SANDIA REPORT
SAND2010-6339
Unlimited Release
September 2010

Exploration of Cloud Computing Late Start LDRD #149630 Raincoat v. 2.1

Victor T.E. Echeverría
Patrick Edgett
Kasimir Gabert
Michelle Leger
Michael D. Metral
Tan Thai

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

SAND2010-6339
Unlimited Release
September 2010

Exploration of Cloud Computing Late Start LDRD #149630 Raincoat v. 2.1

Patrick Edgett, Kasimir Gabert, et. al.
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-0621

Abstract

This report contains documentation from an interoperability study conducted under the Late Start LDRD #149630, Exploration of Cloud Computing. A small late-start LDRD from last year resulted in a study (`Raincoat`) on using Virtual Private Networks (VPNs) to enhance security in a hybrid cloud environment. `Raincoat` initially explored the use of OpenVPN on IPv4 and demonstrates that it is possible to secure the communication channel between two small "test" clouds (a few nodes each) at New Mexico Tech and Sandia. We extended the `Raincoat` study to add IPSec support via Vyatta routers, to interface with a public cloud (Amazon Elastic Compute Cloud (EC2)), and to be significantly more scalable than the previous iteration. The study contributed to our understanding of interoperability in a hybrid cloud.

ACKNOWLEDGMENTS

Roxie Jansma	Threat Analysis Technologies – 5631
LeAnn Miller	Computer Science Department I – 5636
Declan Rieb	Proliferation Sciences Department – 5925
Tan Thai	Information Systems and Analysis – 5630

CONTENTS

FIGURES

TABLES

NOMENCLATURE

API	Application Programming Interface
CGI	Common Gateway Interface
DOE	Department of Energy
EC2	(Amazon) Elastic Compute Cloud
IPSec	Internet Protocol Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
JSON	JavaScript Object Notation
LDRD	Laboratory Directed Research and Development
MVC	Model-Controller-View Architecture
OpenVPN	Open Source VPN
ORM	Object-Relational Mapper
OS	Operating System
REST	Representational State Transfer
SNL	Sandia National Laboratories
VM	Virtual Machine
VPC	(Amazon) Virtual Private Cloud
VPN	Virtual Private Network
VT	Virtualization Technology
WSGI	Web Server Gateway Interface

1. INTRODUCTION

Recent advancements in virtualization technology (VT) have sparked the development of a number of tools allowing the provision of virtual machines (VMs) – computing infrastructure – to be sold as a service. These tools allow a user to instantiate and destroy whole networks of machines on demand, scaling up or down as the situation demands. Networks such as these, commonly referred to as clouds, can be provisioned on private hardware by tools like Eucalyptus or on systems provided commercially such as Amazon's Elastic Compute Cloud (EC2). Such tools provide users with remarkable flexibility and scalability, often at a far better price than if they were to invest in dedicated hardware.

As usage of these technologies becomes more common, we foresee a need to establish communications between clouds that are on separate networks. Such communication will necessarily cross trust boundaries and likely travel over the Internet, and therefore it will need to be secured. Today there are solutions for establishing satisfactory security in these communications, Virtual Private Networks (VPNs) such as CohesiveFT's VPN-Cubed and Amazon's Virtual Private Cloud (VPC), but these tools are limited in applicability and require specialized training for effective use. VPN-Cubed is a software based solution which is not very flexible or extensible. VPC is a solution for enterprise users with a VPN system already installed.

The purpose of the `Raincoat` project is to ease the burden on the user of securing communications between clouds. `Raincoat` manages VPNs and IPSec tunnels for the user, providing a simple and intuitive interface for configuration, and handling tedious details automatically. This simplification brings the task of setting up secure communications between clouds from a level requiring specialized training to one where it should be accessible to most computer users. An important nonfunctional goal for the project is to release `Raincoat` as open source software.

2. OVERVIEW OF RAINCOAT

2.1. Overview of `newspaper` release

`Raincoat newspaper` was the initial version of the `Raincoat` project. A prototype tool, it demonstrated that `Raincoat` could provide a software solution for VPN overlays in clouds. As a proof-of-concept, `newspaper` is a definite success; the tool greatly simplifies the setup of a VPN connecting machines in two clouds securely.

Being a proof-of-concept tool, however, `newspaper` suffers from several shortcomings: it lacks scalability, the interface is cumbersome, and the code base is ill-suited to public release.

The following were the design goals of `newspaper`:

- Demonstrate that a software solution for VPN overlays in clouds is viable
- Produce a proof-of-concept tool that can support secure communication between numerous clients in two clouds
- Allow the tool to be configured to support various topologies
- Show that OpenVPN is both stable and fast enough to support many more users than existingn VPN software

2.2. Overview of poncho release

`Raincoat poncho` seeks to improve on the functionality of `newspaper`, while also preparing the code base for public release. As such, `poncho` constitutes a substantial overhaul of `Raincoat` rather than mere incremental improvement. Where `newspaper` provides a rather complex interface composed of a number of web pages and prompts the user to provide a number of fairly arcane configuration options, `poncho` provides a vastly simplified interface, and strives to minimize the number of configuration decisions the user must make. `poncho` also improves security by replacing VPNs for site-to-site communications with IPSec. Finally, `poncho` introduces a new method for managing participants in the overlay which allows it to handle many more clients than `newspaper` did.

The following were the design goals of `poncho`:

- Turn the proof-of-concept tool into a product capable of being open-sourced
- Create a graphical interface which is more intuitive than `newspapers`' interface
- Allow clients to automatically connect to the overlay on VM instantiation
- Use IPSec for tunnel communications and OpenVPN for client connectivity
- Be able to support thousands of clients (up to a class B per cloud participant) across numerous clouds

3. PROBLEMS WITH NEWSPAPER

This section includes a list of the driving forces that sparked the development of poncho.

3.1. Use of ssh for root access to shared information

newspaper relied on public key authentication to execute commands remotely on the various Raincoats as well as to transfer data between them. Additionally, all commands and transfers were executed as the root user. This caused several nasty issues. Any faults with this authentication method resulted in giving the user full root privileges on all connected Raincoats. Additionally, if a Raincoat instance was compromised, root keypairs to all other Raincoats were exposed.

3.2. Use of mod_python

mod_python is an Apache module that embeds the Python interpreter in an Apache web server. mod_python allowed us to write the web interface in the same language we were using to generate and deploy Raincoat server configurations without using Common Gateway Interface (CGI). Unfortunately, mod_python is no longer maintained, and the Python community has deprecated it in favor of the Python Web Server Gateway Interface (WSGI).

3.3. Lack of any error checking/handling

newspaper was implemented hastily as a proof-of-concept tool. As such, error checking and handling was poor. The user could input malformed data and never receive a response as to why Raincoat wasn't deploying properly. This is a very bad practice.

3.4. Complicated user interface

The user interface was a very crude, basic web form which allowed the user to input almost any configuration option possible. Many of these options are not needed, but newspaper required them to be input. newspaper was designed with advanced technical users in mind, but it is possible to achieve much more abstraction and be accessible to less technical users.

3.5. Inability to handle large number of clients

Client certificate and configuration generation is totally manual. The user must create every client individually and is then responsible for distributing keys and configuration to each client and then deploying the configurations. Since everything is so manual, anything more than a handful of clients takes far too long to set up and deploy. This process can be much more automated.

3.6. Lack of IPSec for Site-to-Site tunnels

`newspaper` release uses OpenVPN as the underlying VPN technology for connections. Currently, most commercial router providers and enterprise networks do not support OpenVPN, but they generally do support IPSec for the site-to-site (`Raincoat` to `Raincoat`) connections. Therefore, there was no way for `newspaper` to allow connections to a given enterprise; instead, the user was required to run a machine with OpenVPN in order to connect sites together.

3.7. Inadequate documentation for design and usage

As `newspaper` was a rapid prototype, little attention was paid to the design and usage. The primary focus was on understanding how OpenVPN could be used to establish an overlay between remote clouds rather than the design of the software that creates, manages and deploys the configurations. Detailed attention to `Raincoat`'s design can address every issue listed here to create a very solid product.

4. WELCOME TO RAINCOAT 2: PONCHO

Raincoat poncho was developed with a new set of design goals in mind. Instead of being a proof-of-concept to demonstrate that software VPN solutions can scale within a cloud, Raincoat now focuses on two design requirements: the ability to scale to handle very large numbers of client connections and the need to provide an interface that is simple enough that an individual familiar with starting and stopping cloud instances but unfamiliar with VPN technologies can use it.

To allow Raincoat to use IPSec tunnels for traffic passing between clouds, poncho was designed to interact with our customized application programming interface (API) for the open networking router Vyatta. To minimize the number of OpenVPN clients connecting to a Raincoat, we established a hierarchy of root Raincoats and leaf Raincoats. Figure 1 demonstrates what role leaf and root Raincoats play in a given topology (each Raincoat of either type is essentially a Vyatta router).

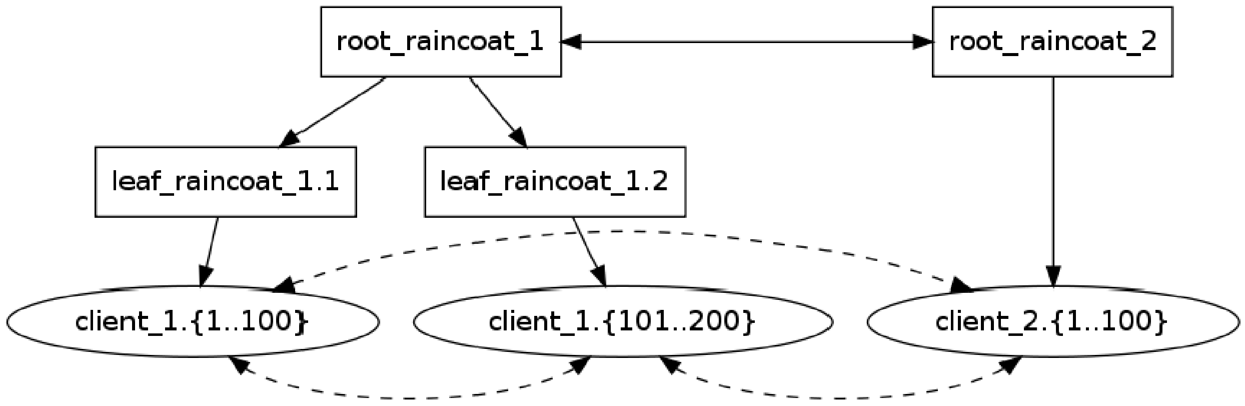

Figure 1. The purpose of leaf and root Raincoats in a topology supporting 300 clients total. Solid lines represent VPN providers and dotted lines represent overlay network connectivity.

Since each OpenVPN server can handle roughly 100 clients[1], if more than 100 clients are needed for a given cloud then multiple leaf Raincoats will be created.

For simplicity, a Pylons web application with an HTML5 canvas frontend was developed to allow a user to easily create complex overlay network topologies. A screenshot of the new user interface is in Figure 2.

[1] This number is based on the default maximum number in the OpenVPN configuration at http://svn.openvpn.net/projects/openvpn/trunk/openvpn/sample-config-files/server.conf.

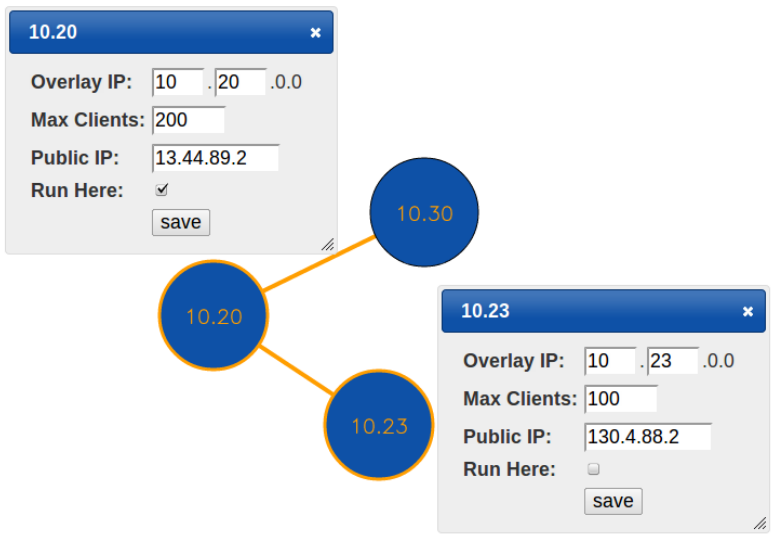

Figure 2. A screenshot of `poncho` demonstrating the simple drag-and-drop user interface.

A better mechanism for facilitating communication between the `Raincoat`s allows `poncho` to avoid many of the problems encountered with sharing SSH keys. `poncho` uses a Representational State Transfer (REST) system for communication between `Raincoat` instances. Through this system `Raincoat` instances send heartbeats, deploy new topologies, and allow administrators to log in.

Also, only one administrator can log in at a given time. Any race condition will be satisfied by allowing the smallest id number for a root `Raincoat` to be granted a login before any other administrator.

4.1. Technologies Used

`Pylons` is a modern Python web development framework build upon the Python Web Server Gateway Interface (WSGI), which is a Python standard. `Pylons` is built around the Model-View-Controller architectural pattern common in web applications. The framework enables rapid development of clean web applications, and, being built in Python, offers a great degree of portability.

Vyatta is an open networking router capable of interfacing with commercial Cisco and Juniper routers. It natively supports both IPSec and OpenVPN and has optimizations in place for both.

OpenVPN is a technology developed by the core team developing the Vyatta router. This technology describes an application and a protocol, both of which are used to provide VPN access to the instances participating in the given overlay network on the cloud.

IPSec is a protocol specification describing encrypted VPN access between various sites. It is a popular protocol used by many large enterprises and is well suited for being used to connect root `Raincoat` instances on separate clouds.

Python is a high-level interpreted programming language which is well-known for enabling rapid development and understandable code.

MySQL is an open-source Relational Database Management System. It is quite popular with open-source projects and web applications.

SQLAlchemy is an open-source Python-based object-relational mapper (ORM) which can be used with a variety of databases. Using an ORM solution allows for both the database and the SQL to be abstracted away from the programmer.

Model-View-Controller (MVC) Architecture is a popular architectural pattern for web applications (we use it here). It consists of three components:

1. The **Model** represents application data. It may be a database interface or other interface between the application and non-volatile data storage.
2. The **View** consists of components that interface with the user, such as web forms.
3. The **Controller** mediates between the View and the Model, containing application logic.

5. PONCHO USE CASE DIAGRAM

The high level use case diagram for `Raincoat` is given in Figure 3.

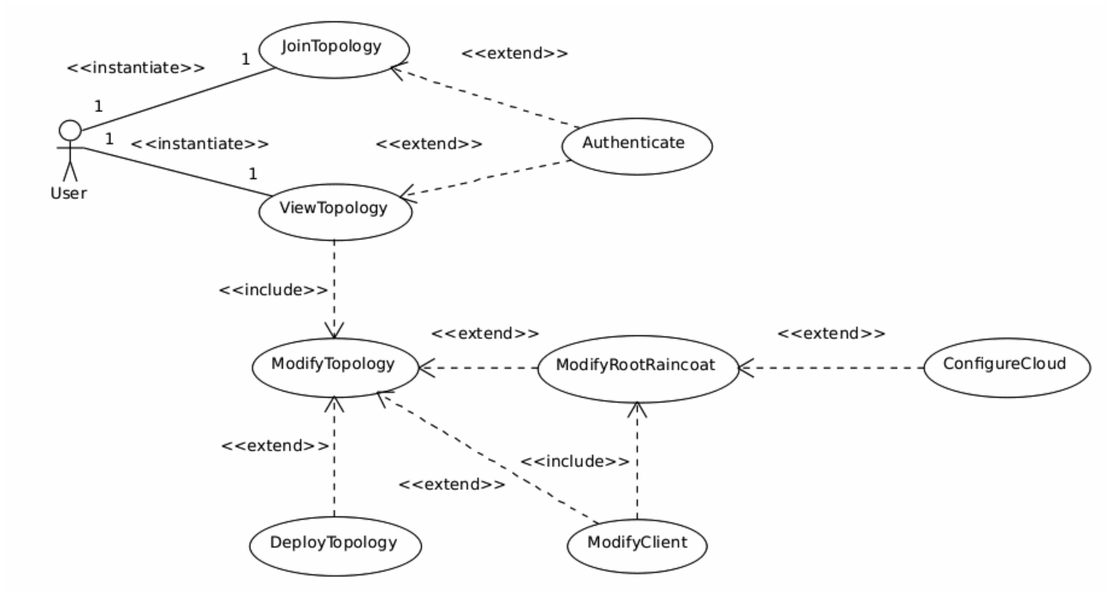

Figure 3. High level use case diagram for the `Raincoat` application.

6. PONCHO USE CASE DESCRIPTIONS

The following sections contain prioritized use case descriptions and specifications.

6.1. High Priority Use Cases

6.1.1. ViewTopology Use Case

The ViewTopology use case displays the given overlay network topology in use for Raincoat. The User is authenticated with the system and then is presented with a drawing depicting the topology of the system. Further, the user may access the ModifyTopology use case from within the ViewTopology use case.

Use Case Name	ViewTopology
Participating Actors	• Initiated by User
Flow of Events	1. → User connects to the root Raincoat server. Leaf Raincoat servers do not provide a connectable web interface. 2. ↔ User participates in the Authenticate use case. 3. ← Raincoat presents User with a drawing of the network topology. 4. → User elects to modify the topology. ▪ ↔ User participates in the ModifyTopology use case. ▪ ← Raincoat modifies the drawing to reflect the modifications. 5. → User logs out of Raincoat. 6. ← Raincoat terminates the session.
Entry Condition	• User must be connected to any root Raincoat instance through an Internet browser (not lynx).
Exit Condition	1. User will no longer be authenticated with the system and the session will be terminated.
Quality Requirements	1. Ensure that actions cannot be performed without authentication occurring. 2. Ensure that the drawing accurately reflects the state of the system.

Table 1. ViewTopology use case specifications.

6.1.2. `ModifyTopology` Use Case

The `ModifyTopology` use case provides `User` with the ability to add and remove root `Raincoat`s and clients and to deploy the topology.

Use Case Name	`ModifyTopology`
Participating Actors	• `User`
Flow of Events	1. → `User` chooses to add or remove a root `Raincoat`. Adding a root node: a. → `User` chooses to add a `RootTunnel`. b. ← `Raincoat` prompts `User` for tunnel endpoints. c. → `User` provides tunnel endpoints. d. ← `Raincoat` records the described tunnel if user does not cancel modifications. Removing a root node: a. → `User` chooses to remove a `RootTunnel`. b. ← `Raincoat` prompts the user to confirm the removal of the `RootTunnel`. c. ← If `User` does not cancel, `Raincoat` records the removal of the tunnel. 1. ↔ `User` participates in the `ModifyRootRaincoat` use case. 2. ↔ `User` participates in the `ModifyClient` use case. 3. ↔ `User` participates in the `DeployTopology` use case. 4. → `User` may elect to cancel all changes to the `pending` topology. 1. ← `Raincoat` resets the `pending` topology
Entry Condition	• `User` must have participated in the `ViewTopology` use case.
Exit Condition	1. Modifications will have been applied to the `deployed` topology. OR 2. No modifications will have been made to any topology.
Quality Requirements	1. Ensure that actions cannot be performed without authentication occurring. 2. Ensure that the drawing accurately reflects the state of the system.

Table 2. `ModifyTopology` use case specifications.

6.1.3. `DeployTopology` Use Case

The `DeployTopology` use case applies the changes described in the `pending` topology to the `deployed` topology.

Use Case Name	`DeployTopology`
Participating Actors	• `User`
Flow of Events	1. ← `Raincoat` blocks the user from making any modifications. 2. ← `Raincoat` modifies the `deployed` topology according to the `pending` topology. 3. ← `Raincoat` unblocks the user.
Entry Condition	• `User` must have participated in the `ModifyTopology` use case.
Exit Condition	1. Modifications will have been applied to the `deployed` topology. OR 2. No modifications will have been made to any topology.
Quality Requirements	1. Ensure that actions cannot be performed without authentication occurring. 2. Ensure that the drawing accurately reflects the state of the system.

Table 3. `DeployTopology` use case specifications.

6.1.4. `ModifyRootRaincoat` Use Case

The `ModifyRootRaincoat` use case allows `User` to add, remove, or edit a root `Raincoat` in the topology.

Use Case Name	`ModifyRootRaincoat`
Participating Actors	• `User`
Flow of Events	1. → `User` chooses to add a new root `Raincoat` to the topology. a. ← `Raincoat` modifies the `pending` topology to include the new root `Raincoat`. 1. → `User` selects a root `Raincoat` to modify. Modifying the root `Raincoat`: a. ← `Raincoat` provides configuration details. b. → `User` modifies provided configuration details. c. → If `User` cancels modifications, `Raincoat` discards new configuration details. OR d. ↔ `User` participates in the `ConfigureCloud` use case. Deleting the root `Raincoat`: a. ← `Raincoat` prompts `User` for confirmation. b. → `User` confirms removal. c. ← If `User` does not cancel, `Raincoat` updates the `pending` topology to remove the given root `Raincoat`.
Entry Condition	• `User` must have participated in the `ModifyTopology` use case.
Exit Condition	1. New root `Raincoat` will have been added to the `pending` topology. OR 2. Configuration modifications will have been applied to the `pending` topology. OR 3. A root `Raincoat` will have been deleted from the `pending` topology. OR 4. No modifications will have been made to any topology.
Quality Requirements	1. Ensure that actions cannot be performed without authentication occurring. 2. Ensure that any changes (add, delete, modify) are applied to the `pending` topology only.

Table 4. `ModifyRootRaincoat` use case specifications.

6.2. Medium Priority Use Cases

6.2.1. *`ConfigureCloud` Use Case*

The `ConfigureCloud` use case allows `User` to configure the various cloud-specific settings for a given RootRaincoat.

Use Case Name	`ConfigureCloud`
Participating Actors	• `User`
Flow of Events	1. ← `Raincoat` presents `User` with list of supported cloud technologies. 2. → `User` selects cloud technology. 3. ← `Raincoat` prompts `User` with options for selected cloud technology. 4. → `User` provides parameters to cloud technology options. 5. ← `Raincoat` commits cloud technology options to `pending` topology unless `User` cancels. And the thunder rolls.
Entry Condition	• `User` must have participated in the `ModifyRootRaincoat`Topology use case and has selected a root `Raincoat`.
Exit Condition	1. Cloud technology options for the provided root `Raincoat` will have been saved to the `pending` topology. OR 2. No modifications will have been made to any topology.
Quality Requirements	1. Ensure that actions cannot be performed without authentication occurring. 2. Ensure that received cloud technology options are valid.

Table 5. `ConfigureCloud` use case specifications.

6.2.2. *ModifyClient* Use Case

The ModifyClient use case allows User to add, remove, or edit a client setting.

This use case has been deprecated, i.e. struck by lightning.

Use Case Name	ModifyClient
Participating Actors	• User
Flow of Events	1. → User selects a root Raincoat. Adding a client: a. ← Raincoat responds with the configuration options from ModifyRootRaincoat. b. → User selects and adds new client to selected root Raincoat. This client type is configured through the ConfigureCloud use case. c. ← If User does not cancel, Raincoat adds the client to the pending topology. OR 1. → User selects a client. 2. ← Raincoat responds with configuration details. Editing a client: a. → User enters client configuration options. b. ← If User does not cancel, Raincoat updates the pending topology to include the client option changes. Removing a client: a. ← Raincoat prompts User for confirmation. b. ← If User confirms, Raincoat updates the pending topology to remove the given client.
Entry Condition	• User must have participated in the ModifyTopology use case. • The ConfigureCloud use case must have successfully configured the selected root Raincoat.
Exit Condition	1. Client options for the modified clients have been saved to the pending topology. OR 2. No modifications will have been made to any topology.
Quality Requirements	1. Ensure that actions cannot be performed without authentication occurring. 2. Ensure that client options are valid for the cloud technology options set via the ModifyRootRaincoat use case.

Table 6. ModifyClient use case specifications.

6.2.3. `JoinTopology` Use Case

The `JoinTopology` use case allows `User` to add, remove, or edit a root `Raincoat` in the topology.

Use Case Name	`JoinTopology`
Participating Actors	• `User`
Flow of Events	1. → `User` connects to the root `Raincoat` server. Leaf `Raincoat` servers do not provide a connectable web interface. 2. ↔ `User` participates in the `Authenticate` use case. 3. ← `Raincoat` prompts user for topology connection details. 4. → `User` provides connection details. 5. ← Upon receiving valid connection details, `Raincoat` terminates connection and configures itself.
Entry Condition	• `User` must be connected to any root `Raincoat` instance through an Internet browser (not lynx).
Exit Condition	1. `User` will no longer be authenticated with the system and the session will be terminated. 2. `Raincoat` joins the overlord topology.
Quality Requirements	1. Ensure that actions cannot be performed without authentication occurring. 2. Ensure that `Raincoat` does not die in a huge fiery ball of death that is visible from space when it joins the topology.

Table 7. `JoinTopology` use case specifications.

6.3. Low Priority Use Cases

6.3.1. *`Authenticate` Use Case*

The `Authenticate` use case allows `User` to add, remove, or edit a root `Raincoat` in the topology.

Use Case Name	`Authenticate`
Participating Actors	• `User`
Flow of Events	1. ← `Raincoat` prompts user for credentials. 2. → `User` provides credentials. 3. ← `Raincoat` checks validity of credentials. 4. ← If credentials are valid, `Raincoat` authenticates `User`. 5. ← If credentials are not valid, `Raincoat` may terminate `User`'s session.
Entry Condition	• `User` must be participating in one of the `ViewTopology` or `JoinTopology` use cases.
Exit Condition	1. `User` will be authenticated with the system. OR 2. The session belonging to `User` will be terminated.
Quality Requirements	1. Do not let bad guys in.

Table 8. `Authenticate` use case specifications.

7. BEHAVIORAL ANALYSIS

In this section we present the sequence diagrams for the use cases. As will be seen, `Raincoat`s need to be able to send a number of messages which will originate from a single source and reach every Raincoat instance in the network. To handle these messages efficiently and simply, we use the following algorithm:

1. `Raincoat` receives a message that must be passed along.
2. `Raincoat` hashes the message and checks the hash against the list of hashes of messages it has already seen. As a part of this check, it unlists the hashes of all messages that are sufficiently old.
3. If the hash is in the list, `Raincoat` does nothing. Otherwise, `Raincoat` handles the message as necessary and sends it on to all neighbors that were not the sender.

7.1. `ViewTopology` Sequence Diagram

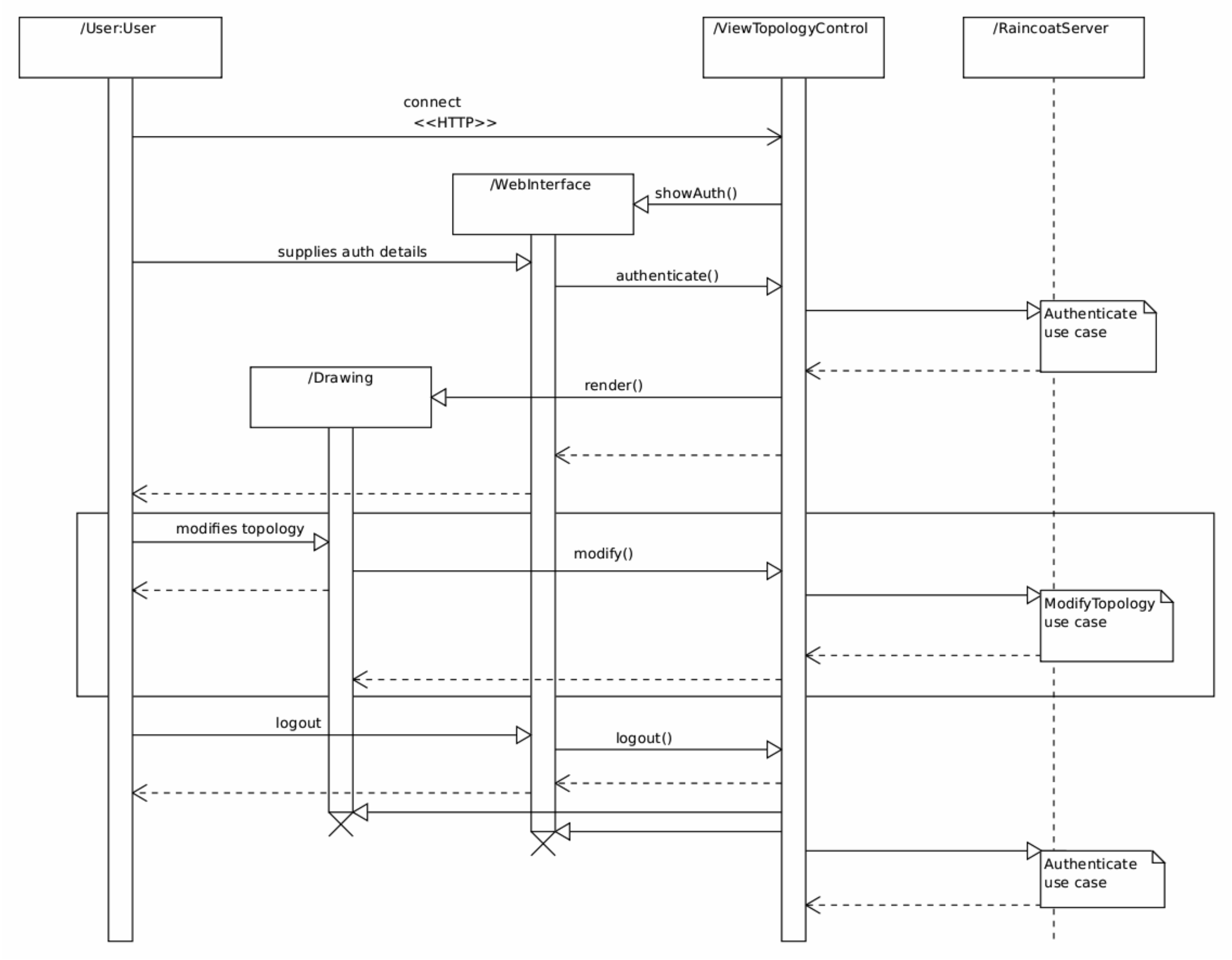

Figure 4. Sequence Diagram for the `ViewTopology` use case.

7.2. `ModifyTopology` Sequence Diagram

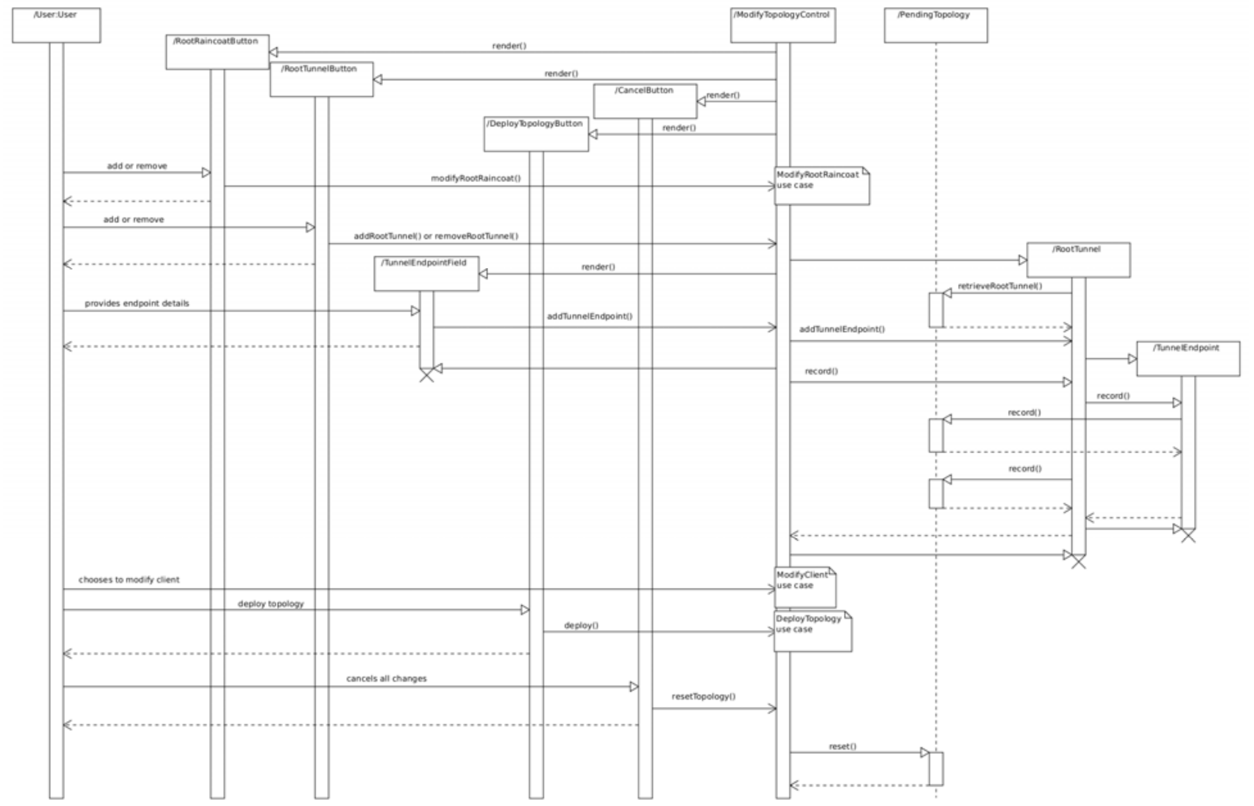

Figure 5. Sequence Diagram for the `ModifyTopology` use case.

7.3. `DeployTopology` Sequence Diagram

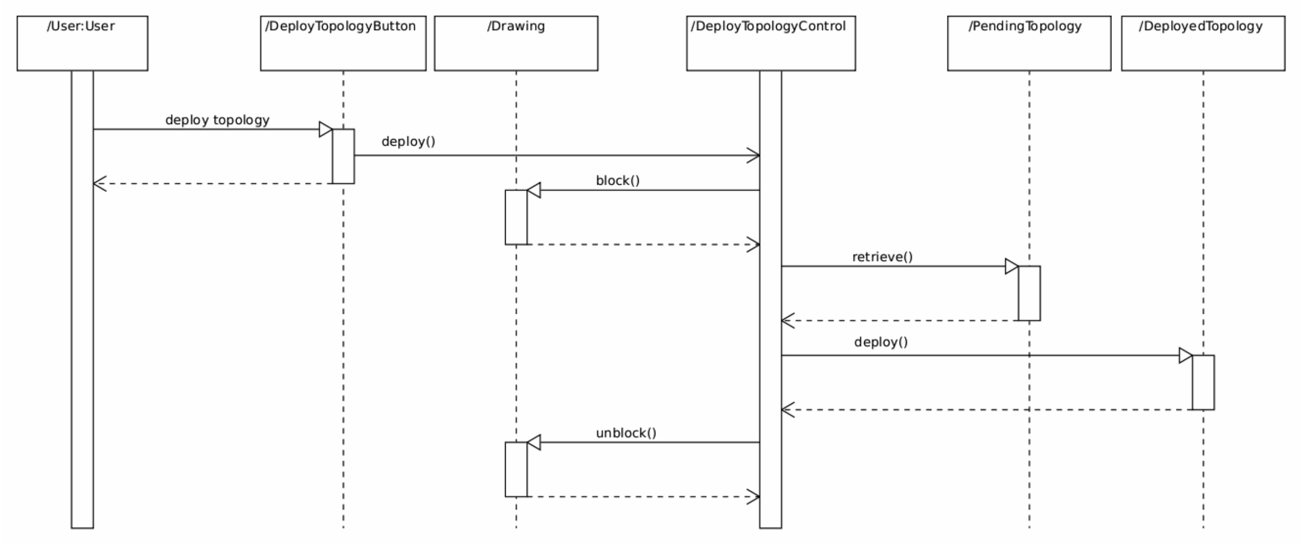

Figure 6. Sequence Diagram for the `DeployTopology` use case.

7.4. `ModifyRootRaincoat` Sequence Diagram

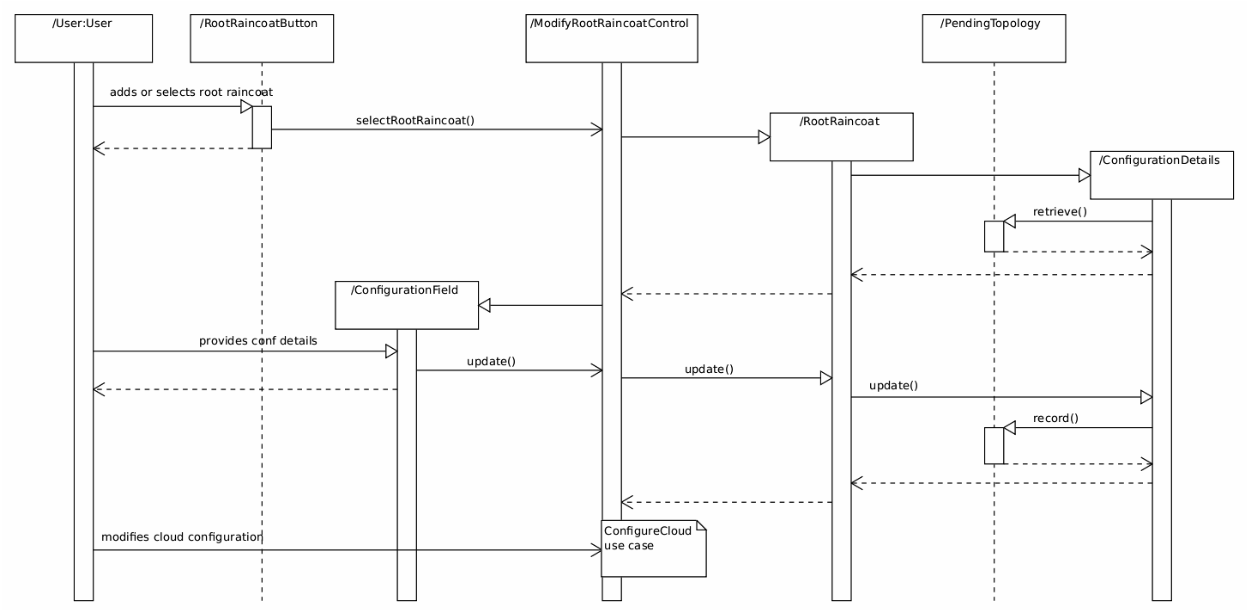

Figure 7. Sequence Diagram for the `ModifyRootRaincoat` use case.

7.5. `ConfigureCloud` Sequence Diagram

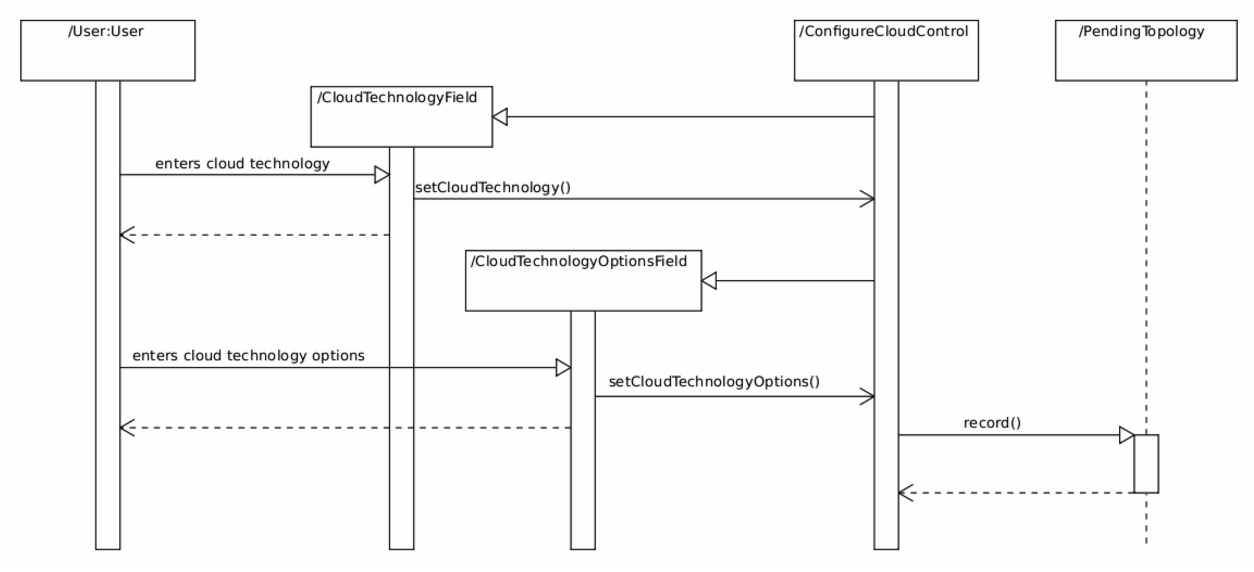

Figure 8. Sequence Diagram for the `ConfigureCloud` use case.

7.6. `ModifyClient` Sequence Diagram

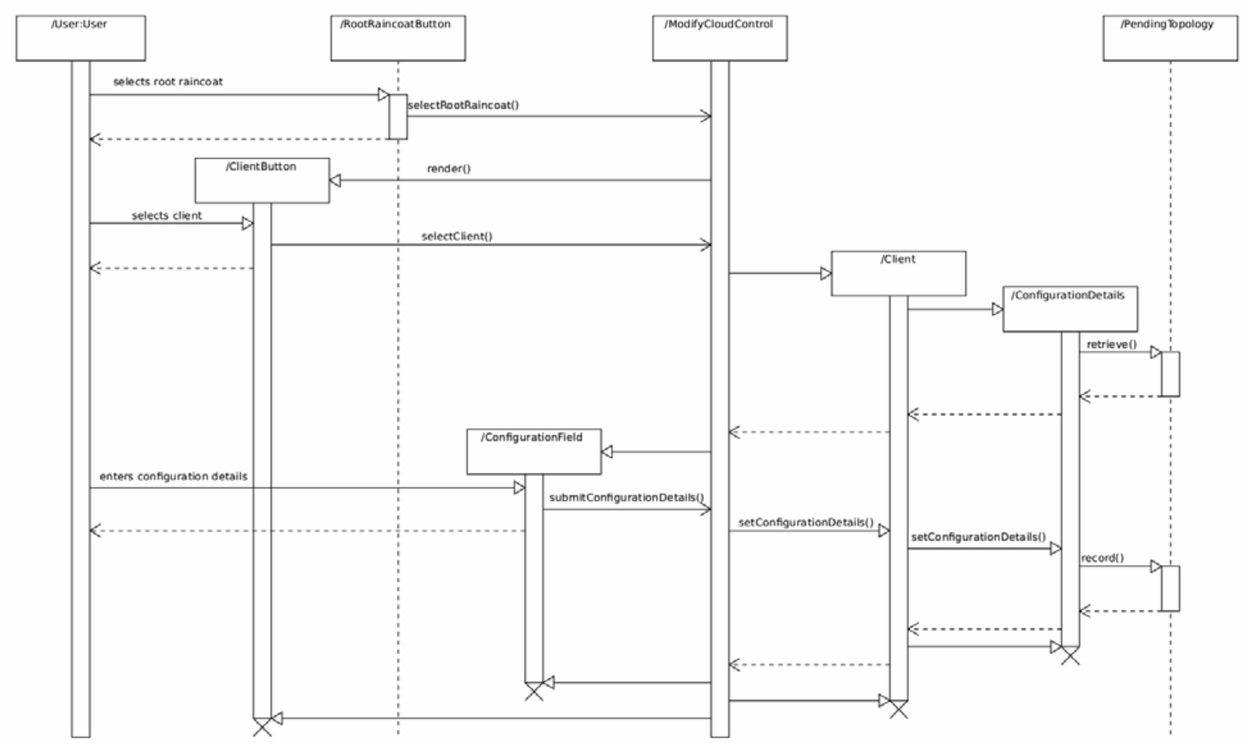

Figure 9. Sequence Diagram for the `ModifyClient` use case.

7.7. `JoinTopology` Sequence Diagram

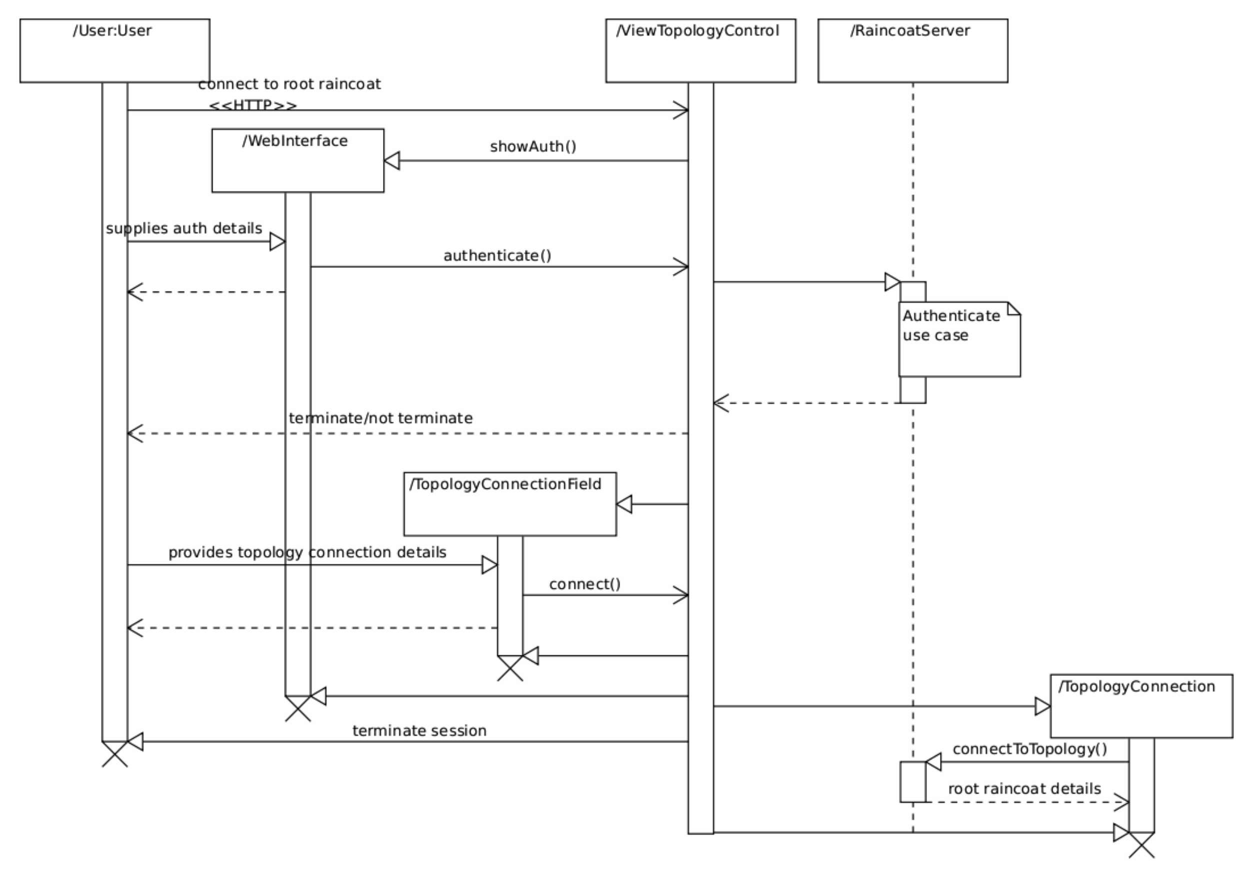

Figure 10. Sequence Diagram for the `JoinTopology` use case.

7.8. `Authenticate` Sequence Diagram

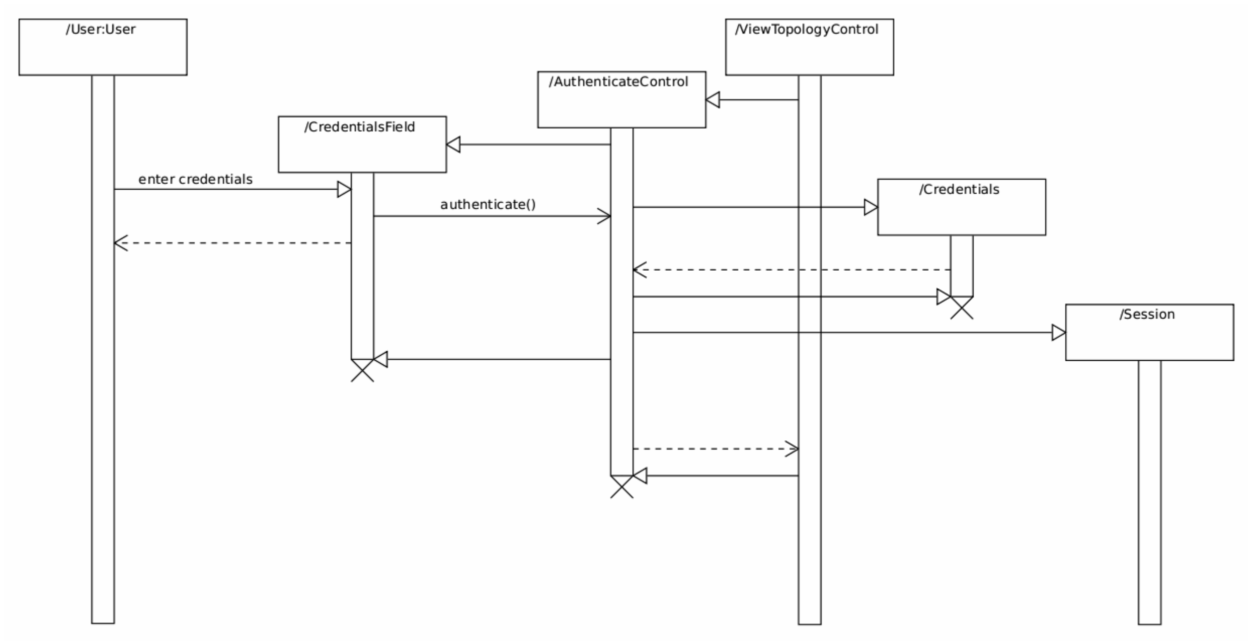

Figure 11. Sequence Diagram for the `Authenticate` use case.

8. STRUCTURAL ANALYSIS

Recall that we use here the model-view-controller (MVC) architecture for `Raincoat`. The *model* represents application data, and in this case it consists of a large database storing a collection of tables that represent all the data concerned with the overlay network. Each overlay consists of a single database, and when overlays merge, their respective databases are also merged. The *view* consists of components that interface with the user, such as web forms, and the *controller* contains the application logic that mediates between the view and the model.

8.1. Authentication

`AuthenticateView` is the interface component `Raincoat` will present to the user to prompt him for his credentials. The field consists of two text entries: one for the user to supply his name, and the other for him to supply his password.

`AuthenticateController` provides the `AuthenticateView` to prompt for user authentication information. When a user authenticates, he will provide his credentials, which will be verified by `AuthenticateController`. If the credentials are legitimate, `AuthenticateController` will indicate in the session that the user is authenticated. Otherwise, the user will have to be returned to the login screen so that he may try again.

The **`AuthenticateModel`** object is used by `AuthenticateController` to validate credentials provided by the user. The username and password do have a hardcoded default: username “administrator” and password “makeitrain”.

When running `Raincoat` on EC2 or Eucalyptus, the valid username and password should be able to be provided in the metadata passed to the current instance.

The **`Session`** object represents the user's session with `Raincoat`. It is represented by a username, token, and access time.

8.2. Cloud Technology

The **`CloudTechnologyView`** is the interface component `Raincoat` presents to the user to prompt him to select a cloud technology. `Raincoat` supports technologies based on the EC2 API: currently, Eucalyptus and Amazon EC2.

The **`CloudTechnologyController`** handles interactions with the user regarding configuration of cloud technologies. It provides the `CloudTechnologyView`. It also facilitates the submission of details from the above view.

The **`CloudTechnologyModel`** receives, validates, and stores information about cloud technology configuration information, such as SSH keys, user names, IP addresses, et cetera. Each `CloudTechnologyModel` will be linked to one and only one `RootRaincoatModel`.

8.3. Topology

The **`TopologyView`** is the interface element `Raincoat` presents to the user to display the topology. Topology information comes from the `TopologyModel`.

This object's functionality is complemented by the `ModifyTopologyView` and `DeployTopologyView` objects (they will be shown simultaneously).

The **`ModifyTopologyView`** is the interface component that the user uses to modify the existing cloud topology. The user will be presented with a page displaying a drawing of the current topology followed by various dialog boxes which contain elements that can be dragged into the drawing.

The **`DeployTopologyView`** is the interface component which the user may activate to tell `Raincoat` to deploy the topology. It replaces the `DeployedTopology` with the `PendingTopology`, ultimately implementing the user's requested changes.

The **`ConnectTopologyView`** is the interface component `Raincoat` presents to the user to prompt him for information about how to connect to an existing topology.

The **`TopologyController`** handles interactions with the user regarding modifications of the topology. It provides the `TopologyView`, `ModifyTopologyView`, `DeployTopologyView`, and `ConnectTopologyView`.

`TopologyModel` is simply a class which contains the `DeployedTopology` and the `PendingTopology` classes.

`PendingTopology` represents the working copy of the topology which the user is modifying before it is applied. It contains information necessary to set up each new `RootTunnel` and remove each old one.

`DeployedTopology` represents the current state of the actual network overlay. It contains information describing each active `RootTunnel`.

8.4. Root Raincoat

`ModifyRootRaincoatView` is the interface component `Raincoat` presents to the user to prompt him for `RootRaincoatModel` configuration options.

`RemoveRootRaincoatView` is the interface component `Raincoat` presents to the user to prompt him for confirmation to remove a root `Raincoat`.

The **`RootRaincoatController`** handles interaction with the user regarding the modification of root `Raincoat`s. It provides the `ModifyRootRaincoat`View and the `RemoveRootRaincoatView`.

The **`RootRaincoatModel`** is used by `RootRaincoatController` to receive, validate, and store root `Raincoat` configuration details provided by the user. It provides the `ModifyRootRaincoat`View and the `RemoveRootRaincoatView`.

8.5. RootTunnel

The **`RootTunnelView`** presents the user with the details pertaining to the selected `RootTunnel`.

The **`RemoveRootTunnelView`** is the interface that prompts the user for confirmation to remove a `RootTunnel` and provides the button to remove the `RootTunnel`.

The **`RootTunnelController`** controller handles interactions with the user regarding `RootTunnel`s. It provides the `RootTunnelView` and `RemoveRootTunnelView`.

The **`RootTunnelModel`** object is used by `RootTunnelController` to add, remove, and retrieve information about a given `RootTunnel`.

9. IMPLEMENTATION DETAILS

In this section we describe the various controllers, models, and views used within `Raincoat` as well as the helping library functions. This is the implementation for `Raincoat` v2.1 conforming to the above structural and behavioral analysis.

9.1. Controllers

9.1.1. Administrator Controller

The **administrator controller** is an authenticated controller representing the logic an administrator would need to manage an overlay network. Because the controller is authenticated, each request must be paired with a valid and complete session token. The URLs all take the following form: `/<token>/<controller>/<action>`

The administrator controller has the following actions:

`index` presents the administrator with the main page.
`logout` logs the given user out and broadcasts the logout.
`login` provides a form allowing the user to login.
`do_login` is called for a login. Given a valid username and password, this will initiate the login process for the given administrator. Once the login has been confirmed the user will be redirected to the index.
`deploy` is called when an administrator wants to deploy. This part of deploy needs to prepare the database for the new topology. This includes:

- Setting the instance root `Raincoat` to be the given root `Raincoat`.
- Deleting all root `Raincoat`s and tunnels to be deleted.
- Setting all pending tunnels to not pending tunnel.
- Creating the database dump to share.
- Setting the new topology id.
- Connecting all root `Raincoat`s.

`join` joins a new topology using a join code supplied by the user.
`wait` requests administrator acknowledgements again and shows the user logging in. This will be called when the user is in the process of logging in but the login is not yet complete. It will refresh every 4 seconds and request acknowledgements from all servers.

9.1.2. Inter- `Raincoat` Communications (IRC) Controller

The **Inter-Raincoat Communications (IRC) controller** is used to handle the message passing among the various `Raincoat` instances.

This is not an authenticated controller. Instead it facilitates message passing by means of the `__before__` and `__after__` Pylon functions. When a request is handled it is checked in `__before__` to ensure that it comes with a valid token and has not been seen before. The __after__ function will pass on a message to all neighboring Raincoat instances if appropriate.

The IRC controller has the following actions:

The heartbeat message carries the current status of the sender. It updates the saved status of the sender in the database.

acknowledge_administrator represents the sender acknowledging a root Raincoat's administrator. If the acknowledged root Raincoat has received all needed acknowledgements, the user may go about his business.

i_have_administrator represents the original sender attempting to log in a user. If this root has not already acknowledged a user, or if the sender root Raincoat has a lower ID number than the one already acknowledged, send back the i_have_administrator message, and create a new remote session for the user. Otherwise, do nothing.

i_lost_administrator represents the original sender reporting that it no longer has a user logged in. Destroy local sessions for that user.

deploy represents the sender saying that it has a new topology to deploy. Clean up any old dumps, get the topology from the sender, write it out to a new dump, and write the dump into the database. Update the topology's creation time, set my status to deploying, and spawn the deployment thread.

get_deploy represents the sender requesting the current dump of the topology for deployment. Send back a page containing the dump.

9.1.3. Root Raincoat

The authenticated **root Raincoat controller** facilitates user interaction with Raincoat objects. It allows the user to create, delete, and modify root Raincoat instances.

The root Raincoat controller has the following actions:

get_all returns a JavaScript Object Notation (JSON) list of display data for root Raincoats. This is used to populate the graph displaying all root Raincoats to the user. Since the user will click the corresponding node in the graph for more details, only the position and ID are required. Display data is a tuple: (id, x, y, overlay).

edit retrieves the edit form for a given root Raincoat. This will be used to display the form for modifying a root Raincoat. It will return a form allowing the user to modify the overlay IP and the max clients supported.

edit_form returns the form for editing a root Raincoat.

do_edit saves the submitted data to the database for a root Raincoat.

add creates a new root Raincoat at the given coordinates.

add_error is displayed when inputs to adding a new Raincoat are invalid.

move moves a given root Raincoat the new (x, y) coordinates.

move_error is displayed if moving inputs are not valid.

delete deletes a given root Raincoat from the system.

delete_error is displayed if delete inputs are not satisfied.

9.1.4. RootTunnel

The **`RootTunnel` controller** is an authenticated controller used to facilitate interactions with a given `RootTunnel`.

The `RootTunnel` controller has the following actions:

`get_all` returns a JSON list of display data for the represented `RootTunnel`. This is used to render all of the `RootTunnel`s connecting the root `Raincoat`s. It returns a list of arrays containing the start and end of the given tunnel.

`add` adds a new root `Raincoat` to the system.
Restrictions on adding a new tunnel include:

- A tunnel must be between distinct root `Raincoat`s.
- A tunnel, even if reversed, cannot already exist.
- Each endpoint must be a valid `Raincoat`.

Since the first two restrictions are difficult to check with a validator, this method will just use an old-school approach for those condition.

`add_form` is the error form for adding a `RootTunnel`.

9.2. Models

9.2.1. SQLAlchemy Tables

A **`RootRaincoatTable`** represents a root `Raincoat` in the given topology. A root `Raincoat` is a `Raincoat` instance which may contain an IPSec tunnel and will support up to a class B of clients by means of leaf `Raincoat`s.
`RootRaincoatTable`s contain:

- `id`, an integer unique to the `Raincoat`.
- `token`, a 32 byte string containing the passphrase for the root `Raincoat` to use to communicate.
- `overlay_1`, the first digits for the overlay IP address. This is `xxx` in `xxx.yyy.0.0` for the user-assigned class B.
- `overlay_2`, the second digits for the overlay IP address. This is the `yyy` in `yyy.xxx.0.0`.
- `x`, the x-position for the center of the root `Raincoat` on the canvas.
- `y`, the y-position for the center of the root `Raincoat` on the canvas.
- `max_clients`, an integer describing the maximum number of clients supported by the given `Raincoat`.
- `connected_clients`, an integer describing the number of clients connected to the given root `Raincoat`.
- `internet_ip`, the address in the form of a string that other `Raincoat`s will use to connect to the given `Raincoat`.

- `status`, a string describing the current state of the `Raincoat`. Status may be one of:
    - `deploying`
    - `pending`
    - `alive`
    - `down`
- `last_heartbeat`, a date and time describing when the last heartbeat was received from the given `Raincoat`.
- `ca_key`, a string containing the certificate authority key.
- `ca_crt`, a string containing the certificate authority certificate.
- `dh_pem`, a string containing the dh1024 key file.
- `server_key`, a string containing the server key for the root `Raincoat`.
- `server_crt`, a string containing the signed certificate file for the `Raincoat`.
- `need_ack`, a boolean describing whether the root `Raincoat` has responded to the login request. If true, then the root `Raincoat` has not responded but needs to.
- `deleted`, a boolean describing whether the `Raincoat` has been deleted. This serves as a soft delete until the topology has been deployed.

A **`RootTunnelTable`** represents an IPSec tunnel between two root `Raincoat`s. A `RootTunnel` is an object which will become an IPSec site-to-site tunnel in Vyatta.
A `RootTunnel` contains:

- `start`, the first root `Raincoat` participating in the tunnel.
- `end`, the second root `Raincoat` participating in the tunnel.
- `key`, the secret key used to encrypt data over the tunnel.
- `pending`, a boolean describing whether the tunnel has been modified since it has been deployed.
- `deleted`, a boolean to demonstrate that a pending tunnel has been soft deleted.

The **`LeafRaincoatTable`** contains the leaf `Raincoat`s known in a given cloud. A leaf `Raincoat` is a `Raincoat` which allows clients to connect to it and provides routing to the remainder of the cloud but does not participate in any IPSec tunnels.
The `LeafRaincoatTable` contains:

- `id`, a unique integer within a cloud representing the leaf `Raincoat`.
- `token`, a 32 character string containing the passphrase for the root `Raincoat` to use to communicate.
- `server_crt`, the certificate file for the leaf OpenVPN server.
- `server_key`, the key file for the OpenVPN server.
- `client_crt`, the OpenVPN client certificate file.
- `client_key`, the OpenVPN client key file.
- `overlay_ip`, the class C network that this leaf `Raincoat` will provide.
- `internet_ip`, the internet reachable IP address to connect this leaf `Raincoat` to the root `Raincoat`.

- `status`, a string describing the state of a given leaf `Raincoat`. Status may be one of:
    - `deploying`
    - `pending`
    - `alive`
    - `down`
- `clients_connected`, an integer containing the number of connected clients to the OpenVPN server.

The **`TopologyTable`** contains the settings specific to a given `Raincoat` instance.
The `TopologyTable` contains:

- `root_id`, the id of the current `Raincoat`, or the `Raincoat` the given leaf is attached to.
- `pending_root_id`, the id that will be set as the current `Raincoat` upon deployment.
- `leaf_id`, the id of the leaf `Raincoat`, or null if the current `Raincoat` is a root `Raincoat`.
- `deploying`, a boolean describing whether the given `Raincoat` is in the process of deploying itself.
- `next_message_id`, an integer containing the id of the next message. The id will wrap around after reaching 1000000, returning to 1. The next message id is only a component of full id of the next message: the full id will be of the form `Raincoat` _id.next_message_id locale, a string containing the locale for the given `Raincoat`.
- `current_topology_id`, a date/time unique to the given topology. The date and time are set to the current time of the start process of deployment.
- `current_topology_location`, a string containing the location on disk of a database dump of the current topology.

The **`UserSessionTable`** contains the information about a logged in user. If a user is logged in, either locally or remotely, then this table will contain an entry. A lowercase token is valid, an uppercase token means that the login is remote and any attempts at logging in should be denied.
The `UserSessionTable` contains:

- `token`, a string containing the login token.
- `root_Raincoat`, an integer referencing the root `Raincoat` to which the login is attached.
- `last_access`, a date and time containing the last time the user has issued a command.
- `complete`, a boolean describing whether the login session has been completely acknowledged.

The **`MessageTable`** contains all of the message ids of recently seen messages. When a message id is checked, any message older than 10 minutes is removed.
The `MessageTable` contains:

- `id`, a string containing the message id.
- `receive_time`, a date and time containing the time the message was received.

The **`LoginTable`** contains the login username and password for the user. This should be set when the instance is loaded.
The `LoginTable` contains:

- `username`, a string containing the username of the administrator.
- `password`, a string containing the SHA1 hash of the administrator's password.

9.2.2. Forms

`Pylon`'s `formencode` was used to validate all inputs to the given forms. A form exists for each controller accepting inputs in the authenticated controllers.

9.3. Views

This section describes the various views that are present in `Raincoat`. The HTML5 interface for `Raincoat` is described in a separate section.

9.3.1. Common Views

These views are used throughout several other views and are not useful for direct display to the end user. The common views present in `Raincoat` are as follows:

`ajax` is a wrapper for all `ajax` pages returned.

`flash` is used to display any flash messages to the user. A flash message is one which should only be displayed once to a given user and, on any subsequent pages, should not be displayed.

`footer` contains the footer describing the copyright, license, et cetera.

`title` contains both the title and subtitle of a `Raincoat` page.

9.3.2. Administrator

The **administrator views** are presented to the user for a given administrator action:

`index` is the page that contains the HTML5 interface with which the administrator interacts with `Raincoat`.

`login` will present the login form to the user, prompting him for a username and password.

`wait` shows a status bar and will refresh itself after 4 seconds, demonstrating to an administrator trying to login that the topology is preparing itself for the given administrator's login.

9.3.3. Root `Raincoat`

The **root `Raincoat`** contains the forms which will be returned to the user for modification of root `Raincoat`s:

- **`edit`** contains the edit form that allows the user to modify the settings for a root `Raincoat`.

9.4. Libraries

Here we outline the various library functions created to allow `Raincoat` to interact with Vyatta.

9.4.1. Deploy

`Deploy` is a library which will turn the `Raincoat` database descriptions into a working Vyatta configuration. It contains the following functions:

- `deploy` performs actions required to turn the pending items into deployable items.
  This should be run on the `Raincoat` server on which the `deploy` action is started (i.e. on which the `Deploy` button is pressed). It will generate keys and do anything else needed so that when the other connected root `Raincoat`s receive a copy of the database they will be able to generate their own configurations.
  For each pending the root `Raincoat`, `deploy` generates the CA keys as well as the OpenVPN server keys.
  For each pending tunnel, `deploy` generates a 60 character secret to be used as a pre-shared-key in the VPN tunnel.
- `received_deploy_msg` performs actions needed once a root has received and loaded a database dump. Once a root `Raincoat` receives the `deploy` message and has loaded the database, it needs to generate server keys for every leaf `Raincoat` that can connect to it and populate the `LeafRaincoatTable` with them, and it needs to generate and start running the topology configuration.
- `generate_root_config` generates the configuration files for a specified root `Raincoat`. This function generates the Vyatta configuration for the root `Raincoat` according to the tunnels and information present in the database.
  Additionally, it generates the OpenVPN configuration files (client configurations, server configuration, et cetera) needed for the root `Raincoat` to be an OpenVPN server.
- `generate_leaf_config` generates and deploys a leaf configuration. This function generates a leaf configuration based on the leaf and root ID passed to it. It will create the configuration for the specified leaf and treat it as a child of the specified root ID.
  The root is needed so we can grab the public IP address and overlay/tunnel information from it. If called on a leaf, this information should be available in the `TopologyTable`.

9.4.2 Rubber

`Rubber` is the part of `Raincoat` which interfaces with `easyrsa` (our home-grown API that wraps key generation) to generate the RSA keys and with the core parts of Vyatta to configure Vyatta as `Raincoat` demands.

9.4.2.1. Config

The **`Rubber config`** file contains the objects which are used to build an arbitrary Vyatta configuration. It has two classes: VyattaConfig and VyattaObject.

- **`VyattaConfig`** provides `generate` to generate the Vyatta configuration file based on the `VyattaObject`s this class contains.
  The load option is used to specify whether to start running the configuration. It first makes a backup of `/opt/vyatta/etc/config/config.boot` and then places the new configuration options at the end of the file.
- **`VyattaObject`** represents an element in the Vyatta configuration file. It has a title and optionally an identifier and can have other configuration objects (`VyattaObject`s) appended to it as children to build a configuration tree. A configuration tree can then be output as a Vyatta configuration. It provides the following functions:
  - `append_object` appends another VyattaObject to this one as a child. It can replace any existing object that has the same configuration title as the one being added. For example, one could add a new ‘vpn’ object to replace all current ‘vpn’ children.
  - `option` currently supports taking a list of values for the option as well as a quote property that can be set to True:
    `option(‘remote-ip’, ‘10.5.5.101’, quote=True)`
  - `onfiguration` generates a configuration that looks like the following:

```
            name (identifier) {
                  child1
                  child2
                  child3
            }     //Identifier is optional
```

9.4.2.2. IPSec

`IPSec` contains the SiteToSite object which is a specific `VyattaObject` class. It defines all options necessary to create a site-to-site VPN tunnel.

9.4.2.3. OpenVPN

Similar to `IPSec`, `OpenVPN` contains configuration data for the various clients and servers required by `Raincoat` for the OpenVPN configuration. It also defines and creates the OpenVPN-specific configuration file which is only referenced by Vyatta.

9.4.2.4. RSA

The RSA library file contains all of the options required to generate and parse the RSA keys required for OpenVPN. It creates and loads certificate authorities, DH parameters, server key files and client key files.

10. CONCLUSIONS

We are about ready to release the `Raincoat` 2 series, which can be used to easily configure and deploy various overlay networks within a cloud environment. Furthermore, it can be used by individuals with varying degrees of exposure to VPN technologies and Vyatta routers: the user interface is a unique new interface allowing drag-and-drop technologies to be used for configuring overlay networks, and it does not require the technological know-how required for similar systems. Additionally, `Raincoat` can now scale to very large numbers of clients across numerous clouds.

REFERENCES

1. OpenVPN Documentation. http://openvpn.net/index.php/open-source/documentation.html.

2. Vyatta Core Documentation. “Vyatta Basic System Reference Guide.” http://www.vyatta.com/downloads/documentation.php.

DISTRIBUTION

1	MS 0549	Richard Neiser	5918 (1 electronic copy)
1	MS 0621	Tan Thai	5630 (1 electronic copy)
1	MS 0621	Roxana Jansma	5631 (1 electronic copy)
1	MS 0621	LeAnn Miller	5636 (1 electronic copy)
1	MS 0621	Michelle Leger	5636 (1 electronic copy)
1	MS 0899	Technical Library	9536 (1 electronic copy)
1	MS0123	D. Chavez, LDRD Office	1011 (1 electronic copy)

Sandia National Laboratories