Chapter 1

Manual to replicate our experiments and to use the testing framework

This chapter describes how to reproduce the results of this thesis. The developed code, along with other necessary files, is available through the provided GitHub link Access_Thesis_on_Github. Additionally, we offer an OVF image containing all code, dependencies, and scripts prepared for immediate execution. Alternatively, you can set up your environment by cloning the main directories from the repository.

1.1 OVF Image Route To Reproduce The Results

The provided OVF image is based on Ubuntu, with the password for all users set as `uwbothell`. Within the directory `/home/anthony/`, you will find two directories containing all the necessary code, dependencies, and scripts to conduct the tests discussed in this thesis.

1.1.1 HTTP_TEST Directory

This directory includes three subdirectories corresponding to the three different implementations explored in the thesis.

Figure 1.1: Overview of HTTP project directory structure

1.1.2 Testing The P4runtime-based Implementation (Directory p4runtime_based_implementation)

The process to test the P4runtime_based implementation involves several steps:

1. Run the bash script `create_architecture.sh` to set up the network architecture for the tests.
2. Upon executing `create_architecture.sh`, a Mininet console will appear. Use the command `xterm h1 h2` to open terminal windows for hosts h1 and h2.
3. In each console, disable receive and transmit offload using: `ethtool --offload h1-eth0 rx off tx off` and similarly for `h2-eth0`.
4. On h2's console, start the HTTP server with: `python3 http_dpi_server.py`.

5. Execute `run_controller.sh` to start the p4runtime controller that contains the HTTP filtering application as dependency to filter out domains.

6. On h1's console, execute `benchmarking.sh` to run the tests.

7. Once `benchmarking.sh` completes, collect the results for accuracy, delay, CPU consumption, memory consumption, etc.

Note: To exist the p4_runtime controller you must press "s", be careful not to press it if you do not want to exit.

```
root@anthony-virtual-machine:/home/anthony/HTTP_TEST/p4runtime_based_implementation# ls
app.log                 http_dpi_client.py  p4-guide                      ss-log.1.txt
benchmarking.sh         http_dpi.py         p4runtime_metric_results.txt  ss-log.2.txt
create_architecture.sh  http_dpi_server.py  p4runtime_test.csv            ss-log.3.txt
http_client.py          http_server.py      run_controller.sh             ss-log.txt
```

Figure 1.2: Needed files for the P4runtime-based Implementation

The results will look like figure 1.3

```
Accuracy and Delay Results:

Accuracy: 0.99
Precision: 1.00
Recall: 0.99
F1 Score: 0.99
True Positive Rate (TPR): 1.00
False Positive Rate (FPR): 0.01
Minimum Delay: 49.77 ms
Maximum Delay: 1175.79 ms
Average Delay: 133.87 ms

----------------------------
Execution Time: 491242ms
Peak CPU Usage During Execution: 57%
CPU Usage Increment Factor: 57.00
CPU Time: 2.44s
Maximum Memory Usage: 99072KB
----------------------------
```

Figure 1.3: P4runtime_based implementation results

1.1.3 Testing The Openflow-based Implementation (Directory openflow_based_implementation)

The process to test the Openflow_based implementation involves several steps:

1. Execute `run_controller.sh` to start the POX controller that contains the HTTP filtering application as a dependency to filter out domains.

2. Run the bash script `create_architecture.sh` to set up the network architecture for the tests.

3. Upon executing `create_architecture.sh`, a Mininet console will appear. Use the command `xterm h1 h2` to open terminal windows for hosts h1 and h2.

4. On h2's console, start the HTTP server with: `python3 http_dpi_server.py`.

5. On h1's console, execute `benchmarking.sh` to run the tests.

6. Once `benchmarking.sh` completes, collect the results for accuracy, delay, CPU consumption, memory consumption, etc.

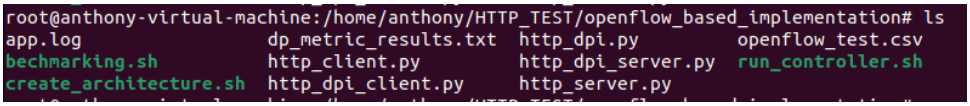

Figure 1.4: Needed files for the OpenFlow-based Implementation

The results will look like figure 1.5

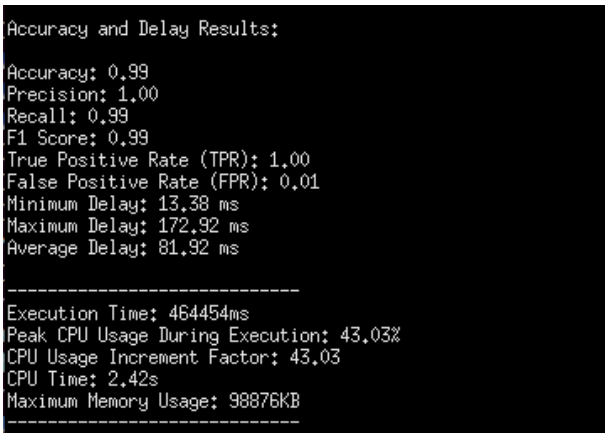

Figure 1.5: Openflow_based implementation results

1.1.4 Testing The Data Plane Implementation (Directory data_plane_implementation)

The process to test the Data Plane implementation involves several steps:

1. Run the bash script `create_architecture.sh` to set up the network architecture for the tests.
2. Execute `architecture_dependencies.sh` to connect interfaces to the network and add necessary table entries for the BMv2 switch.
3. Upon executing `create_architecture.sh`, a Mininet console will appear. Use the command `xterm h1 h2` to open terminal windows for hosts h1 and h2.
4. In each console, disable receive and transmit offload using: `ethtool --offload h1-eth0 rx off tx off` and similarly for `h2-eth0`.
5. On h2's console, start the HTTP server with: `python3 http_dpi_server.py`.
6. On h1's console, execute `benchmarking.sh` to run the tests.
7. Once `benchmarking.sh` completes, collect the results for accuracy, delay, CPU consumption, memory consumption, etc.

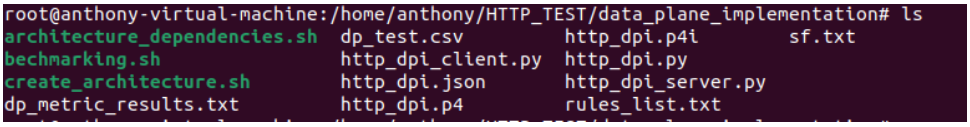

Figure 1.6: Needed files for the Data Plane Implementation

The results will look like figure 1.7

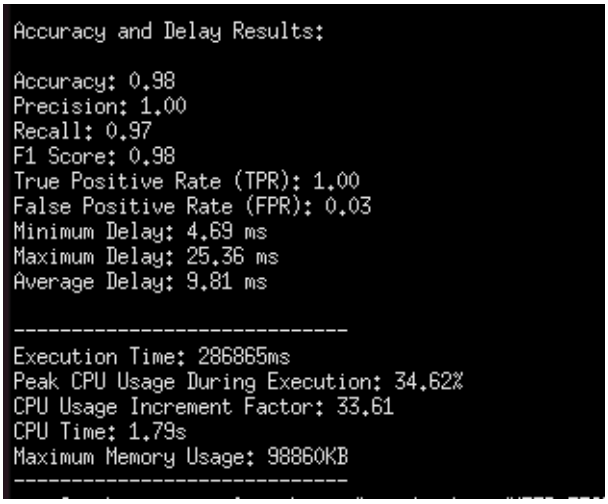

Figure 1.7: Data Plane Implementation Results

1.1.5 SQL_TEST Directory

This directory includes three subdirectories corresponding to the three different implementations explored in the thesis.

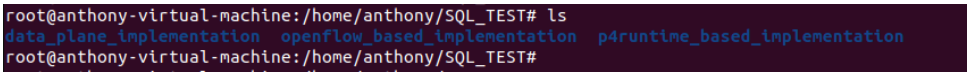

Figure 1.8: Overview of SQL project directory structure

1.1.6 Testing The P4runtime-based Implementation (Directory p4runtime_based_implementation)

The process to test the P4runtime$_{b}asedimplementationinvolvesseveralsteps$:

1. Run the bash script `create_architecture.sh` to set up the network architecture for the tests.
2. Upon executing `create_architecture.sh`, a Mininet console will appear. Use the command `xterm h1` to open terminal windows for hosts h1. Note: In this step, we are not opening the h2 console, because we are connecting our BMv2 switch to a VMware machine that is hosting the MySQL Database, we could use any other solution to host our database like a real machine or docker.
3. In h1 console, disable receive and transmit offload using: `ethtool --offload h1-eth0 rx off tx off`.
4. Execute `run_controller.sh` to start the p4runtime controller that contains the SQL filtering application as a dependency to filter out domains.
5. On h1's console, execute `benchmarking.sh` to run the tests.
6. Once `benchmarking.sh` completes, collect the results for accuracy, delay, CPU consumption, memory consumption, etc.

Note: To exist the p4_runtime controller you must press "s", be careful not to press it if you do not want to exit. Additionally to collect the results for accuracy and delay you have to read the .csv files

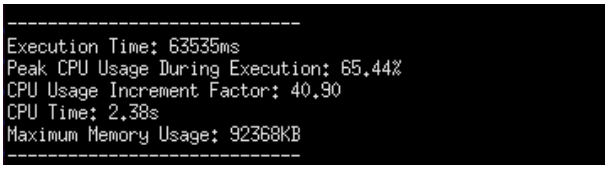

Figure 1.10: Enter Caption

"accuracy_sql.csv" and "command_response_times.csv", make sure to not have these two .csv files before executing the script `benchmarking.sh`.

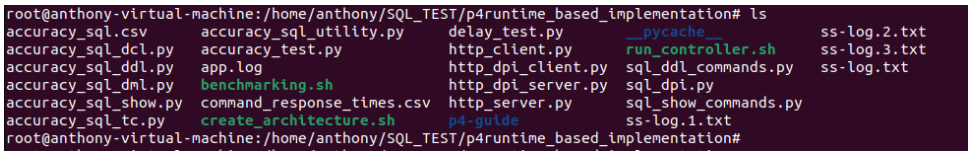

Figure 1.9: Needed files for the P4runtime-based Implementation

The results will look like figure **??**

1.1.7 Testing The Openflow-based Implementation (Directory openflow_based_implementation)

The process to test the Openflow$_{b}asedimplementationinvolvesseveralsteps$:

1. Execute `run_controller.sh` to start the POX controller that contains the SQL filtering application as dependency to filter out domains.
2. Run the bash script `create_architecture.sh` to set up the network architecture for the tests.
3. Then execute the command `ovs-vsctl add-port s1 ens37` that is going to add the interface ens37 to the OpenV Switch, you should add whichever interfaces connecting your Database instead.
4. Upon executing `create_architecture.sh`, a Mininet console will appear. Use the command `xterm h1` to open terminal windows for hosts h1. Note: In this step, we are not opening the h2 console, because we are connecting our Openv Switch to a VMware machine that is hosting the MySQL Database, we could use any other solution to host our database like a real machine or docker.
5. On h1's console, execute `benchmarking.sh` to run the tests.
6. Once `benchmarking.sh` completes, collect the results for accuracy, delay, CPU consumption, memory consumption, etc.

Note: To collect the results for accuracy and delay you have to read the .csv file "accuracy_sql.csv" and "command_response_times.csv", make sure to not have these two .csv files before executing the script `benchmarking.sh`.

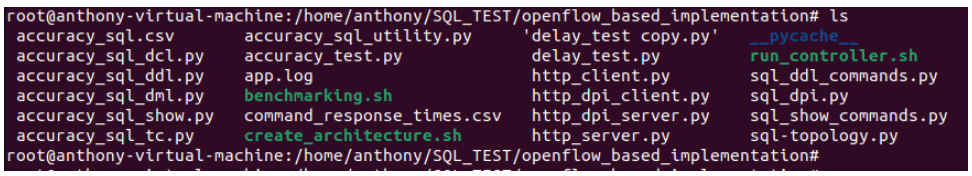

Figure 1.11: Needed files for the OpenFlow-based Implementation

The results will look like figure 1.12

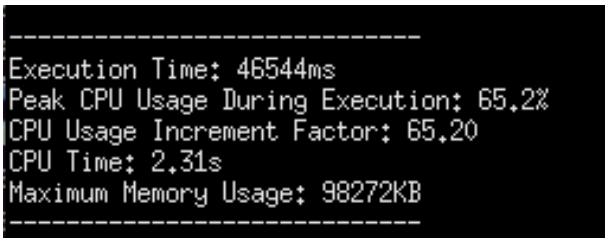

Figure 1.12: Openflow_based implementation results

1.1.8 Testing The Data Plane Implementation (Directory data_plane_implementation)

The process to test the Data Plane implementation involves several steps:

1. Run the bash script `create_architecture.sh` to set up the network architecture for the tests.
2. Execute `architecture_dependencies.sh` to connect interfaces to the network and add necessary table entries for the BMv2 switch.
3. Upon executing `create_architecture.sh`, a Mininet console will appear. Use the command `xterm h1` to open terminal windows for hosts h1.
4. In the console, disable receive and transmit offload using: `ethtool --offload h1-eth0 rx off tx off`. Note: In this step, we are not opening the h2 console, because we are connecting our BMv2 switch to a VMware machine that is hosting the MySQL Database, we could use any other solution to host our database like a real machine or docker.
5. On the virtual machine we have running our MySQL database.
6. On h1's console, execute `benchmarking.sh` to run the tests.
7. Once `benchmarking.sh` completes, collect the results for accuracy, delay, CPU consumption, memory consumption, etc.

Note: To collect the results for accuracy and delay you have to read the .csv file "accuracy_sql.csv" and "command_response_times.csv", make sure to not have these two .csv files before executing the script `benchmarking.sh`.

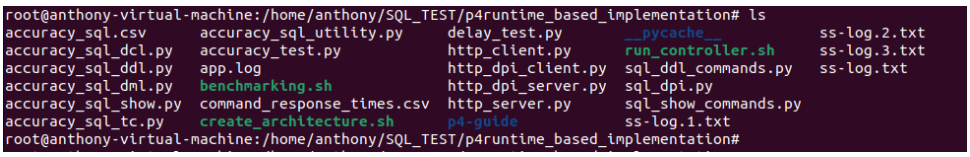

Figure 1.13: Needed files for the Data Plane Implementation

The results will look like figure 1.14

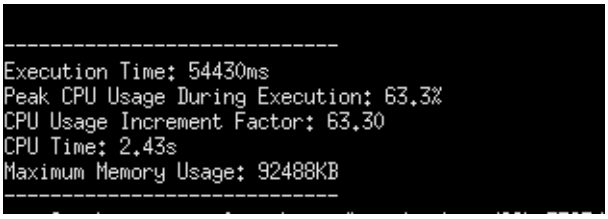

Figure 1.14: Data Plane Implementation Results

1.1.9 About MySQL Database

The Database that we used for out tests is not in the provided OVF image, therefore, to execute the SQL tests successfully we will have to import the dump file that contains the squema, tables, etc. in regards to the DB. Besides, when importing the database make sure to set up this information:

1. 'user': 'username'
2. 'password': 'password'
3. 'host': '192.168.118.136' (IP of the host containing the DB)
4. 'database': 'sqli'

If you decide to opt for configuring different username, password, host, or database name, you will have to update the parameters in the python programs. Also, note that the MySQL Database is not encrypting the communication. Hence, you will have to disable SSL/TLS for the tests.

Appendix A: Database Schema Dump

Below is the complete MySQL dump from the database 'sqli', which includes detailed table structures, data insertion commands, and system settings. This dump is crucial for reproducing the database environment necessary for the experiments conducted in this thesis. Note: You can download the file from the provided GitHub repository.

```
-- MySQL dump 10.13 Distrib 8.0.36, for Linux (x86_64)
-- Host: localhost Database: sqli
-- ------------------------------------------------------
-- Server version 8.0.36-0ubuntu0.22.04.1

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!50503 SET NAMES utf8mb4 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

-- Table structure for table `coffee_table`
DROP TABLE IF EXISTS `coffee_table`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
```

```
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `coffee_table` (
  `id` int DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `region` varchar(255) DEFAULT NULL,
  `roast` varchar(255) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

-- Dumping data for table `coffee_table`
LOCK TABLES `coffee_table` WRITE;
/*!40000 ALTER TABLE `coffee_table` DISABLE KEYS */;
INSERT INTO `coffee_table` VALUES
(1, 'default route', 'Ethiopia', 'light'),
(1, 'Colombian', 'South America', 'Medium'),
(2, 'Ethiopian Yirgacheffe', 'Africa', 'Light'),
(3, 'Sumatra Mandheling', 'Indonesia', 'Dark'),
(4, 'Guatemala Antigua', 'Central America', 'Medium'),
(5, 'Kenya AA', 'Africa', 'Medium'),
(6, 'Costa Rica Tarrazu', 'Central America', 'Medium');
/*!40000 ALTER TABLE `coffee_table` ENABLE KEYS */;
UNLOCK TABLES;

-- Table structure for table `countries`
DROP TABLE IF EXISTS `countries`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `countries` (
  `id` int NOT NULL AUTO_INCREMENT,
  `country_name` varchar(255) DEFAULT NULL,
  `capital` varchar(255) DEFAULT NULL,
  `government_type` varchar(255) DEFAULT NULL,
  `population` int DEFAULT NULL,
  `area_km2` decimal(10,2) DEFAULT NULL,
  `currency` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
    ↪ ;
/*!40101 SET character_set_client = @saved_cs_client */;

-- Dumping data for table `countries`
LOCK TABLES `countries` WRITE;
/*!40000 ALTER TABLE `countries` DISABLE KEYS */;
INSERT INTO `countries` VALUES
(1, 'United States', 'Washington D.C.', 'Federal Republic', 331002651, 9833517.85,
    ↪ 'USD'),
(2, 'United Kingdom', 'London', 'Constitutional Monarchy', 67886011, 242495.00, '
    ↪ GBP'),
(3, 'France', 'Paris', 'Semi-Presidential Republic', 65273511, 551695.00, 'EUR'),
(4, 'Germany', 'Berlin', 'Federal Republic', 83783942, 357022.00, 'EUR'),
(5, 'Japan', 'Tokyo', 'Constitutional Monarchy', 126476461, 377975.00, 'JPY');
```

```
/*!40000 ALTER TABLE `countries` ENABLE KEYS */;
UNLOCK TABLES;

-- Table structure for table `dpi2`
DROP TABLE IF EXISTS `dpi2`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `dpi2` (
  `id` int NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
/*!40101 SET character_set_client = @saved_cs_client */;

-- Dumping data for table `dpi2`
LOCK TABLES `dpi2` WRITE;
/*!40000 ALTER TABLE `dpi2` DISABLE KEYS */;
/*!40000 ALTER TABLE `dpi2` ENABLE KEYS */;
UNLOCK TABLES;

-- Table structure for table `test1`
DROP TABLE IF EXISTS `test1`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!50503 SET character_set_client = utf8mb4 */;
CREATE TABLE `test1` (
  `id` int NOT NULL AUTO_INCREMENT,
  `column1` varchar(255) DEFAULT NULL,
  `column2` varchar(255) DEFAULT NULL,
  `column3` varchar(255) DEFAULT NULL,
  `column4` varchar(255) DEFAULT NULL,
  `column5` varchar(255) DEFAULT NULL,
  `column6` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
    ↪ ;
/*!40101 SET character_set_client = @saved_cs_client */;

-- Dumping data for table `test1`
LOCK TABLES `test1` WRITE;
/*!40000 ALTER TABLE `test1` DISABLE KEYS */;
INSERT INTO `test1` VALUES
(1, 'data1_1', 'data1_2', 'data1_3', 'data1_4', 'data1_5', 'data1_6'),
(2, 'data2_1', 'data2_2', 'data2_3', 'data2_4', 'data2_5', 'data2_6'),
(3, 'data3_1', 'data3_2', 'data3_3', 'data3_4', 'data3_5', 'data3_6'),
(4, 'data4_1', 'data4_2', 'data4_3', 'data4_4', 'data4_5', 'data4_6'),
(5, 'data5_1', 'data5_2', 'data5_3', 'data5_4', 'data5_5', 'data5_6');
/*!40000 ALTER TABLE `test1` ENABLE KEYS */;
UNLOCK TABLES;

-- Footer settings to restore system settings
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
```

```
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2024-05-19 19:39:03
```

1.2 Create Your Own Environment To Reproduce The Results

If on the contrary you don't want to use the provided OVF image, you can also set up your own environment.

Software Installation Guide

These are some of the things to install to have the adequate environment for the tests.

1. **Basic Tools Installation**
   - Command: `sudo apt install git wget vim ethtool`
2. **Mininet Installation**
   - GitHub: https://github.com/mininet/mininet
3. **POX Controller Installation**
4. **Thrift Installation**
   - Required for BMv2.
   - Installation guide: Install Thrift
5. **BMv2 (Behavioral Model v2) Installation**
   - Necessary for running P4 environments.
   - GitHub: https://github.com/p4lang/behavioral-model
6. **gRPC and Protobuf Installation**
   - Needed for communication between P4 devices and controllers.
   - Protobuf GitHub: https://github.com/protocolbuffers/protobuf
   - gRPC GitHub: https://github.com/grpc/grpc
   - Installation steps are available on their respective GitHub repositories.
7. **Docker Installation**
   - For container management.
   - Installation guide: Install Docker
8. **Additional Commands and Software**
   - Configure and manage SDN controllers, switches, and environments using provided scripts and command-line utilities.

Once we have out environment we will have to clone the provided GitHub repository GitHub_repository.

Post-Repository Cloning Instructions

1. **P4-Guide Setup:** After cloning the repository, check for a sub-folder named "p4-guide" within the directories "HTTP_TEST/p4runtime_based_implementation" and "SQL_TEST/p4runtime_based_implementation". If this sub-folder is absent, you should clone it from this link. Once downloaded, replace the "flowcache" folder within the newly downloaded "p4-guide" with the one from "p4runtime_based_implementation".

2. **POX Controller Configuration:** Upon successful installation of the POX controller on your OS, you will have to add all the files contained in the folder "POX_Applications" in the GitHub repository to the pox project you just installed. Add the files to the corresponding sub-folders.

3. **Final Integration Steps:** Once the above configurations are complete, follow the steps analogous to those provided for the OVF image to ensure a consistent setup across different environments. This includes running the scripts and others as detailed in the first section of the appendices.