Francois D'Ugard



MISSION CRITICAL CLOUD COMPUTING

Design Document

# Copyright Notice

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation (THE SOFTWARE). THE SOFTWARE is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.  
  
THE SOFTWARE is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. In no event shall the authors be liable for any claim, damages, or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software. See the GNU General Public License for more details.  
  
You should have received a copy of the GNU General Public License along with this software. If not, see http://www.gnu.org/licenses.

# Abstract

Virtualization is an increasingly popular approach to manage rising information technology costs and complexity in every sector of the economy. Cloud computing allows organizations of any size to provision infrastructure resources as needed and flexibly scale technology resources to meet changing demands. Cloud providers pool hardware resources and allocate them based on the requests of their users. In order to efficiently allocate these resources providers must aggregate users of different requirements and workloads onto the same physical infrastructure. However, this approach increases the likelihood that a malicious user can collocate a VM alongside a target VM in order to extract information or disrupt its functioning in some way.

We propose a solution that can deliver mission assurance to mission-critical applications in cloud computing systems. We will do so by leveraging the unique capabilities of virtualization technology that and develop a dynamic and distributed approach to run applications with good security and reliability in typical cloud computing systems.

Our approach relies on developing a complete network graph on virtual private network peer to peer connections. With the purpose of reducing the likelihood of a malicious VM locating the mission-critical VM and achieving co-residency with it.

This project will deliver mission assurance to mission-critical applications in cloud computing systems. We will do so by leveraging the unique capabilities that develop a virtual machine based approach to run applications with good security and reliability in typical cloud computing systems. This project will build upon the previous project's results namely a P2P overlay network that interconnect the OpenStack VMs based on the IP-over-P2P (IPOP) framework. The project will focus on developing an extension to IPOP that will allow for the communications among the VMs to be routed by an overlay network in an OpenStack-based cloud system.

In this document we discuss and evaluate the design decisions made with regard to the implementation of our projects. We provide a written detailed description of the software product.

Table of Contents

[Copyright Notice 1](#_Toc406065098)

[Abstract 2](#_Toc406065099)

[1. Introduction 5](#_Toc406065100)

[1.1. Problem Definition 6](#_Toc406065101)

[1.2. Design Methodology 7](#_Toc406065102)

[1.3. Definitions, acronyms, and abbreviations 8](#_Toc406065103)

[1.4. Overview of Document 9](#_Toc406065104)

[2. System Design 10](#_Toc406065105)

[2.1. Overview 10](#_Toc406065106)

[2.2. Subsystem Decomposition 11](#_Toc406065107)

[2.3. Hardware and Software Mapping 12](#_Toc406065108)

[2.4. Persistent Data Management 14](#_Toc406065109)

[2.5. Security and Privacy 15](#_Toc406065110)

[3. Detailed Design 16](#_Toc406065111)

[3.1. Overview 16](#_Toc406065112)

[3.2. Static model 16](#_Toc406065113)

[3.3. Dynamic model 18](#_Toc406065114)

[3.4. Code Specification 24](#_Toc406065115)

[4. Glossary 25](#_Toc406065116)

[5. Appendix 27](#_Toc406065117)

[5.1. Appendix A – Use Case Diagram 27](#_Toc406065118)

[5.2. Appendix B – Implemented Used Cases 28](#_Toc406065119)

[**Use Case – Configure XMPP Server.** 28](#_Toc406065120)

[**Use Case – Configure VPN Node.** 30](#_Toc406065121)

[**Use Case – Join VPN.** 32](#_Toc406065122)

[**Use Case – Leave VPN.** 34](#_Toc406065123)

[**Use Case – Start XMPP Server.** 36](#_Toc406065124)

[**Use Case – Stop XMPP Server.** 38](#_Toc406065125)

[**Use Case – Restart XMPP Server.** 40](#_Toc406065126)

[5.3. Appendix C – Detailed Class Diagram 42](#_Toc406065127)

[5.4. Appendix D – Diary of Meetings an Tasks 43](#_Toc406065128)

[6. References 44](#_Toc406065129)

# Introduction

Virtualization is an increasingly popular approach to manage rising information technology costs and complexity in every sector of the economy. Cloud computing allows organizations of any size to provision infrastructure resources as needed and flexibly scale technology resources to meet changing demands. Cloud providers like Amazon, Microsoft, or Rackspace pool hardware resources such as compute, memory, and storage and allocate them based on the provisioning requests of their users and available resources. In order to efficiently allocate these resources providers must necessarily aggregate users of different requirements and workloads onto the same physical infrastructure.

The rising popularity and diffusion of hardware virtualization technology among organizations of every size has led researchers and technology professionals to seriously consider the security of such computer systems. By allocating the virtual machines of different user onto the same infrastructure various security vulnerabilities are introduced. However many of the unique capabilities of cloud computing can also be used to increase the reliability of the cloud as well as ensure its security and resilience to an attack.

An important quality of virtualization is the ability for users to virtualize the underlying communications network. This capability lends itself well to the concept of network masquerading wherein the behavior and routing path of packets is managed in order to disguise the flow of network traffic within the network. This capability can be used to ensure the security of a cloud computing environment.

This chapter presents the problem definition in section 1.1 followed by a description of the scope of the system in section 1.2. Section 1.3 presents a list of common abbreviations and acronyms that are in use throughout the document. We present the overview of this document in section 1.4.

## Problem Definition

The rising popularity of cloud computing concepts has introduced significant security vulnerabilities into the information technology infrastructure of many organizations. Currently virtualized information technology systems are designed and built to operate on relatively static configurations. Typically cloud services providers fulfill provisioning requests on an as needed basis. In order for cloud computing to remain cost effective infrastructure providers must provision pools of resources, such as CPU, memory, and storage among many different users.

This configuration means that a malicious user can reside alongside a virtual machine that computes mission critical or sensitive data. The sharing of physical infrastructure among virtual machines belonging to different users opens up the possibility of a side-channel attack. A side channel attack occurs when a malicious users is able to locate a target virtual machine and spawn another virtual machine alongside this target. The malicious user is then able to extract information by monitoring the hardware resources shared by both virtual machines.

## Design Methodology

We propose a distributed peer to peer virtual private network that maintains the communication network structure of collaborating virtual machines unchanged allowing a decoupling between networking services within the cloud framework and the communications network used by the virtual machines. In order to increase the resiliency of said network we propose a system that masks the communication between the virtual machines, obfuscating the communication patterns of virtual machines. Our system will be leveraging and extending IP over P2P (IPOP), a virtual network software that allows users to create virtual private networks (VPNs). IPOP will be extended such that it will allow users to join a VPN with all of the VMs of their group, thus establishing a complete network graph. This detail is important since it is this capability that allows us to randomly generate any routing path of any size. This will entail functionality to join the VPN, leave it, and relay information correctly between VMs.

Our system should be easy to understand, setup, and use. This will require developing a command-line tool to access the system’s features as well as implementing the service so that it integrates well among existing OpenStack modules.

## Definitions, acronyms, and abbreviations

API: Application Programming Interface

DB: Database

IP Address: Internet Protocol Address

IPOP: IP over P2P

LAN: Local Area Network

MTD: Moving-Target Defense

OS: Operating System

P2P network: Peer-to-peer network

RDBMS: Relational Database Management System

SSHL Secure Shell

UI: User Interface

UML: Unified Modeling language

VM: Virtual Machine

VPN: Virtual Private Network

XMPP: Extensible Messaging and Presence Protocol

## Overview of Document

Chapter 1 of this document serves as an introduction to the system. We start by introducing the problem in section 1.1 followed by an explanation of the scope of the proposed system in section 1.2. In section 1.3, we give a list of all acronyms and abbreviations used throughout this document.

In Chapter 2, we give a detailed overview of the system design as well as the system decomposition into subsystems. In section 2.1 we present the major subsystems and give a short explanation on their function, while in section 2.2 we delve into the details of all the subsystems present. In section 2.3, we give a mapping of the above subsystems to hardware, where applicable, while in section 2.4 we identify the data that must be persistent throughout the operation of the system.

In Chapter 3, we present a detailed overview of the system components presented in the previous chapter. In section 3.1, we introduce the behavior and structure of each subsystem that make up the proposed system. In section 3.2 we provide a detailed description of the design of each of the major subsystems while in section 3.3 we describe in detail the evaluation of the system design. Finally, in section 3.4 we present the code specification for the subsystems.

In Chapter 4, we provide a glossary of terms used in this document.

Chapter 5 contains all the appendices referred to throughout the document. In Appendix A we present the use case diagrams for the use cases that we are implementing. In Appendix B we present the use cases from the Requirements Document that we are implementing. Finally, in Appendix C we present the diary of meetings and tasks during development.

Finally, in Chapter 6 we include the bibliographic references that we cite throughout the document.

# System Design

In this chapter we give an overview of the system by presenting a package diagram of its major subsystems, identifying the architectural patterns used in the system and justifying their selection in section 2.1. In section 2.2, we describe the major subsystems in detail and identify the requirements associated with each subsystem. We present the hardware and software mapping for the system, including a deployment diagram, in section 2.3. In section 2.4, we describe the persistent data managed by the system.

## Overview

As shown in the figure, the architectural patterns of the system are the three-tier and the peer to peer pattern. The reasons for selecting these patterns are the following:

Three-tier: The three-tier architecture pattern partitions an application into three tiers of functionality that may or may not correspond to distinct physical locations. The most relevant feature of this pattern is that it enables flexibility and reusability of the different logical layers of the application. Each layer can evolve independently to incorporate better implementation techniques and algorithms to better fit the evolving system requirements. The two main tools upon which we are building this system as required by our client, OpenStack and IPOP, are built as layered applications to reap the benefits described above.

Peer to Peer: A Peer-to-peer (P2P) network pattern is a distributed software architecture that partitions network workloads between peers. Our system relies on peers making a portion of their network bandwidth, directly available to other authorized network participants. Participants in our VPN network are both suppliers and consumers of network bandwidth. Each member contributes processing power in the form of random path generation or forwarding when required to do so.

## Subsystem Decomposition

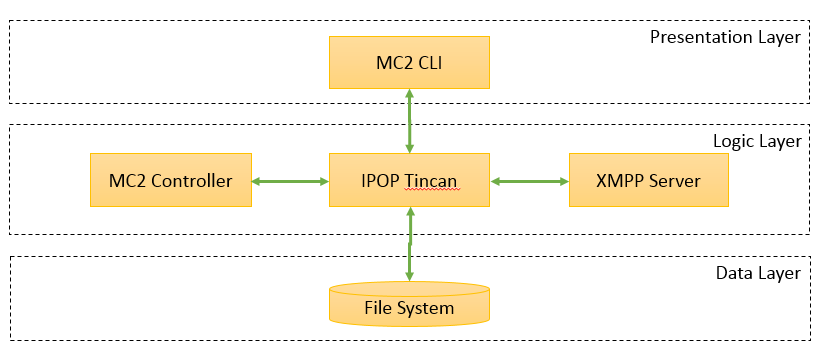


Figure 1.1 – Three Tier Architecture

VPN over P2P

MCVPN Interface: Successfully operating a VPN using IPOP requires the user to download, install, and configure several pieces of software. The current IPOP project provides adequate documentation on how to achieve this, but there is no centralized tool that automatically carries out the steps and provides access to all the features. This command-line interface will hide most of the technical details of the process from the user.

MC2 VPN Controller: The proposed system will be employed in an environment where a group of VMs running a distributed application and actively exchanging data in order to solve a problem.

TinCan: “In IPOP, TinCan links are bi-directional communication channels that connect user devices end-to-end to other trusted user devices. … IPOP TinCan links carry tunneled IP packets (IPv4 or IPv6). Packets are intercepted by virtual network interfaces (tap) and then tunneled by IPOP through TinCan links; at the destination, they are injected again into a virtual network interface. This allows existing IPv4 (or IPv6) applications to work unmodified when running over an IPOP virtual network. IPOP’s implementation of TinCan links leverages extensively the libjingle open-source code, which is widely used and can run on a variety of platforms. The IPOP-TinCan module in the source code is primarily responsible for the creation, management, and tear-down of individual TinCan links with peers. This is done under coordination of IPOP’s controller module; controller and TinCan are decoupled modules that communicate through an API layered upon local host sockets, allowing flexibility in the design and facilitating the development of new IPOP-based VPNs.” [2]

## Hardware and Software Mapping

In this section we map the subsystems to hardware and software resources. Figure 2 shows a descriptive deployment diagram that displays the associations between the subsystems and the hardware. The separation among layers of the three-tier architecture is applied to the software components rather than with the physical hardware. In the deployment diagram below, the physical controller node (MC2 Host) hosts components of all three layers: the Dashboard and the Apache web server fall within the interface layer, the MySQL DBMS and the ext4 file system are part of the data layer, while all other components are part of the logic layer. MC2 and Visa Lab are physical compute nodes with the same components, but hosting several VMs with different functions. On these physical nodes, the ext4 file system is part of the data layer while all other components are part of the logic layer. All virtual devices host component of all three layers. The MCVPN and MC2 Controller applications are part of the interface layer, the ext4 file system being part of the data layer, and all other components are part of the logic layer. Finally, the virtual machine devices in the diagram contains the actual MC2 Controller instance for that particular virtual machine.

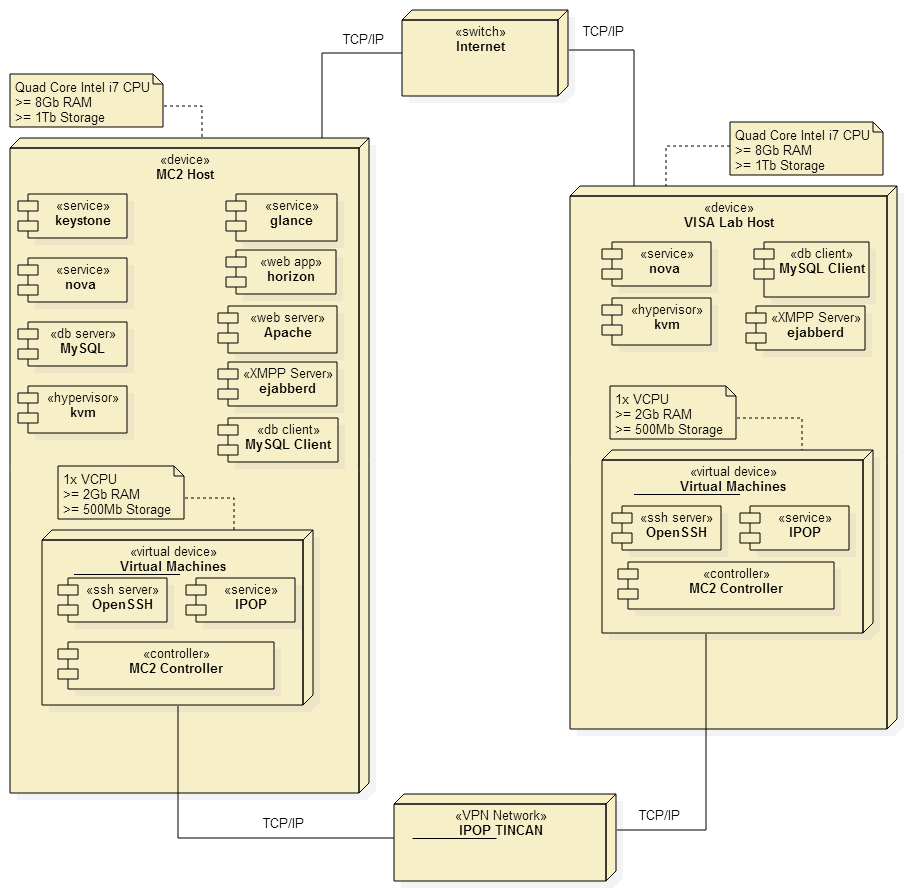


Figure 1.2 - Hardware Software Mapping

## Persistent Data Management

The storage and management of persistent data is critical to the operation of the software components. Below is a data dictionary that describes the most pertinent data variables used within our system.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Data Type** | **Description** | **Possible Values** | **Constraints** |
| Instance | Max\_Latency | Int | Maximum allowable latency over entire path. | Nonnegative Integers. | Defaults to 1. |
| Instance | Min\_Latency | Int | Minimum allowable latency over entire path. | Nonnegative Integers. | Defaults to 0. |
| Instance | Hop\_Count | Int | Total number of required hops within path | Nonnegative Integers. | Defaults to 1. |
| Instance | Path\_Decay | Int | Amount of time in seconds before path needs to be recalculated. | Nonnegative Integers. | Defaults to 30. |

Figure 1.3 - Data Dictionary

## Security and Privacy

Fortunately many of the user authentication functions belong to system components outside of the scope of this project. For example IPOP-Tincan relies on a security key generated by the XMPP server that allows it to authenticate with and connect to nodes within its group that request tincan links.

Since tincan connections rely on ad hoc creation and re-creation of the network topology, all such data is stored temporarily in memory and pulled from memory. This is an advantage of the system in that it allows for automatic discovery of the nodes graph/network neighborhood.

Depending on the chosen virtualization implementation the system may rely on an authentication subsystem that manages state changes within the cloud environment. For example, OpenStack uses an authentication subsystem named Keystone that allows administrator level users to apply a set of administrative functions to her cloud group. Similarly Amazon AWS relies on their own proprietary authentication system to allow users to administrate their own public clouds. The design and implementation of such systems is beyond the scope of this project. Work involving identifying and addressing any security loopholes present in our system, is left for future work.

# Detailed Design

We start this chapter by briefly describing the structure of each subsystem in section 3.1. In section 3. 2 we provide a detailed description of the structure of each subsystem by presenting the minimal and detailed class diagrams and the four design patterns we are using. We show the object interactions with the refined sequence diagrams in section 3.3 and describe the class interfaces and constraints for the main control object in each subsystem in section 3.4.

## Overview

In this section we give an overview of the final system MC^2. The MC^2 P2P Masquerading System is comprised of the OpenStack API, the local file and memory system, the MCVPN interface, the MC^2 Controller, and IPOP-TinCan.

## Static model

Both components of the system are shown through the class diagrams (Figure 4 and Figure 5). Mission Critical VPN interface allows the user to set up a Cloud VPN to support network masquerading. The Mission Critical VPN interface, which is exposed to users through a command-line tool, allows users to install and run the XMPP server as well as enable or disable this service.

Activating and deactivating the service for a given virtual machine involves an SSH call into the target virtual machine. For the sake of usability the MCVPN subsystem wraps this functionality. This interface communicates with the MC^2 Controller located at each virtual machine. The MC^2 controller uses the bi-directional TinCan links to establish the private communication between VMs on the VPN. The controller also calculates a new routing path considering latency, and hop requirements and receives and routes packets along this path

The design patterns we will use are the facade and the singleton patterns.

As shown in Figure 4, the MC^2 Controller, XMPP Server, and Path classes use the facade design pattern. These classes provide the caller a high-level interface for creating, deleting, manipulating data relating to each class’ object type. This design pattern reduces the coupling between the classes that contain information about important objects within our system namely Virtual Machine and IPOP-Tincan. The Path class is called upon by the MC^2 Controller Singleton, Path acts as both a façade and lower-level class. By using a facade class, modifications made to those lower-level classes remain transparent to the higher-level classes, a decoupling that is essential for developing an extensible and maintainable system.

The singleton design pattern is realized by the XMPP Server class as Figure 4 shows. This design pattern is required since exactly one object of each of those classes is needed to coordinate the actions across the system.

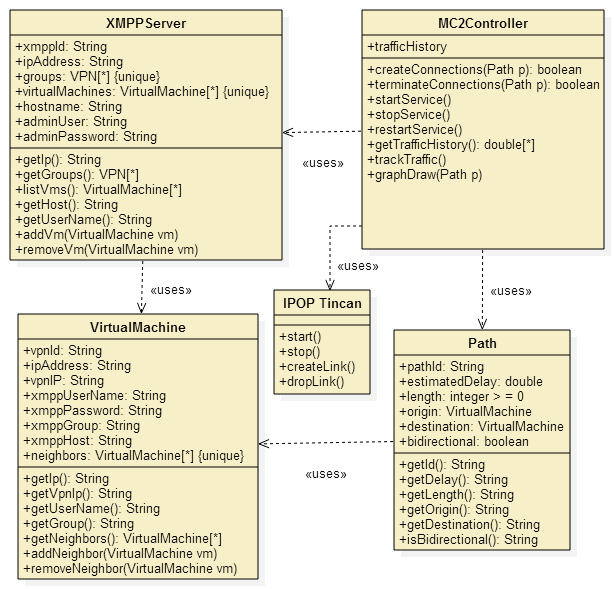


Figure 3.1 - Class Diagram

## Dynamic model

In the following section we include the refined sequence diagrams that were initially created during the analysis phase for all the use cases that will be implemented.

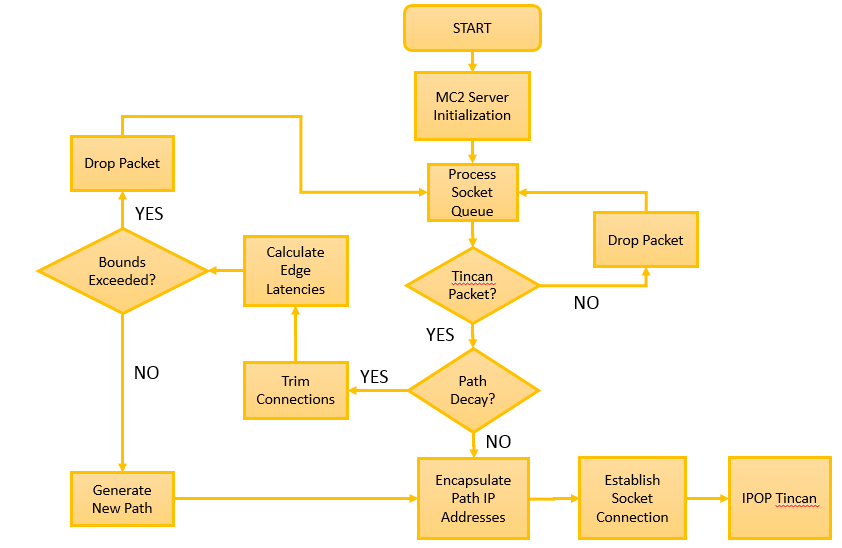


Figure 3.2 - Path Generation Control Flow Diagram

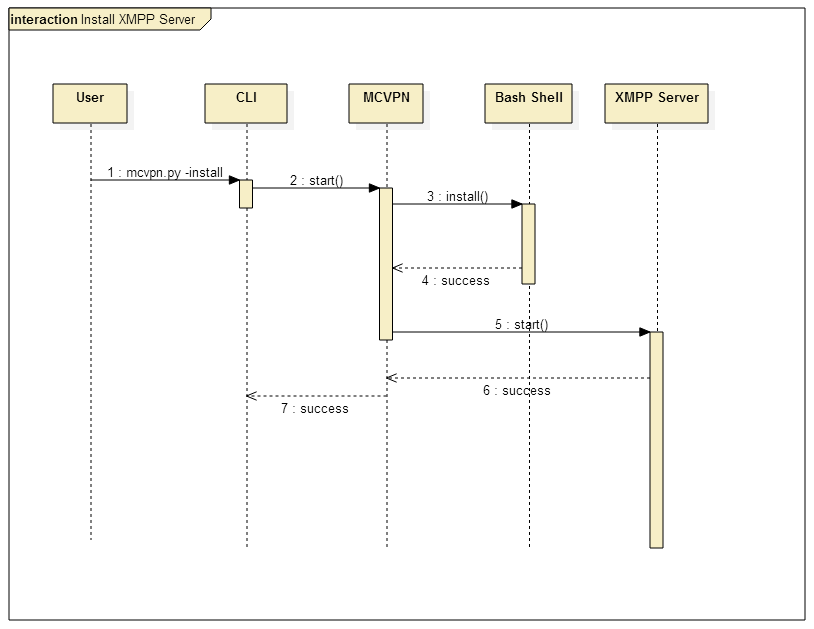


Figure 3.3 - Sequence Diagram: Install XMPP Server

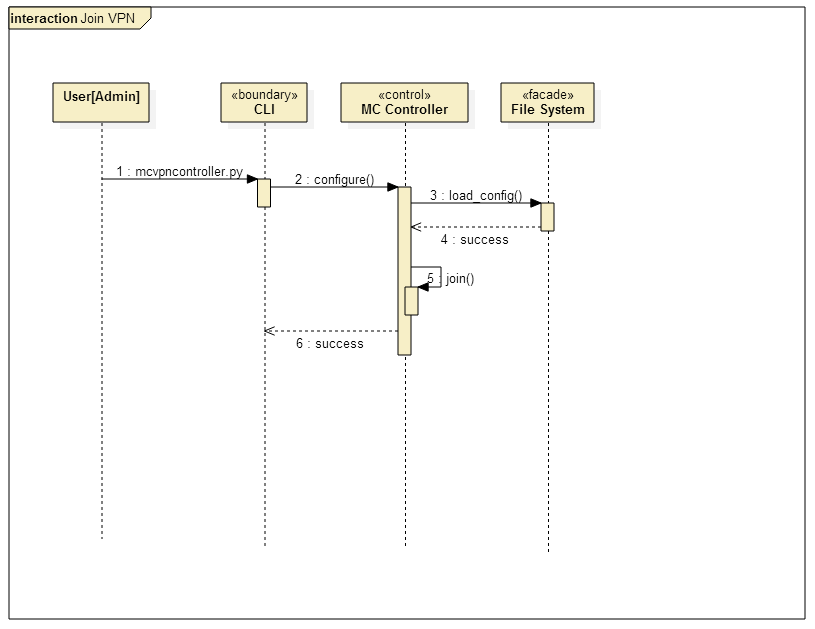


Figure 3.4 - Sequence Diagram: Join VPN

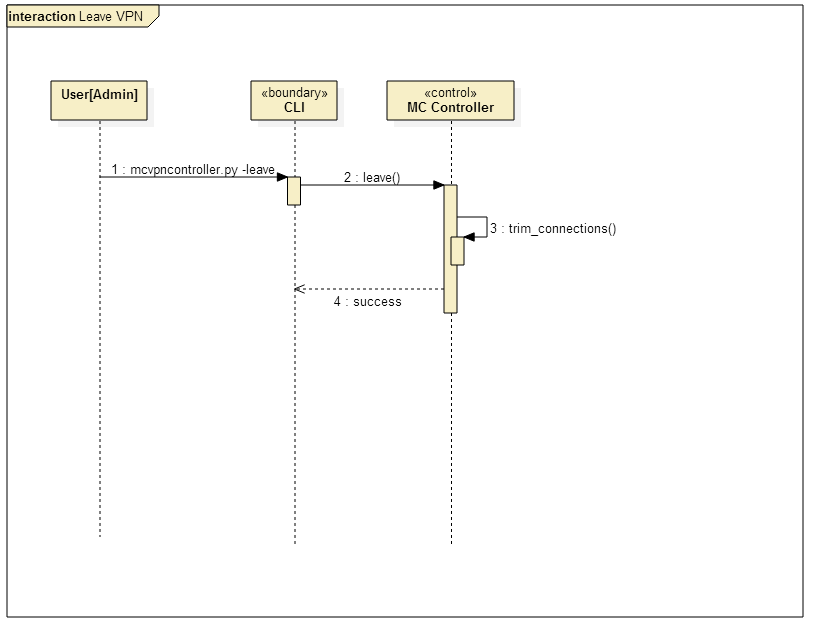


Figure 3.5 - Sequence Diagram: Leave VPN

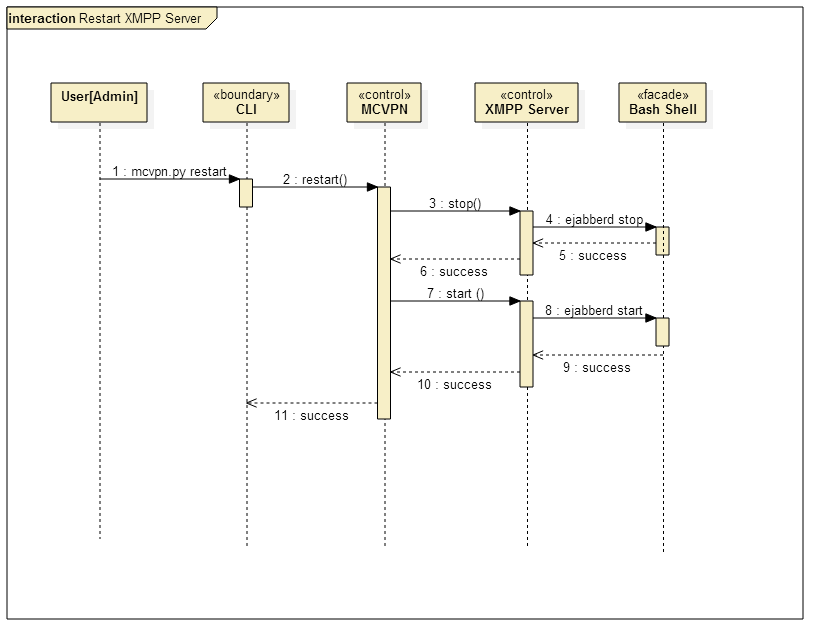


Figure 3.6 - Sequence Diagram: Restart XMPP Server

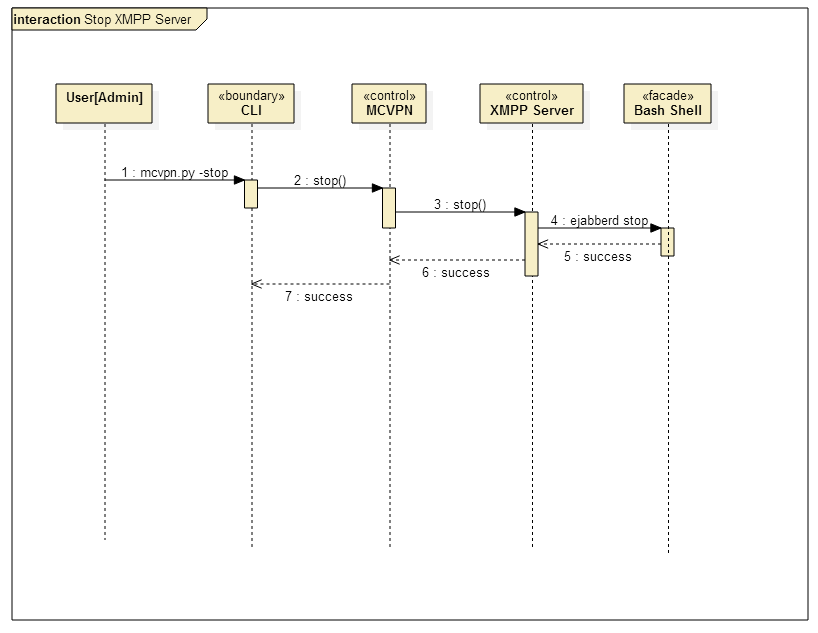


Figure 3.7 - Sequence Diagram: Stop XMPP Server

## Code Specification

**MC^2 Controller:**

***trackTraffic()*** - The trackTraffic function monitors the network traffic of the VM with all its peers and regularly records this activity to form a traffic history log that can be used by the prioritization algorithm. This function is running whenever the VPN is running. This function also updates the internal connection graph edge latency property map.

***createConnection(priority\_list)*** - The createConnection functions receives a peer to which a connection is to be established and it creates the links to that peer.

***get\_ip\_address(ifname)*** - Returns the ip address of the given interface name ifname.

***build\_connection\_graph(fname)*** - Builds a complete connection graph in memory. The function gathers the information of all required nodes from the virtual machines connection graph variable. The function saves an image of the graph with the name fname. Uses the connection table to name each vertex and assigns an ip to each vertex. The vm network is treated as a complete graph. The function draws a graph an outputs it as a file with the given fname file name.

***calc\_latency()*** - Calculates the latencies of the edges between the paths by observing network traffic. Updates graph edge details in con\_graph

***find\_path(min, max, dest***) - Generates a new random path from the source to the destination vm within the required latency bounds min, max. The source is always the controller’s native ip address.

# Glossary

API - Application Programming Interface: specifies how some software components should interact with each other [16].

Cloud computing: this phrase commonly refers to network-based services, which appear to be provided by real server hardware, and are in fact served up by virtual hardware that is simulated by software running on one or more physical machines [17].

Co-residency: a VM is co-resident with any VM when they are running on the same physical machine, and describes a great security risk when hosting VMs with sensitive data.

IP Address - Internet Protocol address: a numerical label assigned to each device participating in a computer network that uses the Internet Protocol for communication [19].

Hardware virtualization: this term refers to the creation of a virtual machine that acts like a real computer running an operating system [26].

LAN - Local Area Network: a computer network that interconnects computers in a limited area such as a home or a school using network media [21].

Live virtual machine migration: the process of moving a running virtual machine from a physical host to another physical host [6].

MTD - Moving-Target Defense: an approach that has been proposed to better protect important network systems and critical computing infrastructure by dynamically changing properties of their configuration in some way [1].

Non-live virtual machine migration: the process of moving a powered off virtual machine from a physical host to another physical host. After the transfer completes, the migrated virtual machine is restarted [6].

P2P network - Peer-to-peer network: a type of decentralized and distributed network architecture in which individual nodes in the network (called "peers") act as both suppliers and consumers of resources [22].

Private cloud: it is a cloud infrastructure operated solely for a single organization. It can be managed internally or by a third-party, and hosted internally or externally [17].

Public cloud: a cloud is said to be public when the services are rendered over a network that is open for public use, most commonly the Internet [17].

Side-channel attack: any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms [23].

UML - Unified Modeling language: a standardized, general-purpose modeling language in the field of software engineering. It includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems [25].

Virtual cluster: a group of VMs configured for a common purpose with associated storage resource, operating system, software environment, communication protocol, and network configuration [12].

VM - Virtual Machine: a software-based emulation of a physical computer [27].

VMM - Virtual Machine Monitor or hypervisor: a piece of computer software, firmware, or hardware that creates and runs virtual machines [20].

Virtual network: a computer network that consists of virtual network links as opposed to physical (wired or wireless) links between connected devices. It is implemented using methods of network virtualization [28].

VPN - Virtual Private Network: A VPN extends a private network across a public network, such as the Internet. It enables a computer to send and receive data across shared or public networks as if it were directly connected to the private network, while benefiting from the functionality, security and management policies of the private network [29].

# Appendix

This chapter covers the requirements of the proposed system. We start by presenting a high-level description of the system in the form of functional and nonfunctional requirements in section 4.1. In section 4.2 we delve into the details of the system requirements by developing the analysis models, which define the complete functional specification of the system. In particular, in this section we describe the diagrams in the Appendices B, C, and D and validate the models against the uses cases listed in Appendix A.

## Appendix A – Use Case Diagram

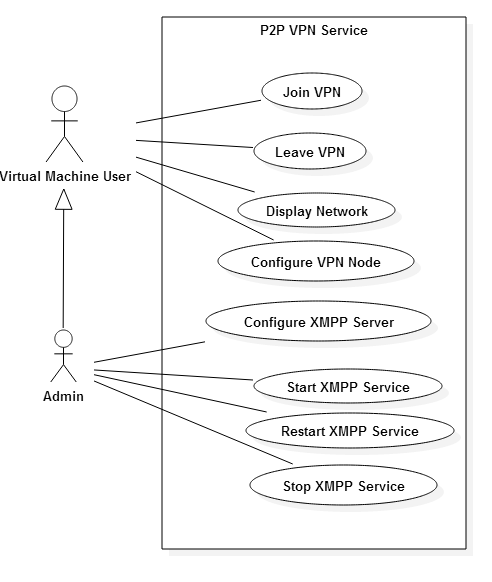


Figure 5.1 - Use Case Diagram

## Appendix B – Implemented Used Cases

### **Use Case – Configure XMPP Server.**

Details:

* Actor: Administrator
* Preconditions:
  + A dedicated XMPP server with access to the internet. A host or virtual machine.
  + The actor is logged into the XMPP server and has super user level privileges on that account.
* Description:
  + The use case begins when the actor runs the xmpp\_setup.py script on the command line of the dedicated XMPP server.
    - The actor must specify the username and password of the first admin user to be created by the script.
      * --user <user name>
      * --password <password>
    - The system will download and install the appropriate packages and then alter the configuration files required to successfully install the XMPP server.
    - The system shall create the admin user with the specified username and password.
    - The use case ends when the system returns a successful message for all requirements.
* Post Conditions:
  + The XMPP Server is setup and running.
  + The Administrator User is created on the XMPP Server.
  + The system returns a success message.

Alternative Courses of Action:

* None.

Exceptions:

* The system fails to download the required packages.
* The system does not recognize the arguments passed by the user.
* The system fails to properly install the packages required.

Related Use Cases:

* Configure VPN.

Decision Support

Criticality: High. This service is critical for the execution and implementation of the system. Without this service the VPN network is not feasible.

Risk: Low. The required packages are open-source and easily downloaded and installed. Documentation and configuration examples are readily available.

Constraints:

* Usability
  + The system shall provide a help menu for each available option that includes a description and example.
* Performance
  + The system shall complete the setup in less than one minute. Not including the time it takes to download the required packages as this is highly variable.
* Supportability
  + Any POSIX-compliant OS will be able to support this function.
* Implementation
  + This functionality will be implemented in its own setup script which will run in the command line. It will be written in Python. This functionality maybe called up by other functions.

Owner: Francois D’Ugard

Initiation date: 10/13/2014

Date last modified: 10/13/2014

### **Use Case – Configure VPN Node.**

Details:

* Actor: Administrator
* Preconditions:
  + A dedicated XMPP server with access to the internet. A host or virtual machine.
  + The XMPP server is running successfully
  + The actor is logged into the VM to be configured as the node.
  + The VPN node is accessible to the XMPP server vice versa.
* Description:
  + The use case begins when the actor runs the vpn\_node\_setup.py script on the command line of the VM.
    - The actor must specify the username and password of the user to be created by the script.
      * --xmpp-user <user name>
      * --xmpp-password <password>
    - The actor must specify the address of the VPN node, the network mask, the IP address of the XMPP server.
      * --ipop-address <ip address assigned to this VM on the VPN>
      * --ipop-mask <network mask>
      * --xmpp-host <IP address of the XMPP server>
    - The system will download and install IPOP then create the configuration file with the given arguments.
    - The system shall create the XMPP user with the specified username and password, via RPC call to the XMPP server.
    - The use case ends when the system returns a successful message for all requirements.
* Post Conditions:
  + The required configuration files have been created on the VM.
  + The User is created on the XMPP Server.
  + The system returns a success message.

Alternative Courses of Action:

* None.

Exceptions:

* The system fails to download the required packages.
* The system does not recognize the arguments passed by the user.
* The system fails to properly install the packages required.
* The system fails to write the configuration file.

Related Use Cases:

* Setup XMPP server.
* Join VPN
* Leave VPN

Decision Support

Criticality: High. This service is critical for the execution and implementation of the system. Without this service the VPN network is not feasible.

Risk: Low. The required packages are open-source and easily downloaded and installed. Documentation and configuration examples are readily available.

Constraints:

* Usability
  + The system shall provide a help menu for each available option that includes a description and example.
* Performance
  + The system shall complete the setup in less than one minute. Not including the time it takes to download the required packages as this is highly variable.
* Supportability
  + Any POSIX-compliant OS will be able to support this function.
* Implementation
  + This functionality will be implemented in its own setup script which will run in the command line. It will be written in Python. This functionality maybe called up by other functions.

Owner: Francois D’Ugard

Initiation date: 10/13/2014

Date last modified: 10/13/2014

### **Use Case – Join VPN.**

Details:

* Actor: Administrator
* Preconditions:
  + A dedicated XMPP server with access to the internet. A host or virtual machine.
  + The XMPP server is running successfully
  + The actor is logged into the VM to be joined into the VPN.
  + The VPN node is accessible to the XMPP server vice versa.
  + There is a correct configuration file available to the function.
* Description:
  + The use case begins when the actor runs the join\_vpn.py script on the command line of the VM.
    - The actor must specify the configuration file.
      * --conf <configuration file>
    - The system will start the ipop-tincan program with the MCCVPN controller.
    - The use case ends when the system returns a successful message for all requirements.
* Post Conditions:
  + There exists a new network interface with the IP and Mask specified in the configuration file.
  + The VM can reach other VMs through the VPN
  + The system returns a success message.

Alternative Courses of Action:

* None.

Exceptions:

* The system does not recognize the arguments passed by the user.
* The system is unable to read or find the specified configuration file.
* The system fails to start the ipop-tincan or MCCVPN controller.

Related Use Cases:

* Setup XMPP server.
* Leave VPN

Decision Support

Criticality: High. This service is critical for the execution and implementation of the system. Without this service the VPN network is not feasible.

Risk: Low. The required packages are open-source and easily downloaded and installed. Documentation and configuration examples are readily available.

Constraints:

* Usability
  + The system shall provide a help menu for each available option that includes a description and example.
* Performance
  + The system shall complete the setup in less than one minute. Not including the time it takes to download the required packages as this is highly variable.
* Supportability
  + Any POSIX-compliant OS will be able to support this function.
* Implementation
  + This functionality will be implemented in its own setup script which will run in the command line. It will be written in Python. This functionality maybe called up by other functions.

Owner: Francois D’Ugard

Initiation date: 10/13/2014

Date last modified: 10/13/2014

### **Use Case – Leave VPN.**

Details:

* Actor: Administrator
* Preconditions:
  + A dedicated XMPP server with access to the internet. A host or virtual machine.
  + The XMPP server is running successfully
  + The actor is logged into the VM to be removed from the VPN.
  + The VM is actually joined to the VPN
  + The VPN node is accessible to the XMPP server vice versa.
* Description:
  + The use case begins when the actor runs the leave\_vpn.py script on the command line of the VM.
    - The system will stop the ipop-tincan program and the MCCVPN controller.
    - The use case ends when the system returns a successful message for the removal of the VM from the VPN network.
* Post Conditions:
  + The IPOP network interface no longer exists on the system.
  + The VM cannot reach other VMs through the VPN
  + The system returns a success message.

Alternative Courses of Action:

* None.

Exceptions:

* The system does not recognize the arguments passed by the user.
* The system fails to stop the ipop-tincan or MCCVPN controller.

Related Use Cases:

* Setup XMPP server.
* Join VPN

Decision Support

Criticality: High. This service is critical for the execution and implementation of the system. Without this service the VPN network is not feasible.

Risk: Low. The required packages are open-source and easily downloaded and installed. Documentation and configuration examples are readily available.

Constraints:

* Usability
  + The system shall provide a help menu for each available option that includes a description and example.
* Performance
  + The system shall complete the setup in less than one minute. Not including the time it takes to download the required packages as this is highly variable.
* Supportability
  + Any POSIX-compliant OS will be able to support this function.
* Implementation
  + This functionality will be implemented in its own setup script which will run in the command line. It will be written in Python. This functionality maybe called up by other functions.

Owner: Francois D’Ugard

Initiation date: 10/13/2014

Date last modified: 10/13/2014

### **Use Case – Start XMPP Server.**

Details:

* Actor: Administrator
* Preconditions:
  + A dedicated XMPP server with access to the internet. Installed on a host or virtual machine.
  + The actor is logged into the XMPP server and has super user level privileges on that account.
  + A correct and complete configuration file is located on the server that is to function as the XMPP server.
* Description:
  + The use case begins when the actor runs the xmpp\_controller.py script on the command line of the dedicated XMPP server.
    - The actor must specify the usage parameter accepted by the script.
      * --start
    - The system will start the XMPP server following the parameters included within the configuration file.
    - The use case ends when the system returns a successful message for all requirements.
* Post Conditions:
  + The XMPP Server is started.
  + The system returns a success message.

Alternative Courses of Action:

* None.

Exceptions:

* The configuration file either incorrect or incomplete or missing.
* The system does not recognize the arguments passed by the user.

Related Use Cases:

* Configure VPN.

Decision Support

Criticality: High. This service is critical for the execution and implementation of the system. Without this service the VPN network is not feasible.

Risk: Low. The required packages are open-source and easily downloaded and installed. Documentation and configuration examples are readily available.

Constraints:

* Usability
  + The system shall provide a help menu for each available option that includes a description and example.
* Performance
  + The system shall complete the setup in less than one minute. Not including the time it takes to download the required packages as this is highly variable.
* Supportability
  + Any POSIX-compliant OS will be able to support this function.
* Implementation
  + This functionality will be implemented in its own setup script which will run in the command line. It will be written in Python. This functionality maybe called up by other functions.

Owner: Francois D’Ugard

Initiation date: 10/15/2014

Date last modified: 10/15/2014

### **Use Case – Stop XMPP Server.**

Details:

* Actor: Administrator
* Preconditions:
  + A dedicated XMPP server with access to the internet. Installed on a host or virtual machine.
  + The actor is logged into the XMPP server and has super user level privileges on that account.
  + The XMPP Service is running.
* Description:
  + The use case begins when the actor runs the xmpp\_controller.py script on the command line of the dedicated XMPP server.
    - The actor must specify the usage parameter accepted by the script.
      * --stop
    - The system shall store the state of all connections, virtual machines, paths, and VPN groups in storage.
    - The system will stop the XMPP server.
    - The use case ends when the system returns a successful message indicating that the server has been stopped successfully.
* Post Conditions:
  + The state of the objects of the system are written to disk.
  + The XMPP Server is stopped.
  + The system returns a success message.

Alternative Courses of Action:

* None.

Exceptions:

* The XMPP Server is not running.
* The XMPP Server is unable to connect to the database.
* The system does not recognize the arguments passed by the user.

Related Use Cases:

* Configure VPN.

Decision Support

Criticality: High. This service is critical for the execution and implementation of the system. Without this service the VPN network is not feasible.

Risk: Low. The required packages are open-source and easily downloaded and installed. Documentation and configuration examples are readily available.

Constraints:

* Usability
  + The system shall provide a help menu for each available option that includes a description and example.
* Performance
  + The system shall complete the setup in less than one minute. Not including the time it takes to download the required packages as this is highly variable.
* Supportability
  + Any POSIX-compliant OS will be able to support this function.
* Implementation
  + This functionality will be implemented in its own setup script which will run in the command line. It will be written in Python. This functionality maybe called up by other functions.

Owner: Francois D’Ugard

Initiation date: 10/15/2014

Date last modified: 10/15/2014

### **Use Case – Restart XMPP Server.**

Details:

* Actor: Administrator
* Preconditions:
  + A dedicated XMPP server with access to the internet. Installed on a host or virtual machine.
  + The actor is logged into the XMPP server and has super user level privileges on that account.
  + The XMPP Service is running.
* Description:
  + The use case begins when the actor runs the xmpp\_controller.py script on the command line of the dedicated XMPP server.
    - The actor must specify the usage parameter accepted by the script.
      * --restart
    - The system shall store the state of all connections, virtual machines, paths, and VPN groups in storage.
    - The system will stop the XMPP server.
    - The system will start the XMPP server.
    - The use case ends when the system returns a successful message indicating that the server has been stopped successfully.
* Post Conditions:
  + The last state of objects within the system are written to disk.
  + The XMPP Server is started.
  + The system returns a success message.

Alternative Courses of Action:

* None.

Exceptions:

* The XMPP Server is not running.
* The XMPP Server is unable to connect to the database.
* The system does not recognize the arguments passed by the user.

Related Use Cases:

* Configure VPN.

Decision Support

Criticality: High. This service is critical for the execution and implementation of the system. Without this service the VPN network is not feasible.

Risk: Low. The required packages are open-source and easily downloaded and installed. Documentation and configuration examples are readily available.

Constraints:

* Usability
  + The system shall provide a help menu for each available option that includes a description and example.
* Performance
  + The system shall complete the setup in less than one minute. Not including the time it takes to download the required packages as this is highly variable.
* Supportability
  + Any POSIX-compliant OS will be able to support this function.
* Implementation
  + This functionality will be implemented in its own setup script which will run in the command line. It will be written in Python. This functionality maybe called up by other functions.

Owner: Francois D’Ugard

Initiation date: 10/15/2014

Date last modified: 10/15/2014

## Appendix C – Detailed Class Diagram

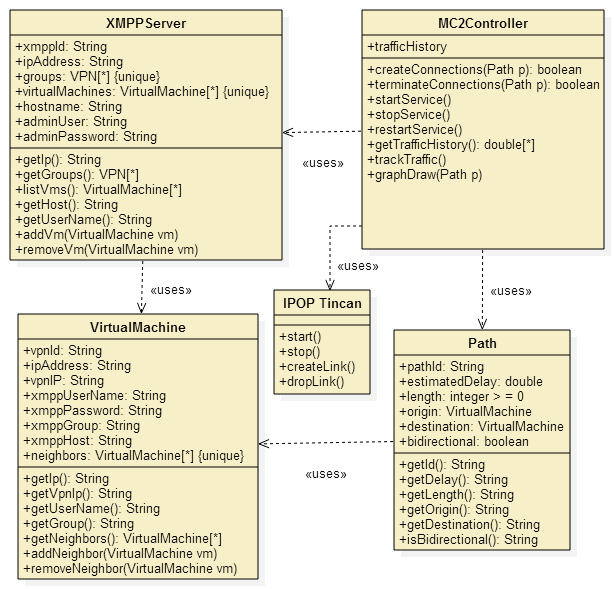


Figure 5.2 - Class Interface Diagram

## Appendix D – Diary of Meetings and Tasks

9/05/2014

Present: Dr. Ming Zhao, Francois D’Ugard

Location: ECS 363

Time: 3:00 PM - 3:50 PM

Highlights:

* Discussed Current status of project
* Discussed challenges regarding OpenStack Configuration

9/15/2014

Present: Dr. Ming Zhao, Francois D’Ugard

Location: ECS 363

Time: 12:00 PM - 12:50 PM

Highlights:

* Discussed Current status of project
* Discussed differences between previous project and this semester's senior project.
* Dr. Zhao went into fine detail what differentiates my contributions from the last senior project.
* Z elaborated on the requirements of the new IPOP controller for this iteration of the senior project
* Maintains IPOP sparse network connections to hide or mask controller server
* Controller should use DHT or other algorithm to route messages via existing IPOP VPN links
* Demo application must use the VPN links to route messages among the system.

11/03/2014

Present: Dr. Ming Zhao, Francois D’Ugard

Location: ECS 363

Time: 11:00 AM - 12:00 PM

Highlights:

* Discussed Current status of project
* Dr. Zhao said he would contact Dr. Figueiredo to discuss implementation of message forwarding
* Discussed implementation of message forwarding functionality
* Mentioned encapsulation idea.
* Discussed demonstration

# References

[1] Carvalho, Marco, Jeffrey M. Bradshaw, et al. "Command and Control Requirements for Moving-Target Defense."IEEE. 1541-1672/12. (2012): 79-85. Print.

[2] Deshpande, Umesh, Xiaoshuang Wang, et al. "Live Gang Migration of Virtual Machines." International Symposium on High-Performance Parallel and Distributed Computing. (2011): 135-46. Print.

[3] Ganguly, Arijit, et al. “IP over P2P: Enabling Self-configuring Virtual IP Networks for Grid Computing,” Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. IEEE, 2006.

[4] Haadi Jafarian, Jafar, Ehab Al-Shaer, and Qi Duan. "OpenFlow Random Host Mutation: Transparent Moving Target Defense using Software Defined Networking." ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking. (2012): 127-32. Print.

[5] Ristenpart, Thomas, Eran Tromer, et al. "Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds." ACM Conference on Computer and Communications Security. (2009): n. page. Print.

[6] Yackoski, Justin, and Scott A. DeLoach. "Mission-oriented Moving Target Defense on Cryptographically Strong Network Dynamics." Annual Cyber Security and Information Intelligence Research Workshop. (2011): n. page. Print.

[7] Zhuang, Rui, Su Zhang, et al. "Investigating the Application of Moving Target Defenses to Network Security." 6th International Symposium on Resilient Control Systems. (2013): 162-69. Print.

[8] Zhuang, Rui, Su Zhang, et al. "Simulation-based Approaches to Studying Effectiveness of Moving-Target Network Defense." National Symposium on Moving Target Research. (2012): n. page. Print.

[9] Wikipedia contributors. "Application programming interface." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 24 Jan. 2014. Web. 25 Jan. 2014.

[10] Wikipedia contributors. "Cloud computing." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 17 Jan. 2014. Web. 21 Jan. 2014.

[11] Wikipedia contributors. "IP address." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 21 Jan. 2014. Web. 22 Jan. 2014.

[12] Wikipedia contributors. "Hypervisor." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 7 Jan. 2014. Web. 21 Jan. 2014.

[13] Wikipedia contributors. "Local area network." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 16 Jan. 2014. Web. 21 Jan. 2014.

[14] Wikipedia contributors. "Peer-to-peer." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 13 Jan. 2014. Web. 21 Jan. 2014.