usage is notable or not.

In this case with IPOP, because it is only a small daemon running on the machine. After checking the memory usage on the VMs I can say that is not going to affect the global performance being the memory consumption of this, less than 2 MB. Regarding CPU consumption was less than 1% all the time.

4.3 Evaluation Conclusions

Having done tests with both solutions we can compare the results obtained.

Firstly, regarding latency added using IPOP or Tinc the difference is very small but the tests done in this experiments show that Tinc has more overhead than IPOP. In Tinc, we have a theoretical added latency of some milliseconds while the latency observed in the tests with IPOP is less than 1 ms which is almost negligible and will not affect the functionality of the services of the cloud. On the other hand, Tinc has proven to have more stable times and a lower standard deviation.

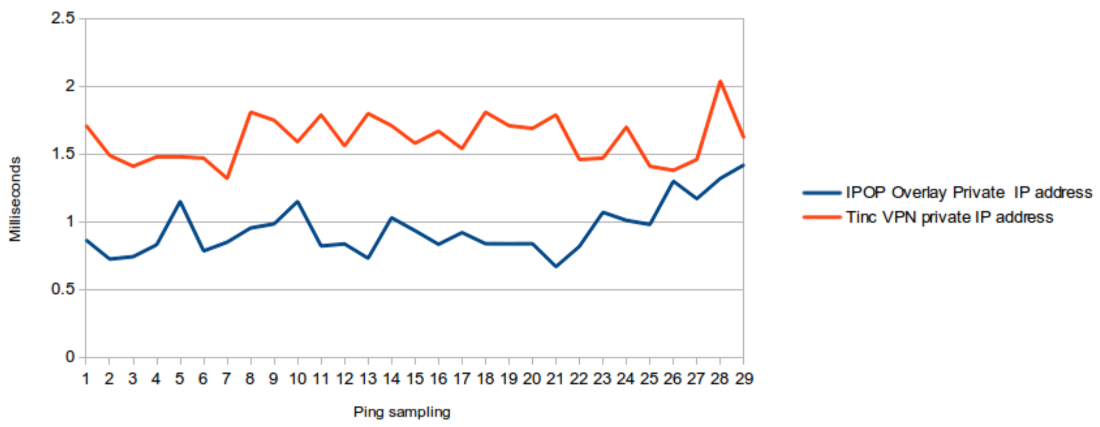


FIGURE 4.3: Tinc and IPOP ping times.

Address	Latency min	Latency avg	Latency max	Deviation
Tinc Overlay Private IP	1.327 ms	1.617 ms	2.042 ms	0.169 ms
IPOP Overlay Private IP	0.668 ms	0.943 ms	1.423 ms	0.188 ms

Secondly, on the fault tolerance tests both solutions need manual management of failures, the difference is that with IPOP and the crawling tool provided we can see how the topology knows about the failures and drop the node of the list so it can avoid routing packets through that node.

Also, checking the resources consumption of both solution show that none of both could be considered as ”high demanding” software. Therefore we can say that both solutions are lightweight and a good option for low-end machines.

Another important thing that has not been tested but has been researched is how these solutions manage NAT transversal. In the case of Tinc, this uses publicly addressable servers to help route traffic whereas IPOP uses Google libjingle. Using this library, it can have lower latencies and higher bandwidth because traffic is not routed directly. also, IPOP performs NAT hole-punching automatically and transparently to the user and with Tinc you need access to NAT/firewall devices to configure them properly.

In terms of scalabilty, according to the official website, with Tinc VPN we have a theoretical maximum of number of nodes which is limited to 1000, however, with IPOP we can find existing overlay networks like PlanetLab [18] which has currently more than 1000 nodes but its developer says that it can handle a few thousand nodes. According to the developer of IPOP, the scalability of IPOP is limited due to the NAT transversal protocol to an approximate number of +10000 nodes. This amount of nodes fits my idea of community cloud for a small/medium community but could be a problem in a near future to expand the number of users.

Chapter 5

Conclusions and future work

5.1 Conclusions

The following conclusions come out after the study of three main VPN technologies: Tinc, IPOP and SDN and the comparison of Tinc and IPOP. Firstly we have to take into consideration that we can focus on having the best results in scalability, robustness and easiness on configuration, being, if it is possible, to find the best all-around solution. Secondly, another important aspect when we are deciding which technology fits better is which is more deprecated or which solution has more ongoing projects and is still under development/improvement.

After this considerations and the development of the tests I came to the conclusion that the solution that seems to fit better for the community cloud virtual private network would be to use IPOP and an initialization script/server with DHCP functionalities for the following reasons.

First of all, even though SDN is the newest and more powerful technology it is more complicated to install and it is in most cases hardware dependent, also, it is not fully decentralized because we need the main controller that manages the virtual network and usually requires specific compatible hardware. Also, due to being a solution with more features than the others it requires more complex configuration. On the other hand, IPOP is a relatively new technology which satisfies all the requirements of the community cloud VPN and does not need any specific hardware compatible. It is more flexible in this sense.

Furthermore, the easiness of configuration of IPOP, its dynamic topology optimizations and the fact that being a newer technology which has more active projects under development and patches/bug fixes makes me decide in favor of this one instead of Tinc. Tinc and IPOP could be considered similar as both solutions create a P2P overlay network and rely on an external server at some point, in the case of Tinc it is to be able to provide the public keys of the nodes and keep a list of all the nodes and in the case of IPOP to create and manage the overlay network. Also, comparing another important aspect to take into consideration due to the dynamism of community network and end-users IPOP manages better the failure of nodes and reconnection process.

Regarding latency added using IPOP or Tinc the difference is very small but the tests done in this thesis show that Tinc has more overhead than IPOP. In Tinc, we have a theoretical added latency of a few milliseconds while the latency observed in the tests with IPOP is an average of approximately 1 ms which is almost negligible and will not affect the functionality of the services of the cloud.

Also, another advantage of IPOP over Tinc is that IPOP performs NAT-transversal automatically and in a transparent way for the user while with Tinc you have to manually configure NAT/firewall rules.

Lastly, another reason that makes me decide in favor of IPOP is the fact that IPOP is proven to be more scalable, with Tinc VPN we have a theoretical maximum of number of nodes which is limited to approximately 1000, however, with IPOP we can find existing overlay networks like PlanetLab [18] which has a few thousand nodes. According to the developer of IPOP, the scalability of IPOP is limited due to the NAT transversal protocol to an approximate number of more than 10000 nodes. This amount of nodes fits my idea of community cloud for a small/medium community but could be a problem in a near future to expand the number of users.

Lastly, another advantage of IPOP over Tinc is that IPOP performs NAT-transversal automatically and in a transparent way for the user while with Tinc you have to manually configure NAT/firewall rules.

5.2 Future Work

Due to the lack of time and in some cases resources, there are a lot of things which I would like to try in a near future, being the last step to fully integrate one of these VPNs in the Clommunity distribution and test it over a community network.

5.2.1 Evaluation on bigger scenario

As a future work, I would like to do a configuration for IPOP with a larger set of nodes (around 50 virtual nodes) geographically spread to make the context more realistic. Furthermore, currently, the private IP address of each node of the overlay network has to be added manually as well as the namespace, an option would be to modify the bootstrap node source code to automatically assign private IP of a specific range when a node wants to connect to the network doing the first-time connection transparent to the end user.

Also, the possibility of using Tinc packet for Clommunity Distribution which is under development by members of Confine Project is a big advantage and could be used to compare face to face Tinc and IPOP in a real case scenario in terms of performance and also, for example, on doing more tests such as resources consumption or compatibility with service discovery software.

5.2.2 Configuration Server in Tinc

The original code of Tinc VPN makes it difficult to use it on a community cloud due to the user has to know the IP of another node and they have to manually configure both to be able to exchange the public keys and establish a connection to access the p2p vpn. To solve this problem an idea of creating a configuration server with public IP address that is responsible of doing this exchange appeared and was developed by members of guifi.net. However its functionality is limited and many improvements can be made.

At the beginning the configuration server was a package that you could install and it was able to receive polls from the nodes to get a list of all the other nodes which it can connect to so they can establish connection with their neighbors.

The new version which is under development will include a web interface where a user could select which Tinc network it wants to create/connect to and then download a .deb package which includes all the required configuration. This is a good approach because the user does not need to have high knowledge about how the network works. The main problem of relying on one server persists and one of the solution could be to make use of some decentralized database with unique identifiers to synchronize other nodes information among all the nodes set. Another feature which is not implemented and could be beneficial is the support for IPv6 which would allow multicasting which can help on finding bootstrappers.

5.2.3 IPOP bootstrapping node improvements

IPOP needs a server which helps the first-time configuration of the node and creating the P2P overlay network, as well as managing the topology, crawling the network and optimizing links among nodes. The problem is that with the basic deployment scripts the node has to provide a manual private IP to be used on the Overlay Network and the namespace that it wants to use.

To make the configuration easier we can implement a functionality on the bootstrapping node that works as a DHCP server assigning privates IPs of a previously configured range. IP addresses can be assigned to nodes through the use of a DHT by performing a lookup to see if the key already exists, if so a different IP is looked up until an unallocated IP is found. Doing such a thing the user would only need to provide the name of namespace.

Appendix A

IPOP Deployment Scripts

SCRIPTS TO DEPLOY IPOP.

Bootsrap prepare.

```
#!/bin/sh

if [ $# -ne 3 ]; then
  echo "Usage: $0 P2PNamespace BootstrapIP BootstrapPort"
  echo "  Where P2PNamespace is a unique string you should choose for your P2P network,"
  echo "  BootstrapIP is the public IP address of this machine"
  echo "  and BootstrapPort is the port you would like to run the P2P bootstrap node"
  exit;
fi

echo "-----------------------------------------------------------------------------"
echo "Configuring P2P node:namespace $1 IP address:port $2:$3"
echo "-----------------------------------------------------------------------------"

sed -i "s/BRUNET_NAMESPACE/$1/g" bootstrap.config
sed -i "s/BOOTSTRAP_IP/$2/g" bootstrap.config
sed -i "s/BOOTSTRAP_PORT/$3/g" bootstrap.config
```

Run Bootstrap Node or node of the Overlay.

```
#!/bin/sh

export MONO_PATH=/opt/ipop/lib

if [ $# -ne 1 ]; then
  echo "Usage: $0 BootstrapConfFile"
  echo "  Where BootstrapConfFile is the name of the bootstrap configuration file"
  exit;
fi

echo "------------------------------------------------------------------------------"
echo "Downloading IPOP bootstrap code and mono"
echo "------------------------------------------------------------------------------"

echo "deb http://www.grid-appliance.org/files/packages/deb/ stable contrib" >> /etc/apt/sources.
wget http://www.grid-appliance.org/files/packages/deb/repo.key
apt-key add repo.key
apt-get update
apt-get -y install zip
apt-get -y install ipop


echo "------------------------------------------------------------------------------"
echo "Starting up bootstrap node"
echo "------------------------------------------------------------------------------"

nohup mono /opt/ipop/bin/P2PNode.exe -n $1 -c 16 &
```

Bibliography

[1] Guifi.net, documentació de guifi.net, 2008. URL http://guifi.net/ca/CADocs.

[2] M. Gall, A. Schneider, and N. Fallenbeck. An architecture for community clouds using concepts of the intercloud. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 74–81, 2013. doi: 10.1109/AINA.2013.42.

[3] Openvpn, 2014. URL http://openvpn.net.

[4] Freelan, a vpn client, done it right., 2014. URL http://www.freelan.org/.

[5] Fpeervpn - the open source peer-to-peer vpn, 2014. URL http://www.peervpn.net/.

[6] Freifunk wiki, tinc vpn, 2013. URL http://wiki.freifunk.net/Tinc.

[7] Ipop (ip over p2p) opensource project, 2013. URL http://www.grid-appliance.org/wiki/index.php/IPOP.

[8] Clommunity project, a community network cloud in a box., 2013. URL http://clommunity-project.eu/.

[9] D. Vega, L. Cerda-Alabern, L. Navarro, and R. Meseguer. Topology patterns of a community network: Guifi.net. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 IEEE 8th International Conference on*, pages 612–619, 2012. doi: 10.1109/WiMOB.2012.6379139.

[10] D.I. Wolinsky, Kyungyong Lee, P.O. Boykin, and R. Figueiredo. On the design of autonomic, decentralized vpns. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2010 6th International Conference on*, pages 1–10, 2010.

[11] G. Pierre and C. Stratan. Conpaas: A platform for hosting elastic cloud applications. *Internet Computing, IEEE*, 16(5):88–92, 2012. ISSN 1089-7801. doi: 10.1109/MIC.2012.105.

[12] G. Briscoe and A. Marinos. Digital ecosystems in the clouds: Towards community cloud computing. In *Digital Ecosystems and Technologies, 2009. DEST '09. 3rd IEEE International Conference on*, pages 103–108, 2009. doi: 10.1109/DEST.2009.5276725.

[13] J. West and J. Dedrick. Proprietary vs. open standards in the network era: an examination of the linux phenomenon. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*, pages 10 pp.–, 2001. doi: 10.1109/HICSS.2001.926525.

[14] S. Sezer, S. Scott-Hayward, P.K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. Are we ready for sdn? implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7):36–43, 2013. ISSN 0163-6804. doi: 10.1109/MCOM.2013.6553676.

[15] Quick mesh project - wiki, 2013. URL `http://dev.qmp.cat/projects/qmp/wiki/Start`.

[16] Amazon elastic compute cloud (amazon ec2), information, 2014. URL `http://aws.amazon.com/es/ec2/`.

[17] K. Bakshi. Considerations for software defined networking (sdn): Approaches and use cases. In *Aerospace Conference, 2013 IEEE*, pages 1–9, 2013. doi: 10.1109/AERO.2013.6496914.

[18] Planetlab - an open platform for developing, deploying, and accessing planetary-scale services, 2014. URL `https://www.planet-lab.org/`.