B.Comp. Dissertation

# Tools for Security Scenario Setup

By

Sun Mingyang

Department of Computer Science

School of Computing

National University of Singapore

2018/2019

B.Comp. Dissertation

# Tools for Security Scenario Setup

By

Sun Mingyang

Department of Computer Science

School of Computing

National University of Singapore

2018/2019

B.Comp. Dissertation

# Tools for Security Scenario Setup

By

Sun Mingyang

Department of Computer Science

School of Computing

National University of Singapore

2018/2019

Project ID: H0041300

Project Supervisor: Assoc Prof Chang Ee-Chien

Deliverables:

- Report: 1 Volume
- Manual: 2 Volumes
- Program: 1 Git Repository

# Abstract

Normally, during Cyber Defence Exercise (CDX), it would be relatively easy to perform an attack or deploy an exploit once the vulnerability is known, however, setting up an appropriate security scenario or target system will be rather difficult. In a particular example, during a Cyber Defence Exercise targeting a virtual enterprise-scale network, the attacking traffic performed by the Red Team (who will conduct the attacks) may be too obvious to be spotted or identified by the Blue Team (who will defend the network) when the attacks are performed in the an “empty” virtual network where there is no background noise or “normal” web traffic.

The goal of this project is to build tools that could help to simulate the real-world scenario and generate normal and benign network traffics in the virtual environment during the CDX similar to what has been described above.

This project will consist of two components, while the first one is to build an orchestration tool for generating a sufficiently large amount of network traffic in a distributed system scenario, and the second one is to simulate human behaviors in web browsing to make the web traffic more realistic.

The evaluation and demonstration of this project will be conducted in the virtual network provided by the National Cyber-Security R&D Lab (NCL) and the final product may have international impacts on the CDX held by KYPO Cyber Range in the Czech Republic.

**Subject Descriptor:**

C.2.4: Distributed Systems
D.2.2: Design Tools and Techniques
D.2.3: Coding Tools and Techniques
D.2.13: Reusable Software
G.3: Probability and Statistics

**Keywords:**

Cyber Defence Exercise Setup, Orchestration Framework, IT Network Traffic Generation, Network Security, Distributed System, Human Behavior Simulation

**Implementation Software:**

Ubuntu Linux 18.04 Bionic Beaver, Python 3.6, Flask 1.0, Flask-Marshmallow 0.10, Flask-SQLAlchemy 2.3, Marshmallow-SQLAlchemy 0.16, Selenium 3.141, SQLAlchemy 1.3, osBrain 0.6, APScheduler 3.5, ntlk 3.4, React 16.6+, Ant Design 3.10+

# Acknowledgments

I wish to express my sincere thanks to Chang Ee-Chien, Associate Professor from the School of Computing, National University of Singapore, for providing me with all the necessary information and facilities for this project.

I place on record, my sincere thank you to Dr. Jan Vykopal from National Cyber-Security R&D Lab (NCL), for your help. I am extremely thankful and indebted to him for sharing expertise, and sincere and valuable guidance and encouragement extended to me.

I am also grateful to Ivo Nutar from the Institute of Computer Science, Masaryk University, Brno, the Czech Republic, for sharing with me lots of relevant work and past experience while I am developing the project.

Finally, I will also take this opportunity to express gratitude to all of the members in NCL who everlent their hands to me along this project for their help and support.

# Contents

# Chapter 1

# Introduction

## 1.1 Background

Practical Cyber Defence Exercise (CDX) creates virtual network environment and helps participants experience and understand how attacks against the infrastructures in their organizations are performed. The Blue Team (i.e. defenders) in CDX are normally supposed to come up with countermeasures and deploy some defending actions against a large variety of attacks that will be performed by the Red Team (i.e. attackers), as well as to monitor the context or the environment, a virtual network in most of the case, for any suspicious traffics or actions, or to recover from some successful exploits.

Since the CDX should simulate the real operational network environment as much as possible, instead of attacking traffics, there should be plenty of benign and normal web traffic existing during the CDX as well. In such case, the quality and accuracy of benign traffic has a significant impact on the learning experience and the effectiveness of the whole security exercise because usually we do not want the attacks to be the only traffics in the virtual network. Therefore, ensuring that the quality and volume of background traffic is able to camouflage the attacking traffics from any automated intrusion detection systems will be a very essential goal to set up the CDX virtual environment, or otherwise, it will be too trivial for the Blue Team to detect intrusive traffics.

To achieve the goal above, a large amount of background traffic that simulates actions performed by normal users or actual humans should be guaranteed. Many current practice of simulation of normal traffics in CDX may engage human beings or hard-coded traffic generators, for example, to use dedicated scripts or softwares on every individual machine, which are neither scalable nor efficient, especially when there is a dramatic system requirement change. Besides, as the scales of many CDX are enlarging quickly, more operators or clients will be needed to produce enough realistic network traffic. As a result, automating the process of generating sufficient amount of high-quality benign traffic according to a reasonable amount of configurations is becoming necessary. And this is also the main goal of this project.

To fulfill the requirements illustrated above, we are going to propose a brand-new framework to simulate and generate normal traffic during CDX, based on some state-of-the art works on automotive traffic generating and some studies on real human behavior. This framework will involve multiple client nodes and a controller node. Each of the clients will play a role of a fictitious user to simulate a normal network user who is performing regular actions such as web browsing and sending emails, while the controller node will play a role of an orchestrator, who is going to instruct all the clients based on some configurations, as well as to monitor system status on each client machine.

As for the types of traffic that will be generated by this framework, we will primarily focus on email and web traffic in this project. This is because these two kinds are of the most common activities that the internet users will have, while the portion of web traffic in the whole internet traffic mix has become dominant (Gebert et al., 2012). But we will also open the options and possibilities by design and implementing a highly extensible framework for the future for more types of traffic. Furthermore, since there is a much larger degree of randomness in web browsing behaviors of humans, we will also design and implement an algorithm or a mathematical model to predict or reproduce the realistic human web browsing traffic.

In this report, chapter 2 and chapter 3 will mainly focus on the scope of the design of orchestration tool, while chapter 4 and chapter 5 will mainly talk the methods that will be used to simulate human behaviors. Last but not least, we will address some of the limitations and discuss some of the future work.

## 1.2 Illustration of actions to be done by fictitious users in CDX

Based on the background we illustrated before, here, we are going to elaborate a more specific scenario where a “fictitious” user or a client is created and what it is supposed to do in order to generate normal traffic during a CDX. The fictitious users will reside in a virtual network created during CDX, and there will be numerous of them existing simultaneously to ensure the traffic volume. The actions specified below describes the actions needed to be performed by a certain client during CDX, and these actions will be executed on a routine base (i.e. repeat every day or every few hours) and will stop until the CDX ends.

| | |
|---|---|
| Client LAN IP: 10.0.26.4 | Browser: Chrome |

| **Time** | **Actions** | **Data** | **Destination application/service** | **CDX actions** |
|---|---|---|---|---|
| 8:00 to 9:00 | Browsing news sites | N/A | www.bbc.com<br>www.cnn.com | Red Team will disable DNS in the virtual network. |
| 9:00 to 10:00 | Send emails to a list of email address | email account credentials | admin1@ncl.org<br>admin2@ncl.org | Red Team performs MITM attack and hijack the email traffic |
| 10:00 to 11:00 | Check whether some internal web services is accessible (if not, send email to admin) | email account credentials | www.nclweb1.org<br>www.nclweb2.org | Some web services in Blue Team virtual network will be under DoS attack by Red Team. |
| 11:00 to 11:30 | Check external websites | N/A | www.google.com | The website will be defaced by the Red Team. |

## 1.3 The need for an automatic traffic generator and orchestrator to simulate human behaviours in CDX

Since most of the actions specified in the previous section are referenced from the actual actions to be done during some CDX held by NCL, and are actually achieved by actual humans. These tasks are nearly impossible to be directly replaced by automation scripts or softwares since they introduce lots of **randomness and decision making in terms of human behaviors**. However, this method will become really expensive when the scale of CDX becomes larger. Thus, we would like to propose a traffic simulator which could not only orchestrate a large scale distributed system, but also be able to include modules that could simulate human behaviors.

In the context of the table above, our proposed framework will help to schedule and execute all the entries appeared in the "Actions" columns, where the users of our framework (e.g. CDX coordinators) will only need to provide the information appeared in "Time", "Data" and "Destination" columns.

## 1.4 System design requirements

In this part, we would like to define all major requirements that our proposed traffic generating orchestration framework and the specific math model for human behavior simulation should satisfy. In the context of this project, the software or framework delivered will be the traffic generating orchestration tool, while the human web-browsing behavior simulation model will be one of the key modules in the framework, which would only help to generate web browsing traffics.

### 1.4.1 Requirements for Traffic Generating Orchestration Tool

1. **Modular architecture**: Modular software architecture should be achieved to enable extensibility and efficient development of individual components in parallel.

2. **Distributed system with controller**: There should be a controller acting as an orchestrator to instruct a number of clients that act as fictitious users distributed at various hosts in virtual

environment, and the controller and the clients should be able to communicate seamlessly.

3. **Real clients and tools are used**: Real clients and tools are used for generating realistic traffic. In other words, forged traffic or crafted packets should not be injected to network, but interactive sessions between clients and web services are established.

4. **Extensibility of protocols is supported**: Framework can be extended to generate traffic of various network and application protocols. For this project, we will only involve protocols that are related to web browsing (e.g. HTTP/HTTPS) and email (e.g. POP3/IMAP) first. But we will also ensure the high extensibility for the framework.

5. **Client-side cross-platform supported**: Client side framework should be deployable at least at Linux and Windows operating systems.

6. **High-level script of actions supported**: Traffic will be generated according to a human–readable high-level script of actions (which will be explained in more details in a later section) provided to the controller node by a human operator beforehand. The operator is able to specify fictitious users with their routine or scheduled actions, but not network packets nor flows. For example, valid actions may refer to "browsing the news website", "sending an email to X", or "check web service Y, if not accessible, send email to admin, and if accessible, go to the service and upload file Z".

7. **Scheduled/repeatable/conditional actions**: Actions defined in the high-level script should be able to be executed either according to a scheduled timeline (repeatable/non-repeatable) or certain conditions specified by the operator.

8. **On-demand actions**: Traffic can be also generated on-demand during the execution of the scheduled script. Framework provides a simple way for the operator how to instruct all clients from the controller.

9. **Graphical user interface**: The framework should provide a graphical user interface for the controller, enabling the operator to instruct and monitor the clients status and traffic status in real time with minimum effort.

10. **Dynamic plan for traffic generation**: The framework should give some flexibility for the operators to dynamically change the traffic generation plans in the middle of generation stage.

### 1.4.2 Requirements for Human Web-browsing Behavior Simulation Model

1. **Continuous generation of traffic**: The model designed should be able to continuously generate web traffic without any interruptions unless specified or instructed.

2. **Simulation of human behavior**: The model should be able to simulate how a real human being browses web pages to a certain degree. For example, if we want one fictitious user to browse inside a news website, it will be more likely for him to navigate through many news pages, but rather less likely to end up in clicking an advertisement link after a few minutes.

# Chapter 2

# Related Work: Traffic Generating Orchestration Tool

In this chapter, we would like to review a few relevant projects about traffic generation tools and framework as well as some of their key features, which we could potentially benefit from.

## 2.1 BrowSim

BrowSim (Hirner, 2018) is a web traffic simulator developed in Python. It makes use of Selenium framework for web traffic generation and Vagrant to manage all virtual machines (i.e. clients) involved. It supports both local web traffic generation where the traffics are merely produced by the machine where BrowSim is installed, and distributed web traffic generation where there is a hub (as a controller) to monitor and control all other nodes (as clients).

### 2.1.1 Usage

To use this tool, the user will utilize the provided API (which are just defined by python functions, and are also known as “browsing functions” in BrowSim's definition) to write a python script for each of the client nodes to explicitly specify their behaviors.

Essentially, BrowSim provides four browsing functions/APIs for web traffic generation:

- **keyword searching**: The user will apply this function to let the node search for certain keywords using a common searching engine as specified. The searching engine can be specified by passing a string parameter when this function is called.

- **news site crawling**: The user will apply this function to let the node crawl through a certain news website as specified in one of its parameters. The user will also need to specify how the crawling will happen – by clicking the "visually largest" hyperlink, or one of the first ten "visually largest" hyperlink, or just a random hyperlink inside the current webpage every time the Selenium web driver decides to proceed to a new webpage.

- **website logging in**: This function will try to login to a webpage, such as a social media website, according to user-specified username and password (provided as function parameters).

- **execute a file**: This function will execute a given file (where the file path is provided by a function parameter) using the Selenium webdriver.

### 2.1.2 Discussion

BrowSim is good enough to automatically generate a large amount of web browsing traffic, provided with some level of user customization. It can also be easily extended by implementing more powerful browsing functions with more robust browsing logic or predictions. And the Selenium framework it used is a really powerful tool for developers to conduct network reliability test or traffic simulation related to web browsing.

However, as all behaviors and logic flows happening on each node during generation are specified by the user crafted python script, the customization for different nodes becomes very expensive, and an operator who is lack of python programming skills may not be able to adept it very quickly. Besides, the communication between each client and the controller will happen every time when there is a new traffic to be generated, which injects lots of unnecessary traffics to the network. Another major problem is that BrowSim is not scheduler-based, which means creating iterative actions or scheduled actions will become expensive or even impossible during CDX.

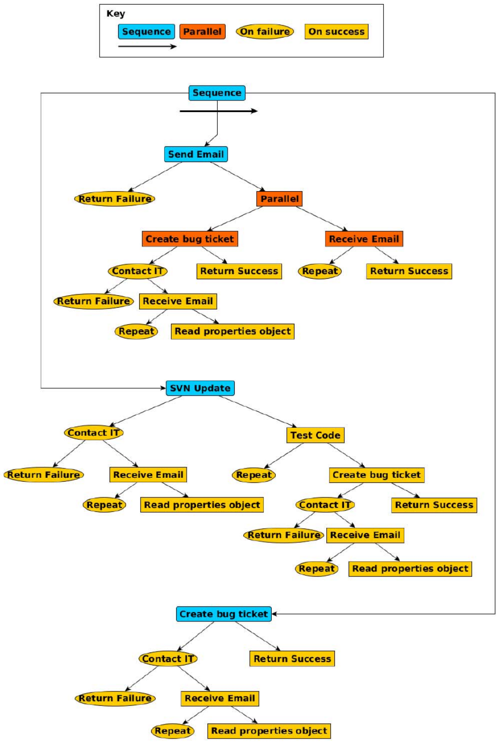
## 2.2 Yoshka

Yoshka (Mammadov et al., 2017) is a platform-independent framework that aims to generate high–fidelity network traffic and create lifelike virtual environment during CDX. The traffic is scheduled and generated based on a behavior tree compiled from a user specified high-level script written in YAML, a human-readable serialization language widely used in configuration files.

Behavior tree (Shoulson et al., 2011) (Nicolau et al., 2017) is basically a directed acyclic graph commonly used in game development industry to determine the behaviors of non-playable characters (NPC). To further simplify the behavior logic, Yoshka makes use of binary trees which will restrict the outcome of each vertex in behavior tree to only success and failure. Below are examples of the high-level script of actions and the behavior tree used in Yoshka.

```
- name: Visit web service
  task: web.GetRequestTask
  args:
      name: Visit web
      url: <web_server>
  success: returnSuccess
  failure: send mail to IT
  root: true
  duration: 10

- name: send mail to IT
  task: mail.SendMailTask
  args:
      name: send mail
      mailServer:<mail_server>
      from: <user>@<mail>
      subject: Can’t access web service
      message: Web service is giving bad
↪   requests
      recipients: <user>@<mail_server>

- name: returnSuccess
  task: generics.ReturnSuccessTask
  args:
      name: success
```

(a) High-level Script



(b) Binary Behavior Tree

Figure 2.1: Examples and Concept Illustration in Yoshka

### 2.2.1 Discussion

Yoshka provides support for a wide spectrum of network traffic, such as mail traffic, FTP traffic and web browsing traffic, along with a graphic interface on controller node to monitor the real-time status of network traffics and other client machines. Moreover, Yoshka is also of a very high extensibility for new type of traffic or protocols to be added on, which provides a lot more adaptability to changing scenarios and conditions in CDX.

Nevertheless, because Yoshka is a closed project, people outside their institute will not be able to access the source codes. But we can always borrow some valuable concepts from this project through what they have already published, such as the idea of how to make use of binary behavior tree to construct the human readable high-level script.

## 2.3 Cryton

Cryton is an extensible framework written in Python developed by Nutar (2018) from the Institute of Computer Science, Masaryk University, Brno, Czech Republic, and was originally designed to automate the attacking process (or generating attacking traffic) during CDX. This framework was developed based on client-server architecture, where a centralized server/controller will orchestrate all client behaviors. It implemented a similar idea of binary behavior tree in Yoshka, and it also makes use of high-level script of actions (written either in JSON or YMAL) for user specification.

Cryton introduced a concept called “modules”, each of which represents a specific “attacking” action and will only be deployed on the client machines. There are plenty of pre-prepared modules could be used in Cryton, such as “mod_scanner” module which will make use of the nmap tool to scan a certain IP address on certain ports, and “mod_msf” module which will make use of certain metasploit resource to conduct corresponding exploits or attacks. An example of the high-level script of actions is shown in the figure below:

```
- name: scan-victim-ssh
  is_init: true
  action:
    # Use mod_scanner
    name: mod_scanner
    args:
      ports:
      - 22
      - 80
  # In case of success continue with Step: access-to-ssh
  succ_ok:
  - access-to-ssh
- name: access-to-ssh
  # In case of success create named session ssh-victim-sess
  create_named_session: ssh-victim-sess
  action:
    name: mod_msf
    args:
      exploit: auxiliary/scanner/ssh/ssh_login
      USERPASS_FILE: /usr/share/metasploit-framework/data/wordlists/root_userpass.txt
      RHOSTS: 192.168.56.101
  # In case of success continue with Step: get-etc-passwd
  succ_ok:
  - get-etc-passwd
  # In case of failure continue with Step: access-to-ssh-alt
  succ_fail:
  - access-to-ssh-alt
```

Figure 2.2: Example of high-level script used in Cryton

### 2.3.1 Design Architecture

Thanks for the support from the author, Nutar (2018), we could get the chance to access the original source code and examine and learn from the software architecture of Cryton. The figure below is a high-level review of how Cryton was implementing the server-client software architecture, and here in Cryton implementation, word "master" is used to define the controller/server node, while word "slave" is used to for client node.

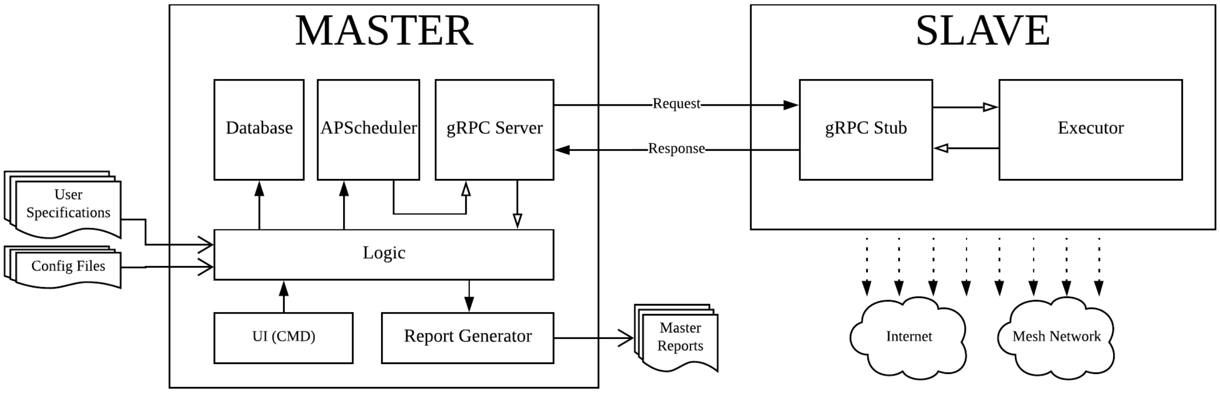


Figure 2.3: Software architecture of Cryton

Basically, the user of Cryton will interact with Cryton through a command-line user interface on the master node. He or she can pass in a high-level script of actions that specifies the slave behaviors slaves. Then it will become master's responsibility to parse the input file, generate the behavior tree logic and then schedule the tasks or actions accordingly. When certain tasks are to be executed, the master will notify and give the instruction to the slaves and the slaves will execute the modules and return the execution status (success or failure) back to the master to generate an execution report.

### 2.3.2 Discussion

While there are quite a few design concepts we could learn from Cryton, there are also several major limitations about Cryton explaining why we may not be able to extend it to fulfill our project requirements. Here, we are going to discuss three major ones.

First of all, in attacking scenario, consecutive actions are connected or linked via session (e.g. ssh session or TCP session) and the decision of action flow may merely depend on a binary status (i.e. either success or failure), for example, once an SSH session is created successfully, Cryton may remember the name of the session and return a success code for the next action to start execution based on the existing session. However, for the actions or tasks to be used for normal web traffic generation, the behaviors may highly depend on the real output (e.g. an HTML page) from its predeceasing action, which cannot be simply achieved by the binary behavior tree implemented in Cryton.

Secondly, in Cryton, every action can only be scheduled for once. The scheduler implemented on master node does not support any iterative tasks. Though repeating tasks can still be created by passing in multiple identical high-level script of actions at different time stamp, this process may become very trivial and this will become much less intuitive for the users.

Thirdly, every action is scheduled on master node, presenting a lot of overhead on the controller machine as well as some delay of execution for some tasks. Though multiple threads will be created for each individual chain of actions, the overhead and the delay will still be inevitable as the number of slave nodes increases. For example, if there are one hundred tasks scheduled on one hundred slaves at a certain time point, the master node must instruct these one hundred slaves simultaneously to finish all task execution.

## 2.4 DumpsterFire Toolset

The DumpsterFire toolset (Unknown, 2017) is an open source project for creating scheduled, repeatable and distributed security events during CDX. By saying security event, it may refer to any actions done by either Red Team or Blue Team during CDX. Each atomic action is called DumpsterFire, and the user can schedule the DumpsterFire as they need.

DumpsterFire toolset is modular, cmd-menu-driven, cross-platform and easily extensible. However, the major issue with DumpsterFire toolset is that this tool may not be able to be deployed on a distributed scenario. It will be really helpful for a specific member in either Red Team or Blue Team to use this tool to save their time on some trivial or standardized actions. However, to meet the requirements of the CDX we specified before, each client machine needs to be installed with a DumpsterFire toolset and configured separated. Besides, to schedule a DumpsterFire, there is no easy way such as passing in an input configuration file, but the user has to use the command line interface to specify and configure each task explicitly.

## 2.5 AutoTTP

AutoTTP (Automated Tactics Techniques & Procedures) (Unknown, 2018) is another open source project that aims to simplify only the attacking procedures during CDX. Again, it is not capable to generate a sufficient amount of traffic since this tool does not support distributed deployment and it is hard to be extended to generate normal traffic since most of its atomic actions are dependent on some attacking frameworks.

## 2.6 Summary

As we have explored a few existing tools or frameworks for traffic generation (either for normal traffic or attacking traffic), no one of them is very suitable to be applied or extended to our scenario, though some of the design decisions and concepts can be learnt and studied. Here is a table of summary on how these related projects fit our project requirements specified in section 1.4.1.

| | BrowSim | Yoshka | Cryton | DF toolset | AutoTTP |
|---|---|---|---|---|---|
| Modular architecture | NO | YES | YES | NO | NO |
| Distributed system | YES | YES | YES | NO | NO |
| Real clients and tools | YES | YES | YES | NO | YES |
| Extensibility of protocols | NO | YES | YES | YES | YES |
| Client-side cross-platform | YES | YES | YES | - | - |
| High-level script of actions | NO | YES | YES | NO | NO |
| Scheduled actions | NO | YES | YES | YES | YES |
| Repeatable actions | NO | YES | NO | YES | YES |
| Conditional actions | NO | YES | YES | NO | NO |
| On-demand actions | NO | YES | YES | YES | YES |
| Graphical user interface | NO | YES | NO | NO | NO |
| Dynamic scheduling plan | NO | - | NO | NO | NO |
| Access to Source | YES | NO | YES | YES | YES |

# Chapter 3

# System Design: Traffic Generating Orchestration Tool - Autraff

In this section, we are going to propose our own framework, Autraff, for traffic generation to satisfy the requirements specified in section 1.4.1. Firstly we will introduce the software architecture followed by detailed explanation of every single component. Then we will introduce the high-level script used by users in our framework followed by some implementation details of the framework. Last but not least, we will be illustrating a simple use case by using a flow chart.

## 3.1 Terminologies

Before we start, we would like to first clarify some terminologies used in our system.

| **Term** | **Meaning** |
|---|---|
| Client | Referring to the actual node in the network that is used to generate the network traffics. |
| Controller | Referring to the node where the orchestrator (which is used to manage all client nodes) of the orchestration framework is installed. |
| UI | In the content of Autraff, most of the time UI refers to a graphical user interface. |

| **Term** | **Meaning** |
|---|---|
| Module | It is used to define a type of atomic actions that will be stored on the client side. For example, module “mod_visit_any_page” defines a type of actions that will visit one specific web page only. |
| Argument | Each module will consist of several arguments which will decide the exact behaviors of a module. For example, module “mod_visit_any_page” may have an argument called ‘url’ to tell the module to visit which web page and another argument called ‘time’ to tell how long this module should stay on this page before leaving/closing. |
| Job | It is a definition to represent the actual actions that will be performed by the client. Each job will make use of exactly one module, and it will also define when the action will be executed and consist of all arguments needed to execute a module. Two different jobs can use the same module but with different arguments. |
| Start Time | Some jobs may be defined with a start time which denotes the time when this job is executed for the first time. |
| Interval | Some jobs may be defined with an interval attribute, which denotes the time interval between the two consecutive execution of the same job. |
| Schedule | When used as a verb, this word means to schedule a job on a specific client according to how this job is defined. Note that, not all jobs can be scheduled since some jobs may be triggered by some events. |

## 3.2 Software Architecture and Overview

In this section we are going to present the high-level architecture design of Autraff. Basically, it follows the server-client architecture inspired by Cryton (Nutar, 2018), but we have designed it to make it more suitable to fit our assumptions and scenarios described in section 1.2 and 1.3.

In the architecture diagram shown in figure below, controller plays the role of an orchestrator to manage and monitor all client machines that are assigned to generate the traffics. One controller can connect to multiple clients at the same time.

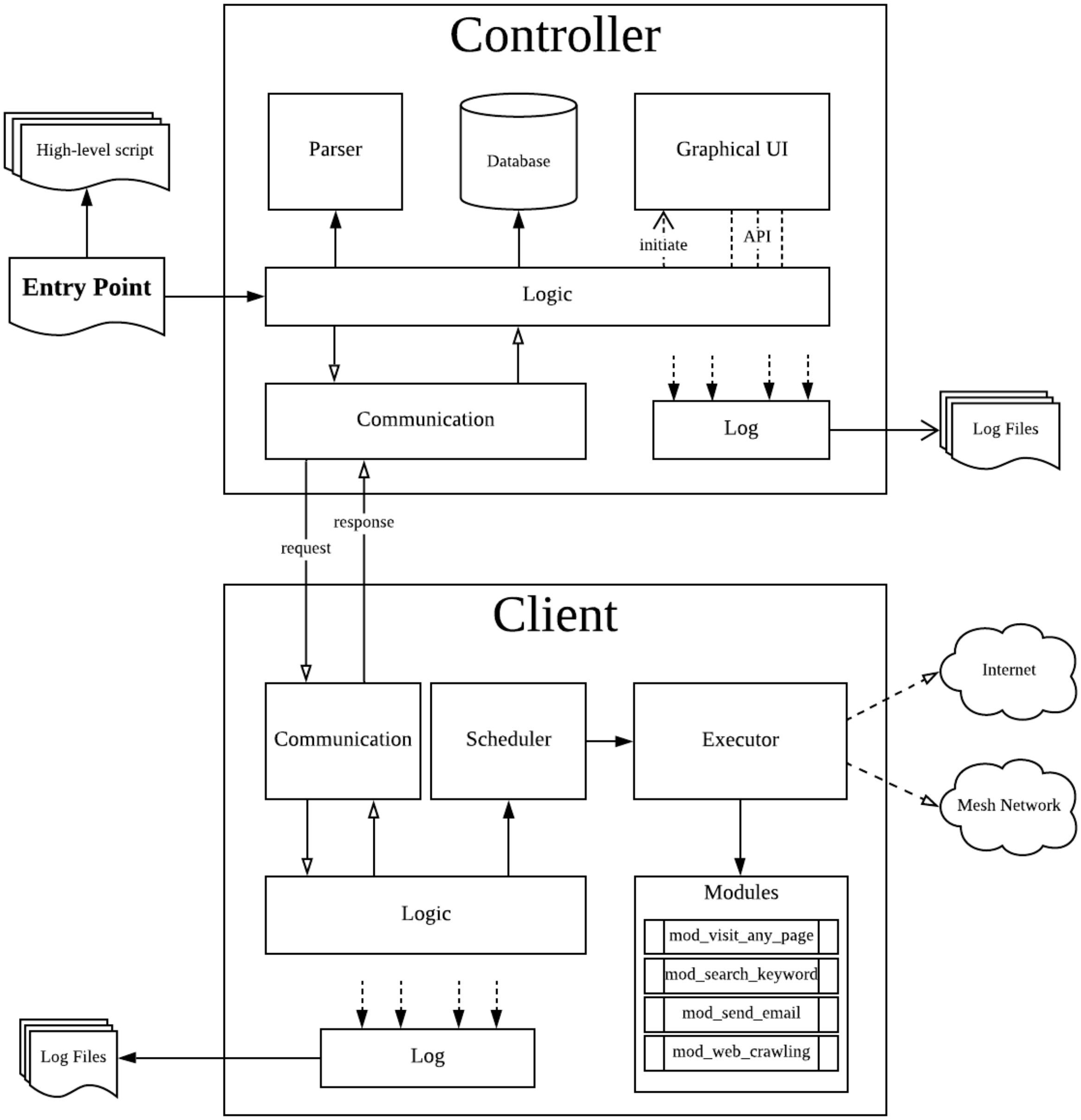


Figure 3.1: Software Architecture of Autraff

### 3.2.1 Controller

#### Entry point

Entry point is where the user uses to start up the controller-side framework. It takes all user configuration files as well as other user input files such as the the high-level script that we are going to illustrate later.

### Logic

The entry point will start the logic component on controller side, and logic will start other components including parser, database, graphical ui and communication sector. After start, the logic component will also provide APIs through the graphical UI component for the user to control the controller or give instructions to the controllers.

### Parser

The parser component is a component where all user configuration and our high-level script will be parsed into program parameters or objects, some of which will be then saved into database or displayed through graphical UI.

### Graphical UI

The graphical UI will be the main interface for users to interact with the logic controller or to instruct client nodes (indirectly through controller) after the user starts the controller program through the entry point. For example, after the controller has started, the user may want to schedule a job on a certain client, he or she may go to a specific UI page and schedule that job there.

### Communication Component

The communication component will be mainly responsible for all information exchanges between the controller and each of its clients. The communication pattern will mainly be “requests and response” format, and the controller is always pulling information and updates from all its clients, whereas there will be no way for the client to pushing information to the controller unless requested.

### Log Component

This component will be mainly in charge of the logging logics of all other components. All other components except for the graphical ui will be using it for logging.

### 3.2.2 Client

#### Logic

The logic component on the client side orchestrates all other components and manages the logic and data flow among other components. The logic component here is also a so-called entry point of the client-side program. On the client side, the user will start up the logic component first to establish the connection between this client to the controller.

#### Communication Component

The communication component on the client side will keep listening on the requests sent by the controller. It will pass all the request to logic component and the logic component will handle accordingly.

#### Scheduler

The scheduler will be instructed by the logic component and schedule the jobs accordingly on the client machine. Every time there is a job to be executed, the scheduler will call the executor, and the executor will execute the module that defined in this job.

#### Executor

The executor will execute the actual jobs according to the instructions from either the scheduler component or the executor itself. If a job is scheduling based (i.e. the job is defined with a start time or an interval), then the scheduler will instruct the executor to execute this job based on its schedule. If a job is event triggered (i.e. the job does not have a start time nor an interval, but it will be executed based on the successful or failing execution of another job), then the execution of this job will be triggered after the executor executes the triggering job.

#### Module

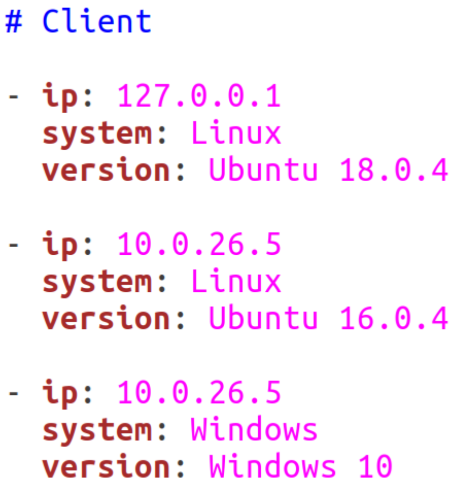
The module here refers to a type of “actions”. For example, module “mod_visit_any_page” and module “ mod_search_keyword”. Each standalone module may have nothing special, but is just a single script or program. Each module may take a few arguments, each of which will decide the actual behaviors of the module during execution.

## 3.3 High-Level Script

The high-level scripts will be used as a "user configuration" files in our framework, which will only be stored and used on the controller side. The users will use the scripts to specify and define where and how the traffic will be generated. Basically, we will need two types of high-level scripts (written in YAML), one of which will specify all the information of clients that will be involved in generating the web traffic, while the other one will be used to specify the chain of actions to be done by different clients.

### 3.3.1 Script of client information

In the script defining the client node, the syntax will be rather simple and each YAML "object" is referring to a certain client including the information about IP address (the ip address used to connect this machine from the server/controller), operating system and system version. An example of the script is shown in the figure below:



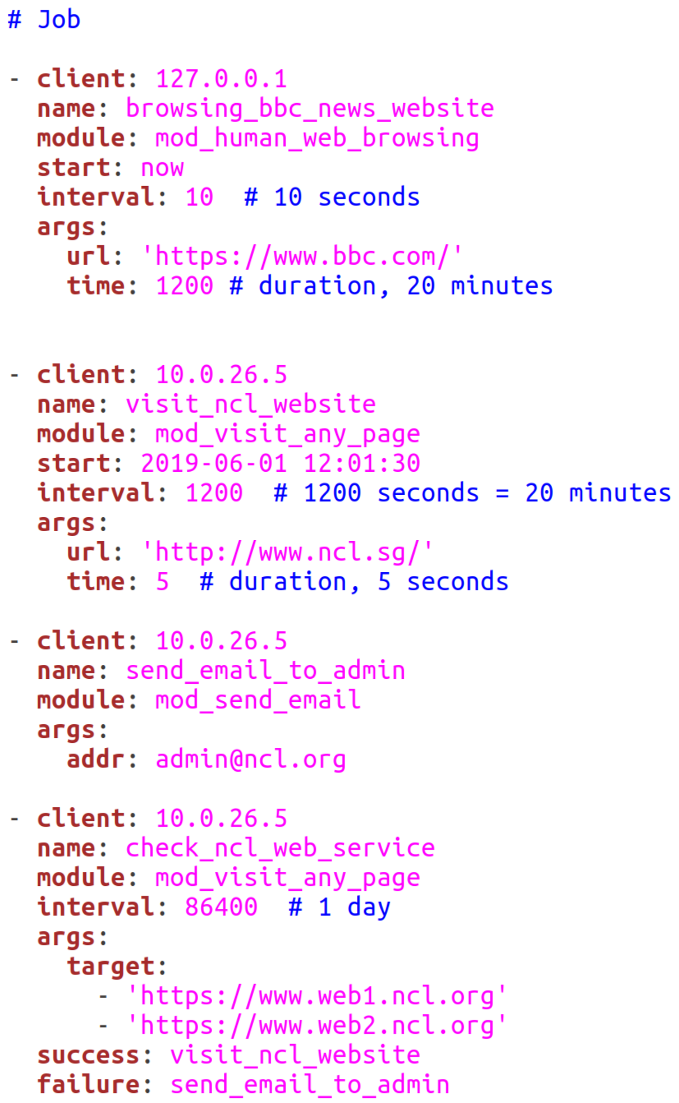
```
# Client

- ip: 127.0.0.1
  system: Linux
  version: Ubuntu 18.0.4

- ip: 10.0.26.5
  system: Linux
  version: Ubuntu 16.0.4

- ip: 10.0.26.5
  system: Windows
  version: Windows 10
```

Figure 3.2: Example of script of client information

In the script described above, the controller will use 'ip' field to connect to and differentiate each client, and for each client, they will have and only have one unique ‘ip’.

### 3.3.2 Script of job information

During the design process of the script of job information, we refer to the idea from Yoshka Mammadov et al. (2017) and Cryton (Nutar, 2018). We give users the capabilities to define binary behavior tree of actions/jobs in the script. A snapshot of a sample script is shown in the figure below.



```
# Job

- client: 127.0.0.1
  name: browsing_bbc_news_website
  module: mod_human_web_browsing
  start: now
  interval: 10  # 10 seconds
  args:
    url: 'https://www.bbc.com/'
    time: 1200 # duration, 20 minutes


- client: 10.0.26.5
  name: visit_ncl_website
  module: mod_visit_any_page
  start: 2019-06-01 12:01:30
  interval: 1200  # 1200 seconds = 20 minutes
  args:
    url: 'http://www.ncl.sg/'
    time: 5  # duration, 5 seconds

- client: 10.0.26.5
  name: send_email_to_admin
  module: mod_send_email
  args:
    addr: admin@ncl.org

- client: 10.0.26.5
  name: check_ncl_web_service
  module: mod_visit_any_page
  interval: 86400  # 1 day
  args:
    target:
      - 'https://www.web1.ncl.org'
      - 'https://www.web2.ncl.org'
  success: visit_ncl_website
  failure: send_email_to_admin
```

Figure 3.3: Example of script of job information

In the example above, each job will have compulsory attributes such as:

- **client** an ip address to specify on which client the job is to be executed (Note that the client ip must exist in the script of client information)
- **name** a unique alias of the job
- **module** the module used in this job

There are also optional attributes that will give a job more capabilities or responsibilities:

- **start** the start time of a job (If the start time is not specified, then it means that the job will be either triggered by other jobs or the user will schedule this job manually. If the start time is specified as now, then right after the controller starts, the controller will tell the client to schedule the job immediately.)
- **interval** the time interval between two consecutive execution of a job (If this is not specified, the job will be either triggered by other jobs or it will only be executed once.)
- **args** the arguments that will be passed into the module during execution of the job (This attribute can be empty only when the module used in this job requirs no arguments.)
- **success** the job that will be executed if this job is executed successfully
- **failure** the job that will be executed if the execution of this job fails (Both success and failure attribute will contribute to the binary behavior tree of actions.)

To illustrate more on the scenario described in the figure above: there are four jobs defined in the script – one of them will be scheduled on client 127.0.0.1 (which is localhost, the same as the controller machine), and three of them will be executed on client 10.0.26.5. To be more specific:

1. job **browsing_bbc_news_website**: This job will be executed on client 127.0.0.1 and it will make use of module “mod_human_web_browsing”. It will be scheduled right after the controller is up with an interval of 10 seconds.

2. job **visit_ncl_website**: This job will be scheduled according to its start time and interval duration accordingly. Besides, this job can also be triggered by the successful execution of job “check_ncl_web_service”.

3. job **send_email_to_admin**: This job is defined without a start time or an interval, thus it can only be triggered when the execution of job “check_ncl_web_service” fails.

4. job **check_ncl_web_service**: This job will only be scheduled when the user explicitly schedules it from the controller UI. Upon being scheduled, the job will be run once every day. For every execution, if it is successful, then job “visit_ncl_website” will be executed for once, otherwise, job “send_email_to_admin” will be executed for once.

Thus as a result, in the example above, jobs defined on client 10.0.26.5 will result in a simple binary behavior tree, which can be illustrated by the figure below:

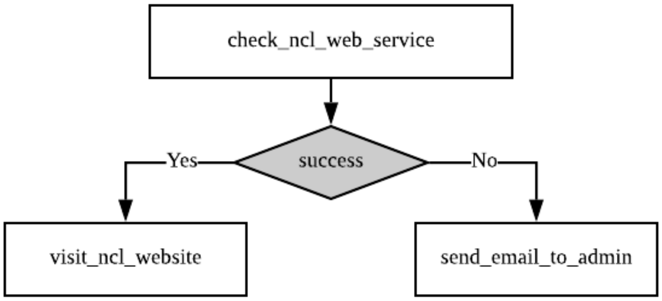


Figure 3.4: Binary behavior tree generated according to the template script of job  information

As more jobs are added in the high-level script, a more complex behavior tree will be generated, which may include iterations or recursions.

## 3.4 Implementation Details

In this section, we will talk about the actual implementation in more details. We will firstly explain the implementation details of the communication components on both controller and client side, followed by the components details on the client side, and lastly we will address the controller side.

### 3.4.1 Communication

As for communication component, we are using osBrain (Open Sistemas de Informacin Internet S.L., 2019), a general-purpose multi-agent system module, which could be used in a distributed system for message exchanging between the server and clients.

The figure shown below roughly describes how the communication channels are established between the client node and the controller node. More details will be explained.

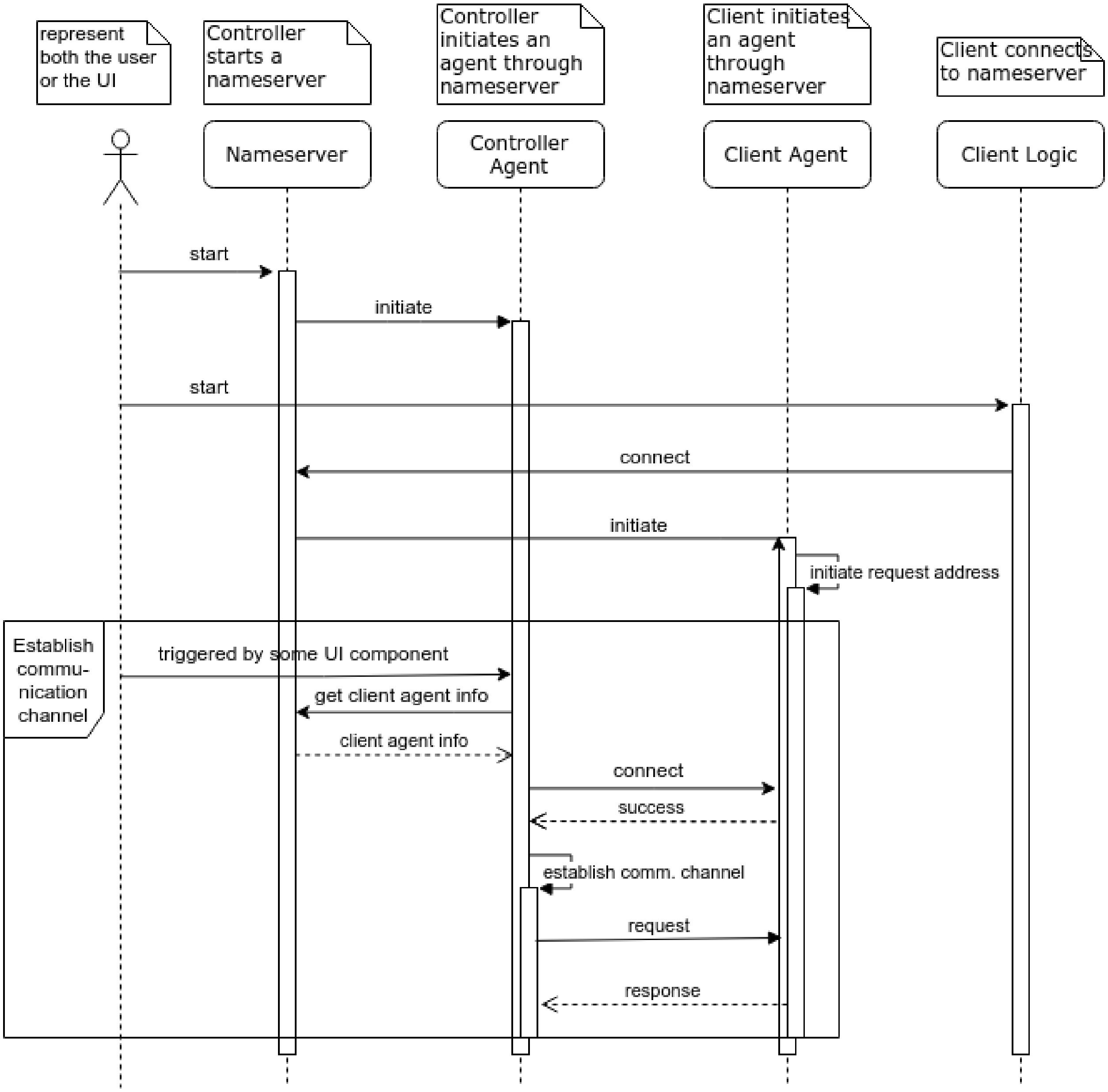

represent both the user or the UI
Controller starts a nameserver
Controller initiates an agent through nameserver
Client initiates an agent through nameserver
Client connects to nameserver
Nameserver
Controller Agent
Client Agent
Client Logic
start
initiate
start
connect
initiate
initiate request address
Establish commu- nication channel
triggered by some UI component
get client agent info
client agent info
connect
success
establish comm. channel
request
response

Figure 3.5: Binary behavior tree generated according to the template script of job information

Every time the user starts up the controller side framework, a nameserver will be running on the controller machine and an agent will also be initiated to represent the controller node. Before all the jobs are scheduled, the user needs to go to each client machine and start up the client side program (basically the logic component as described in the previous section). The client will firstly try to connect to the nameserver and then initiated another agent to represent itself.

After initiation, the client agent will initiate a special "address" (a special object in osBrain terminology) for receiving request and dispatching response. Then the user on the controller side will decide or trigger an event to make the controller agent connect to the client agent. The controller agent will firstly get the client agent information from the nameserver and then it will try to establish a TCP connection with the client agent. If it succeeds, then the controller agent will initiate a "connection" object (another special object in osBrain terminology) which will be used directly for future request and response communication.

### 3.4.2 Client

#### Logic

The logic component on client side is implemented as a wrapper python script of all other components. It is in charge of the establishment of the client agent and the communication "address", as well as to instruct the scheduler component to schedule certain jobs when such requests are received from the communication channel.

#### Module & Executor

Each module in Autraff is implemented as a standalone python script where there will be only one function called "execute" is defined. And the concept of executor defined in the previous section will only be an abstract concept, and there will be no concrete and explicit implementation of the executor. The skeleton of the function is defined as the following:

```
def execute(args, driver=None):
    # function body
    return driver
```

“arg” is a dictionary object that contains all compulsory and optional arguments needed for executing a certain module. “driver” is a Selenium web driver object which will be used to perform the real web browsing actions. If the execution of the module is triggered by a predecessor module, then the predecessor will pass the driver used by itself to its successor module, otherwise, a new driver will be initiated by the module itself.

For the declaration of a module's successors (i.e. the success and failure field specified in the high level script), they will be passed together inside the “args” dictionary. For the binary behavior tree descirbed in figure 3.4, the args parameter in module “mod_visit_any_page” used by job “check_ncl_web_service” will be constructed as:

```
args = {
    target: ['https://www.web1.ncl.org',
             'https://www.web2.ncl.org'],
    success: {
        module: 'mod_visit_any_page',
        args: {
            url: 'https://www.ncl.sg/',
            time: 5,
        },
    },
    failure: {
        module: 'mod_send_email',
        args: {
            addr: 'admin@ncl.org'
        }
    },
}
```

And as for the return value, a module will always return the driver object that it uses. If this module is called by the scheduler, then the return value will have no meaning, but it will make sense to a module's predecessors if this module is triggered or used by other modules.

Here we will list some most useful modules that have already been implemented:

- **mod_visit_any_page**: this module will take a url as a compulsory argument and visit and stay on this web page for a certain amount of time.
- **mod_search_key_word**: this module will take a keyword as a compulsory argument and search this keyword using a popular searching engine as specified in another argument. And then finally click on one of the searching results.
- **mod_human_web_browsing**: this module will take a url as a compulsory argument and start to crawl from this web page, which will give a simulation of how a real human is browsing the web. The underlying theory and algorithms used will be explained in the following chapter.
- **mod_send_email**: this module will take an email address as a compulsory argument and send an email to this address. The user can also specify the username and password to be used to send the email, as well as the email subject and body.

**Scheduler**

As for the scheduler implementation on the client side, we made use of the APScheduler (Advanced Python Scheduler) library (Grnholm, 2019), which could help to schedule to execute python code either periodically or at a fixed time.

Each client node will have exactly one background scheduler defined in logic component, and it will be initiated as long as the user starts the client side program. The figure below roughly illustrates how the scheduler works in our context.

To illustrate, every time there is a request regarding scheduling or stopping a job sent from the controller to the client, the client logic will resolve the request body and then give the instructions

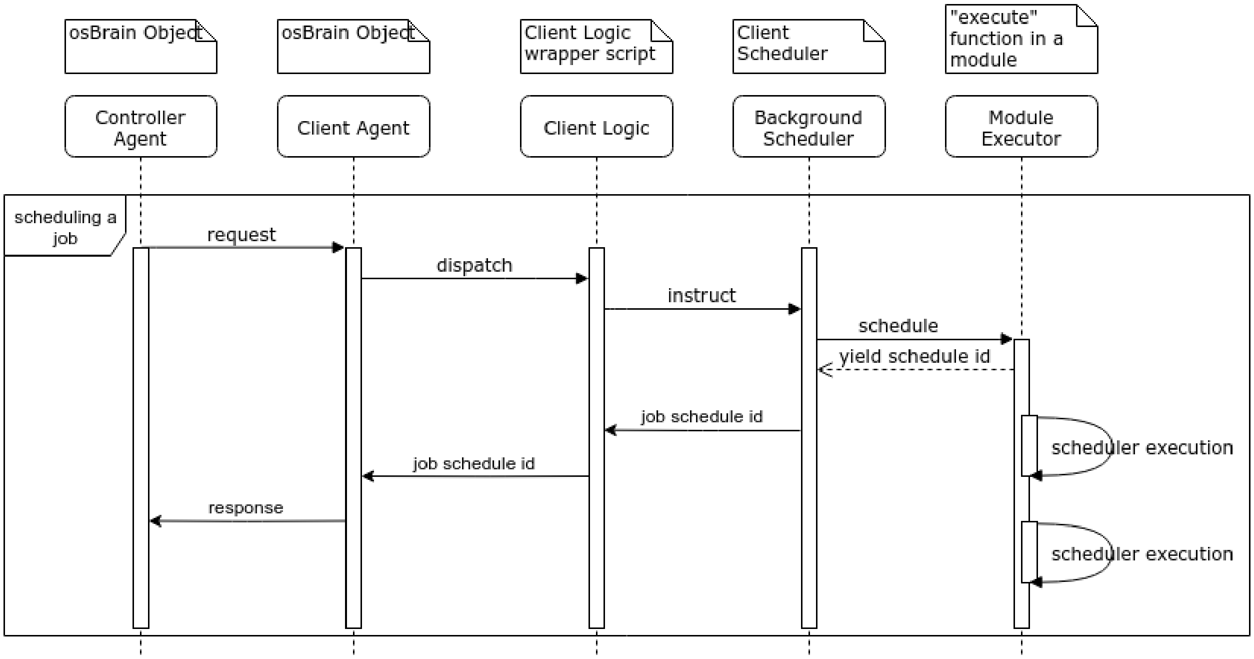


Figure 3.6: Sequence diagram of how scheduler works.

to background scheduler. And if nothing goes wrong, the logic component will response the controller with a success message.

In such a manner, both the controller and the client logic component will not be aware of what is happening to the actual execution of a job if not requested. But the log component will always keep track of all the subtle changes happening in this system.

### Log

The log component on the client side mainly consists of two part. One is a general logger object, which can be accessed anywhere in any components. By using this logger, each component can specify where to write their logging messages or just to a global general log file. Another one comes as a python decorator, which will be used when defining or developing each of the modules. After being wrapped by this logger decorator, the logging system will output with a standard format to record the modules and jobs execution. And now, a sample module will be defined as:

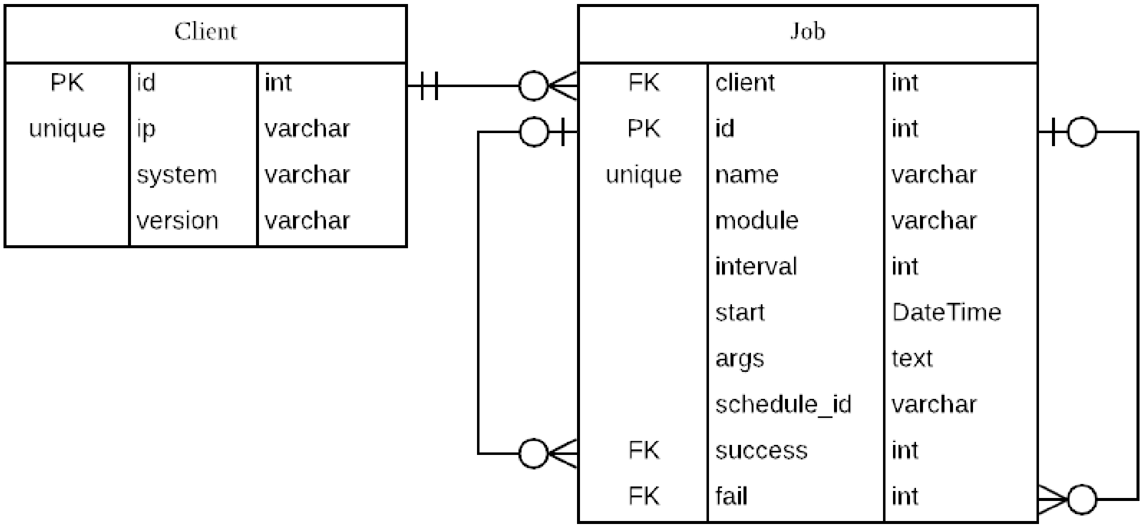
```
@log_mod_execution
def execute(args, driver=None):
    # function body
    return driver
```

### 3.4.3 Controller

**Parser & Database**

The implementation of parser is done by yaml reader in python standard library. It will take the input file (i.e. the high level script) and then parse them into python object which the logic component could understand.

After parsing, the logic component will store all the parsed information into the relational database which is implemented using SQLite database. The database mainly consists of two tables, one of which stores the information about clients, and the other one stores the information about the jobs. The figure below shows the schema of the relational database.



| Client | | |
|---|---|---|
| PK | id | int |
| unique | ip | varchar |
| | system | varchar |
| | version | varchar |

| Job | | |
|---|---|---|
| FK | client | int |
| PK | id | int |
| unique | name | varchar |
| | module | varchar |
| | interval | int |
| | start | DateTime |
| | args | text |
| | schedule_id | varchar |
| FK | success | int |
| FK | fail | int |

Figure 3.7: Controller database schema diagram.

### Logic & Backend

The logic component on the controller side does not only play the role of a wrapper for all other components, but also serves a backend engine for the UI component. The backend logic was developed by Flask web framework, and after the user starts up the controller program, there will be a web server running locally providing restful APIs for the outsides (i.e. the UI component) to interact with the database or to communicate with other clients through the communication component.

Here is a list of the major APIs provided by the backend engine:

1. **GET /nameserver/status**: This API will return whether the osBrain nameserver is running or not.

2. **GET /nameserver/debug**: This API will check the status of nameserver, if the nameserver is not running, the backend will try to restart the nameserver.

3. **GET /controller/connection**: This API will return whether the controller osBrain agent is running or not.

4. **GET /controller/debug**: This API will check the status of controller agent, and if it is not running, the backend will try to start it.

5. **POST /connect**: Calling this API will make the backend attempt to connect to a client whose ip address will be specified in the request body.

6. **POST /connect/all**: Calling this API will make the backend attempt to connect all the clients that are stored in the database.

7. **GET /connect/<client-ip>**: This API will check whether the controller agent is connected with the specified client agent.

8. **POST /scheduling/schedule**: By calling this API, the backend will take the job id inside the request body and find the corresponding job information (i.e. client, name, args, etc.) in the database. Then the backend will dispatch the scheduling job request to the responsible client. After the job is scheduled by the client, a scheduling id (which is nothing but a hash

value) will be returned to the controller backend, and the controller backend will send back this id as the response of the API call.

9. **POST /scheduling/stop**: By calling this API, the backend will take the scheduling id as well as the client ip inside the request body. Then, it will send a stopping job request to the corresponding client, the client will then kill the job in the scheduler using the sheduling id. If the job is successfully killed, the client will respond with a success message and the controller backend will respond with a success code as well.

10. **GET /log/<client-ip>**: Retrieve the client logs.

11. **GET /log/<client-ip>/<job-scheduling-id>**: Retrieve the logs for a specific job from a specific client.

There are also some other APIs used to manipulate the database. We will list out some of them below, but we will not explain in details since most of them have self-explaining API endpoints.

- **GET /client**: Get all clients' information.

- **GET /client/<client-ip>**: Get a specific client's information.

- **POST /client**: Post with a new client's information. (Note that this function has not been developed in the frontend)

- **GET /job**: Get all jobs' information.

- **GET /job/<job-id>**: Get a specific job's information.

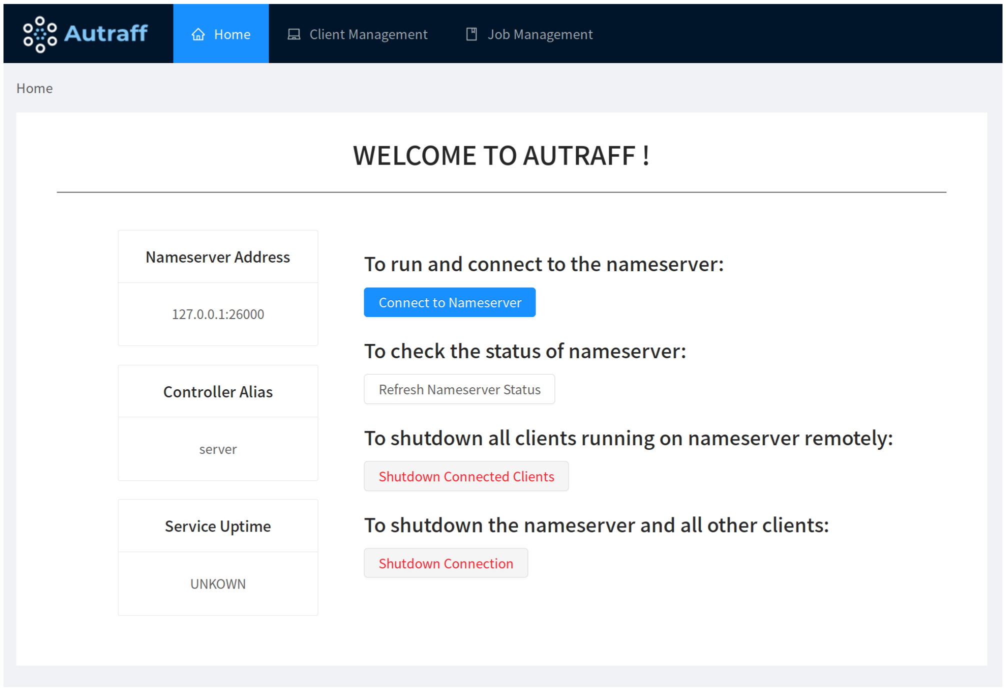
- **POST /job**: Post with a new job's information. (Note that this function has not been developed in the frontend)

By make the controller backend logic component a separate web server, we can now build user interfaces of any kind (e.g. graphical or command-line driven) with minimum efforts.

### UI & Frontend

The graphical user interface of Autraff is developed using React framework. After the user starts up the controller side program and right before the back end web service stated, the logic component will start the React engine first, and the user can access the UI using a designated url, such as “localhost:3000”, from the web browser on the controller machine. And before user would like to make any changes through the UI, for example, scheduling a task on certain client, he or she has to go to each client and start the client side program there. Once done, the user can start to orchestrate all the traffic generation process through this web UI. A detailed explanation of how Autraff works from the user perspective will be explained in next section.

As for the Autraff UI itself, it provides huge convenience for the user to monitor and orchestrate the traffic generation process. It will display all client information and all job information in a clear and neat manner. It has dashboards for both client and job management, and it makes the job scheduling and log retrieving process rather simple with just a single click. A few sample screenshots of Autraff UI is attached below.



WELCOME TO AUTRAFF !

Nameserver Address: 127.0.0.1:26000

Controller Alias: server

Service Uptime: UNKOWN

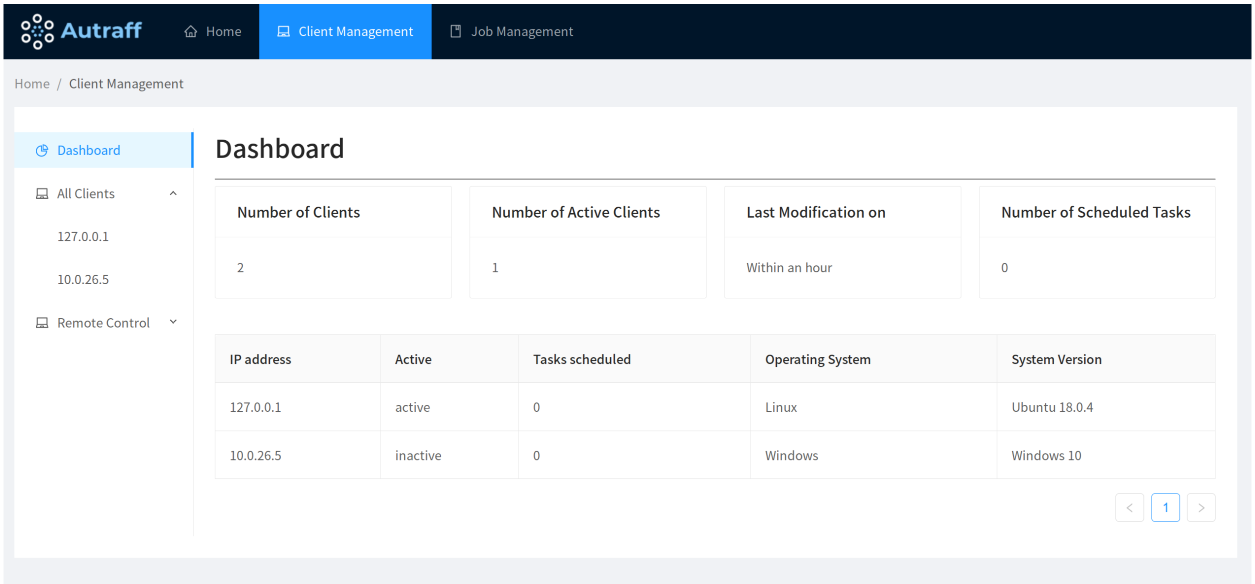
To run and connect to the nameserver:

To check the status of nameserver:

To shutdown all clients running on nameserver remotely:

To shutdown the nameserver and all other clients:

Figure 3.8: Autraff UI - home page

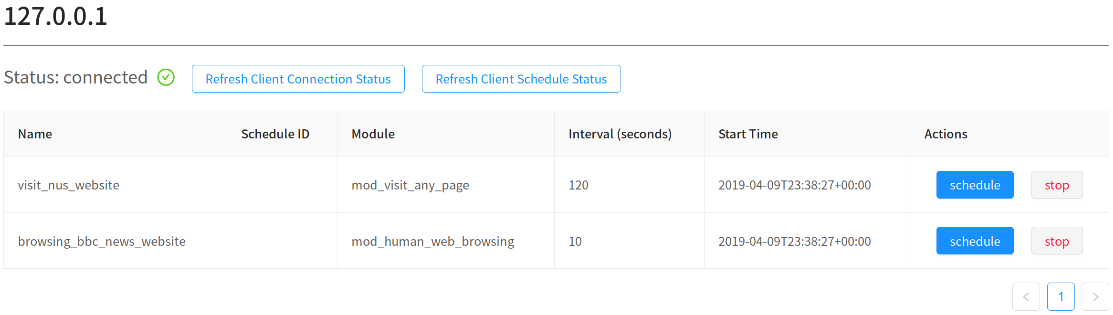


## Dashboard

| Number of Clients | Number of Active Clients | Last Modification on | Number of Scheduled Tasks |
|---|---|---|---|
| 2 | 1 | Within an hour | 0 |

| IP address | Active | Tasks scheduled | Operating System | System Version |
|---|---|---|---|---|
| 127.0.0.1 | active | 0 | Linux | Ubuntu 18.0.4 |
| 10.0.26.5 | inactive | 0 | Windows | Windows 10 |

Figure 3.9: Autraff UI - Client Dashboard

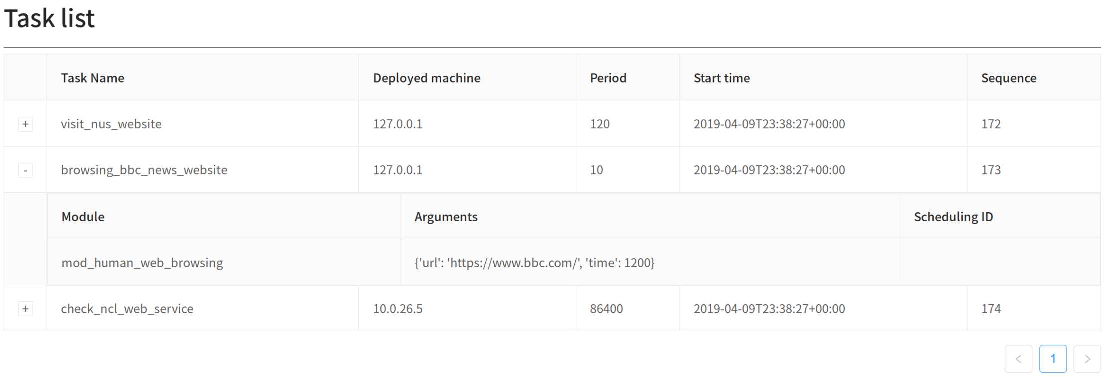

Refresh Client Connection Status
Refresh Client Schedule Status
1

## 127.0.0.1

Status: connected ✓

| Name | Schedule ID | Module | Interval (seconds) | Start Time | Actions |
|---|---|---|---|---|---|
| visit_nus_website | | mod_visit_any_page | 120 | 2019-04-09T23:38:27+00:00 | schedule stop |
| browsing_bbc_news_website | | mod_human_web_browsing | 10 | 2019-04-09T23:38:27+00:00 | schedule stop |

Figure 3.10: Autraff UI - Client Management for client-127.0.0.1



## Task list

<table>
<tr><th></th><th>Task Name</th><th>Deployed machine</th><th>Period</th><th>Start time</th><th>Sequence</th></tr>
<tr><td>+</td><td>visit_nus_website</td><td>127.0.0.1</td><td>120</td><td>2019-04-09T23:38:27+00:00</td><td>172</td></tr>
<tr><td>-</td><td>browsing_bbc_news_website</td><td>127.0.0.1</td><td>10</td><td>2019-04-09T23:38:27+00:00</td><td>173</td></tr>
<tr><td></td><td colspan="5"><table><tr><th>Module</th><th>Arguments</th><th>Scheduling ID</th></tr><tr><td>mod_human_web_browsing</td><td>{'url': 'https://www.bbc.com/', 'time': 1200}</td><td></td></tr></table></td></tr>
<tr><td>+</td><td>check_ncl_web_service</td><td>10.0.26.5</td><td>86400</td><td>2019-04-09T23:38:27+00:00</td><td>174</td></tr>
</table>

Figure 3.11: Autraff UI - Task List for All

### 3.4.4 Use Case by Flow Chart

In this section, a flow chart that will show how a typical user may use Autraff as a traffic generating orchestrator in a distributed system scenario will be shown. The flow chart will mainly focus on the most typical and the most normal cases, some edge cases will not be discussed here.

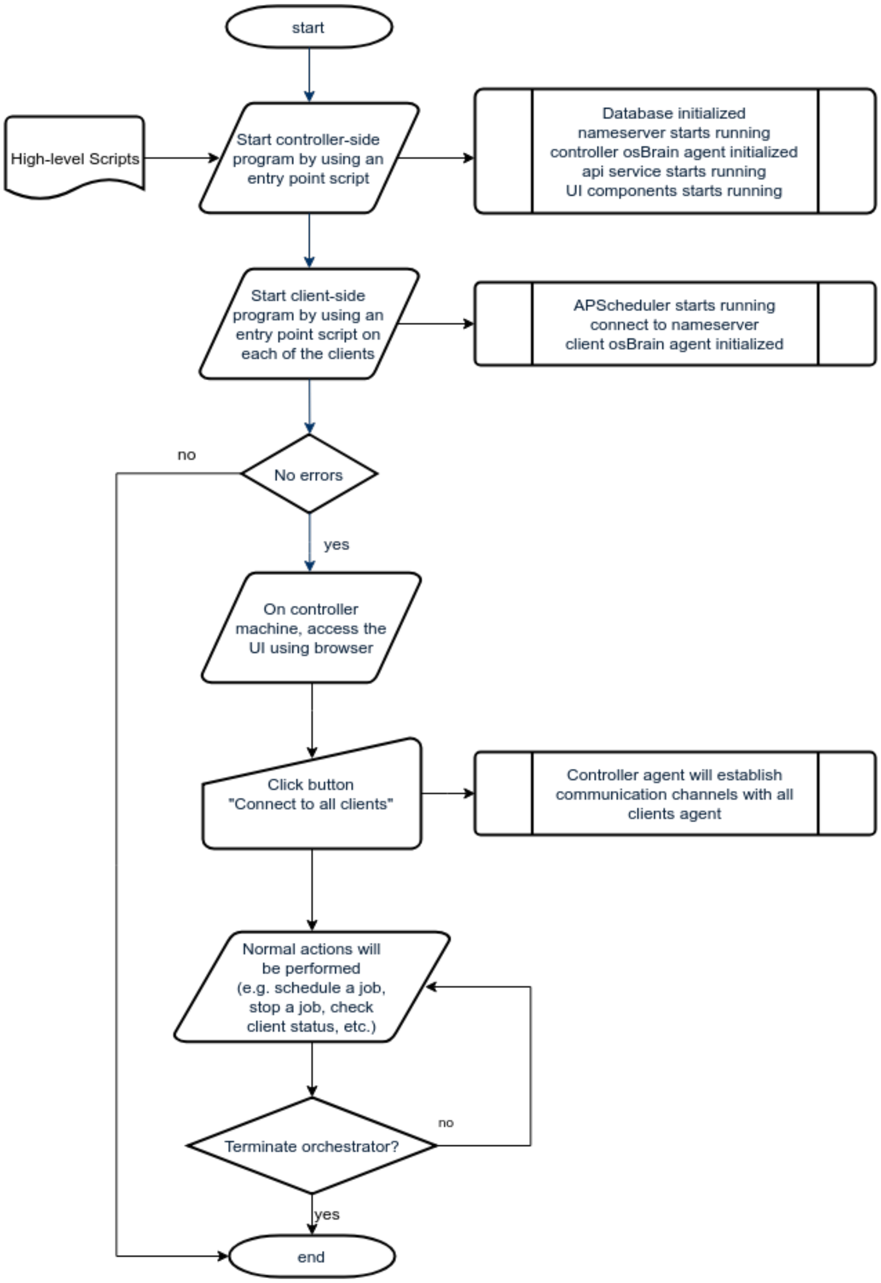

start
High-level Scripts
Start controller-side program by using an entry point script
Database initialized
nameserver starts running
controller osBrain agent initialized
api service starts running
UI components starts running
Start client-side program by using an entry point script on each of the clients
APScheduler starts running
connect to nameserver
client osBrain agent initialized
no
No errors
yes
On controller machine, access the UI using browser
Click button "Connect to all clients"
Controller agent will establish communication channels with all clients agent
Normal actions will be performed (e.g. schedule a job, stop a job, check client status, etc.)
Terminate orchestrator?
no
yes
end

Figure 3.12: A typical use case by flow chart

Basically, after the user starts up the controller-side program (by executing an entry point script), he or she needs to go to each client to start up the client-side program there. And then, the user will need to make the controller agent connect to every client agent and establish communication channels between them. This will be achieved by clicking a single button in the Autraff UI. There are also some other ways to do so through the UI, but they will be not discussed in this report. And after these are all done, the user can start to perform scheduling process for traffic generating.

## 3.5 Summary

This chapter mainly introduced about our proposed traffic generating orchestration framework – Autraff. Autraff framework followed the client-server architecture, designed and implemented with very high extensibility and modularity. The users of Autraff framework may write a high-level script to define and construct a behavior-tree-like traffic generation process for each client. He or she can also implement his or her own client-side modules for more complex protocols and applications used to generate various types of network traffics since each module is just a standalone python script and there is almost no restriction on their capabilities.

# Chapter 4

# Literature Review: Human Web Browsing Simulation

In this section, we will briefly present some of the current research/studies about human web browsing modelling/simulation, and discuss about their feasibility and limitations.

## 4.1 Random and Probabilistic Model

The easiest and the most trivial method to generate a huge amount of continuous web traffic is to use random and probabilistic method. By using this method, traffics are generated by sending HTTP requests through URLs that are randomly selected from a target website or webpage. Then the staying time for each web page will depend on Poisson or Pareto distribution (Busari and Williamson, 2002). This method is widely used in testing environment to test the load capacity of a web server. For example, *ab - Apache HTTP server benchmarking tool* is using this trick as a load testing method (The Apache Software Foundation, 2018).

However, as pointed put by Gavrilis et al. (2007), this method may end up in some URL requests that are all irrelevant to each other or link to some web pages where no useful information is contained, such as an advertisement page. In such case, it will be easy to find out that these traffics are not aligned with human behaviors.

## 4.2 Server-side Web Log Analysis Model

Lots of researchers have studied on data-mining technologies or machine learning techniques to analyse the web server logs to predict and simulate human behaviors. These methods are frequently used by web developers to enhance and improve their websites and web servers (Tyagi et al., 2010) (Chen and Chau, 2005).

This method was implemented by many researchers such as (Schechtera et al., 1998), who proposed a way to predict the sequence of HTTP request based on the path profiles logged by the web server. While as data mining and machine learning techniques become more and more advanced, by applying these methods on the server-side logs can always give a very accurate prediction of an HTTP request sequences that could be injected into the network if traffic generation is our primary goal.

Nevertheless, since in our scenario of generating background traffic for a CDX, we usually do not possess any server logs of the websites or web service that we would like to crawl. Secondly, in our framework and software requirements for the traffic generator, we are supposed to use real tools and clients (i.e. a web browser) for the traffic generation. If we merely use the HTTP request sequence for traffic injection, some hidden traffics that come from the application layer will be missed. Thus, this method will not be an ideal way for us to generate human-behavior-like web browsing traffic.

## 4.3 Content Analysis and Interest-driven Browsing Model

When a real human is looking at a web page, the actual content of the page may play a significant role in influencing and impacting how he or she will continue browsing. Several studies done by Shen et al. (2005) and Kuljis (2010) have shown that, by analysing the content of a web page, we can derived a so-call possibility matrix to show how human will transit from one topic to another topic while browsing the web.

Besides, in the study by Yang (2016), it has also depicted how the content of the current page may have influence on human's interest in a certain topic. The underlying theory is supported by the study of Barabsi (2005), who illustrated that as an event or activity occurs repeatedly, a person's interest in this event or activity will firstly increase to a peak point before dropping gradually.

Based on these theory, Yang (2016) has proposed a method to simulate human behaviors using content analysis with the concept of Markov process, which is also a very popular mathematical model that is widely used to simulate and predict web traffic state, such as a study done by Antoniou et al. (2007). And Yang (2016) has also validated this method with a very considerable accuracy (however, by the way, there is no universally adapted method to judge how close a URL request sequence is closed to the real human behavior, but the researcher defined his own method of validation.).

## 4.4 Summary

There are mainly three trending methods to simulate traffics that produced by real human while browsing the web. While the random/probabilistic model and server-side log analysis model are not suitable for our scenario, the method proposed by Yang (2016) using content analysis and interest-driven theory may be a feasible method for us to implement.

# Chapter 5

# Mathematical Modelling: Human Web Browsing Simulation

In this section we will mainly be focused on the mathematical modelling for our human web browsing simulation model for our traffic generator. The implementation of this model will be used as one of the client-side modules in Autraff, with the capability to generate continuous and human–behavior–like web browsing traffics. Here we will be mainly using the theory introduced by Yang (2016) with some enhancement and improvements.

## 5.1 Terminologies

Before we start, we would like to first clarify some terminologies used in this chapter.

| **Term** | **Meaning** |
|---|---|
| Page | Referring to a single web page. |
| Title | The page title, which will normally be wrapped with a META tag in HTML dom. |
| Content | Referring to the main context of a web page. It is usually the main information that a web page wants to deliver. |
| Link | Referring to all the hyper-link elements that appear on a certain web page.<br>A link may consist of a clickable url and a piece of text that appear on the web page. |
| Theme | Referring to the main topic derived from a piece of content or a link. |

Besides, there are also some newly defined concepts and nouns we would like to introduce here:

- **content quality**: A factor to indicate how high is the quality of the content of a page.
- **content closeness**: Between two pages, content closeness defines how their contents are similar to each other.
- **theme closeness**: Usually between the page title and each of the page links, theme closeness will define how a page title is similar to the text of a link.
- **visibility closeness**: It defines how likely a link is able to be seen by a human on a certain page. Normally, a link that appears on the top will have a higher visibility closeness.

## 5.2 Assumption

By using the model we will be introducing in this chapter, we will firstly assume that all the web pages that the client will be visiting are mostly text-based, which means that text will be the main media and information that will be delivered on this page. Other media such as audios and videos will not be considered as the content of a web page. This assumption also aligns with the truth that most of the modern websites are still focused on text-based content.

## 5.3 Key Concepts behind

1. As illustrated in the study of Barabsi (2005), the frequency of people to conduct a certain event or activity and people's interest in such event or activity are having some relations, which could be interpreted by the figure shown below:
2. Human web browsing behaviors will also follow the pattern described above. But instead of saying conducting a certain type of event, we will rephrase it as “browsing web pages of a certain topic”. And instead of saying the frequency, we will rephrase it as “how willing a person will want to keep browsing about this topic”.

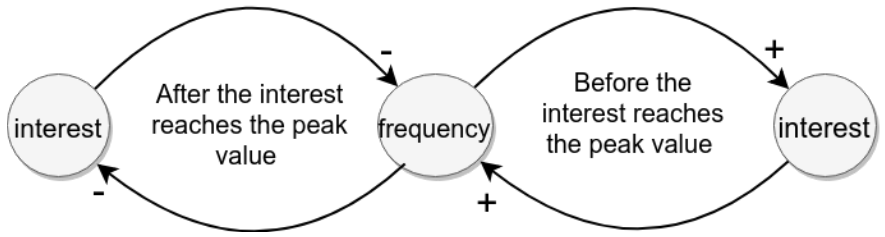

interest
After the interest reaches the peak value
frequency
Before the interest reaches the peak value
interest
-
-
+
+

Figure 5.1: Relationship between people's interest and frequency of doing some activities or conducting some events.

## 5.4 Model Design

### 5.4.1 Factors influencing the web browsing behaviors

Based on the key concepts introduced above, Yang (2016) has proposed the following model which defines all the factors that would influence the human behaviors of browsing and how they are dynamically related to each other in a certain page:

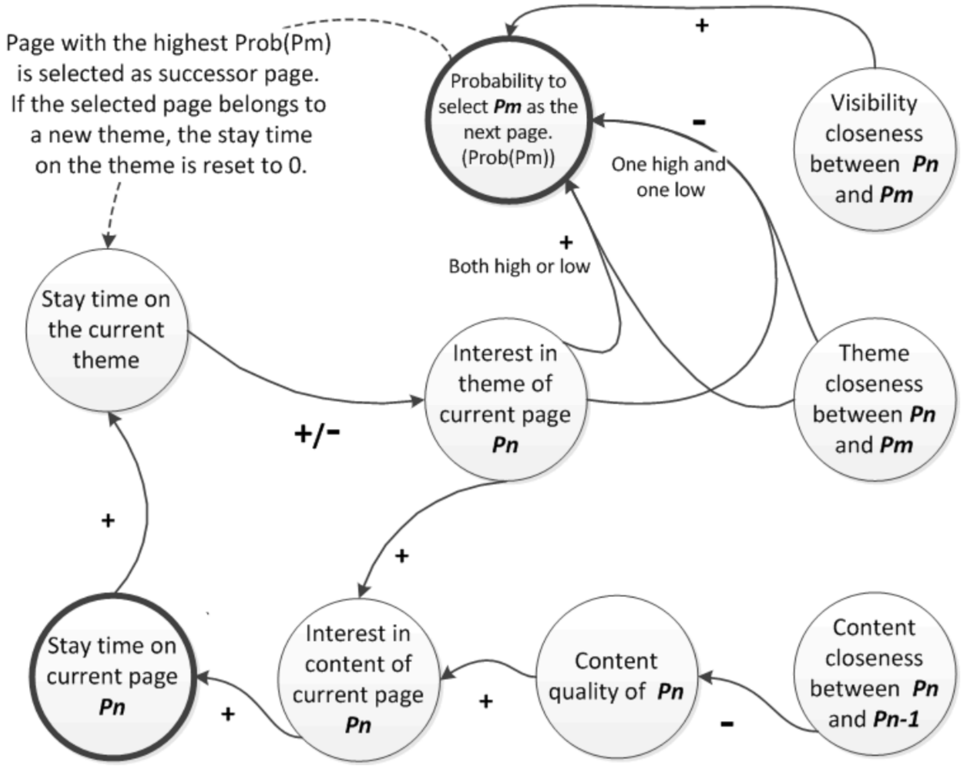

Page with the highest Prob(Pm) is selected as successor page. If the selected page belongs to a new theme, the stay time on the theme is reset to 0.
Probability to select Pm as the next page. (Prob(Pm))
+
Visibility closeness between Pn and Pm
-
One high and one low
+
Both high or low
Stay time on the current theme
Interest in theme of current page Pn
Theme closeness between Pn and Pm
+/-
+
+
Stay time on current page Pn
Interest in content of current page Pn
Content quality of Pn
Content closeness between Pn and Pn-1
+
+
-

Figure 5.2: Dynamic relationships among all factors that will influence human web browsing behaviors

In the figure above, current page is denoted as $P_n$, while $P_{n-1}$ and $P_m$ are denoting the previous page and the next page respectively. Here $P_m$ is not referring to a specific page, but just one of the pages that are linked with page $P_n$ through all the links in $P_n$.

### 5.4.2 General flow of browsing behavior simulation

By applying the model above, we derived a general flow of how to simulate the human browsing process. This flow is an enhanced extension and a modified version of the study of Yang (2016). While the flow is listed below, the actual math formulas will be introduced in the next section.

1. Start visiting from a web page.
2. Compute interest in the theme of current page.
3. Compute the staying time on current page.
4. Compute theme closeness and visibility closeness for every link on current page.
5. Compute probability to be selected for every link.
6. Wait until the staying time is reached.
7. Choose a link and visit the url base on the probabilities calculated in the previous step.
8. Repeat from step 2.

### 5.4.3 Mathematical Details and Formulas

In this section, we will mainly introduce all the math formulas that we used to calculate the factors that will influence the browsing behaviors.

**Interest in the theme of current page $P_i$**

$$c_i = Max(0, \frac{c'-1}{{d_m}^2}(d_i - d_m)^2 + 1)$$

Here, $c'$ and $d_m$ are constant values which denote the initial interest value of a new theme and the point of time when interest are about to drop respectively.

### Theme closeness and content closeness

Theme closeness will be compute between the title of the page and the text of a link, while the content closeness will be computed between two page contents. In the study done by Yang (2016), WordNet word domain is used for linguistic analysis and cosine similarity is used to compute the closeness.

However, we are not using the exact same way since computation using this method may take up too much time which may potentially delay the simulation process. Instead, we are applying a method proposed by Mihalceaand et al. (2006), which will still be using WordNet word domain. This method has been tested and validated with the correctness above 85%. Basically by applying method, when computing the closeness or similarity between two sentences (the content of a page can be considered as a long sentence), we compute the domain similarity of each word in sentence one with each word in sentence two. Then we will cumulate all the scores and divide by the product of the word counts in sentence one and two. In the real application, the domain similarity between two words are calculated using NLTK python library, and the time spent on calculation will be much less then calculating the cosine similarity.

In later sections, we will use $S(i,j)$ to denote the theme closeness between the title of $P_i$ and the link in $P_i$ that could connect to $P_j$, and $S'(i,k)$ to denote the content closeness between $P_i$ and $P_k$.

### Interest in page content

In the following formulas, $L(i)$ denotes the length or word count of the content of page $P_i$, and $L_{max}$ denotes the longest content that the client has been met so far.

As for the interest level in page content: if $i = 0$:

$$q_i = \frac{L(i)}{L_{max}}$$

if $i > 0$:

$$q_i = \frac{L(i)}{L_{max}}(1 - S'(i, i-1))$$

**Staying time for page $P_i$**

$$t_i = \frac{L_{max}}{constant\_human\_reading\_speed} \cdot c_i \cdot q_i$$

**Visibility closeness for every link in page $P_i$**

$$V(i,j) = 0.5 - 0.5sin((\frac{Loc(i,j)}{L_i} - 0.5)\pi)$$

Here, $Loc(i,j)$ denotes to the location of the link, or the count of the words that appear before this link in current page.

**Probability of click on the link that connects to page $P_j$**

$$E(i,j) = \varphi \cdot V(i,j) + (1-\varphi) \cdot \frac{2 \cdot c_i \cdot S(i,j)}{c_i{}^2 + S^2(i,j)}$$

Here $\varphi$ defines how significantly will the visibility closeness will impact the final probability. Similarly, $(1-\varphi$ will define how significant of factors such as interest in current theme and content closeness. Regardless of the effect of visibility closeness, the probability can be depicted using the graph below, where a larger circle denotes a larger probability to be clicked.

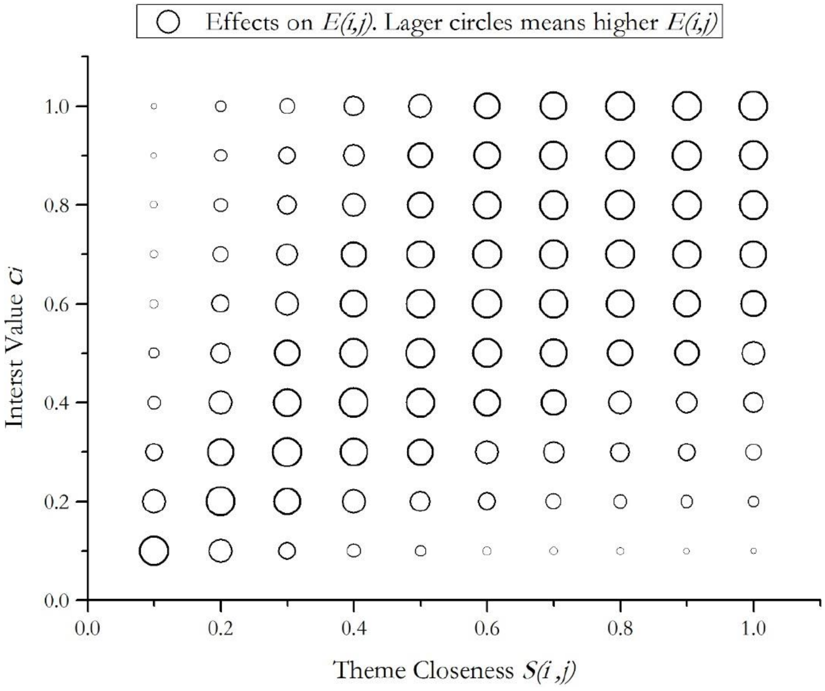


Figure 5.3: Probability to be clicked of all the links in a page

## 5.5 Implementation

The implementation of this module was achieved by python with NTLK library and Selenium framework. The logic flow has been depicted using a flow chart below:

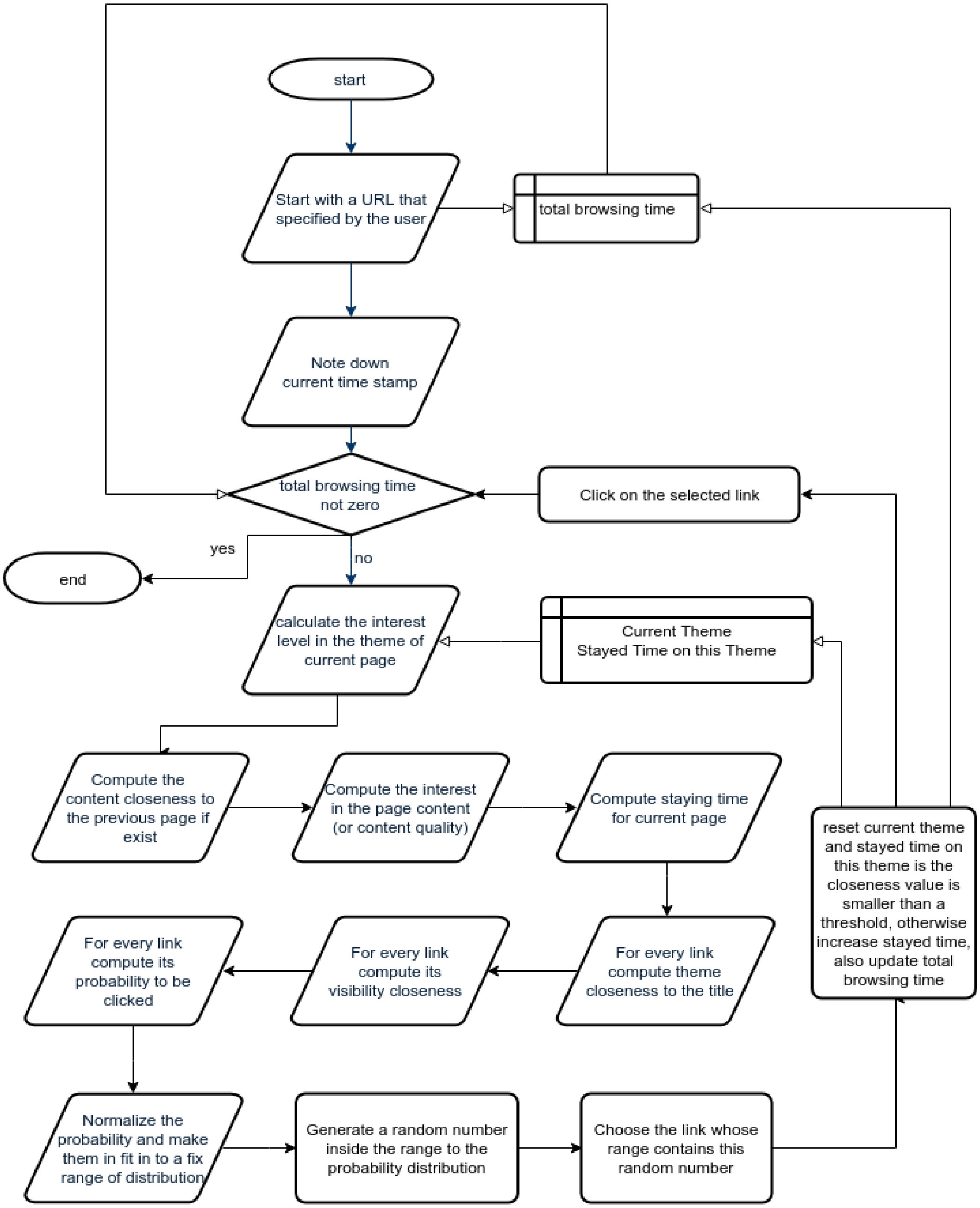

start
Start with a URL that specified by the user
total browsing time
Note down current time stamp
total browsing time not zero
Click on the selected link
yes
end
no
calculate the interest level in the theme of current page
Current Theme
Stayed Time on this Theme
Compute the content closeness to the previous page if exist
Compute the interest in the page content (or content quality)
Compute staying time for current page
reset current theme and stayed time on this theme is the closeness value is smaller than a threshold, otherwise increase stayed time, also update total browsing time
For every link compute its probability to be clicked
For every link compute its visibility closeness
For every link compute theme closeness to the title
Normalize the probability and make them in fit in to a fix range of distribution
Generate a random number inside the range to the probability distribution
Choose the link whose range contains this random number

Figure 5.4: Logic flow of module "mod_human_web_browsing"

## 5.6 Enhancement using Machine Learning Model

Since currently, all of our flows are generated and derived based on pure mathematical analysis and probabilistic, the judgement of the validity and accuracy may not be very intuitive, as also pointed out by Yang (2016) in his study. Besides, the current mathematical model is based on the current studies about human behaviors, which may evolve or change over the years. In such case, by introducing some machine learning techniques, we could easily address the two issues mentioned above.

Based on the requirements of this human behavior simulation module, we may apply supervised machine learning models to fulfil our needs.And here we will use Artificial Neural Network (ANN) as an example. Since we will be still doing context analysis on a webpage to derive a possible link that could be "clicked", and also derive the staying time for current web page, we could build two machine models and use all the factors mentioned in section 5.4.1 to be the features of our machine learning model. One of the models will be taking all the features such as content closeness, and interest in theme to give a prediction on expected staying time on current page, while the other one will be taking all links with their visibility closeness, theme closeness and interest in current theme as input and then give prediction on the possibilities to be clicked for all the links. Thus we could choose a link to proceed to accordingly.

Moreover, small procedures in our model such as calculating the content closeness or the interest in current theme can also be enhanced or enriched by some classificational machine learning algorithms, which may also apply the knowledge in the field of Natural Language Processing.

The proposal of machine learning models here are not of the main scope of this project. However, we would like to propose it for any future work and further research on this topic.

## 5.7 Summary

In conclusion of this chapter, based on the mathematical model we have built according to the theory proposed by Yang (2016), we have delivered an Autraff module that could simulate human's behaviors while browsing the web. On each page, staying time and the possibilities to click on every single link are calculated. And the next step will just depend on the “state” of current page, which can be essentially regarded as a Markov chain model, which has been applied with some theory of web content analysis and human interest-driven behaviors as raised by Barabsi (2005). Finally, we also proposed that some of our calculation and prediction could be enhanced by some trending machine learning methods, which could be furtherly studied in the future.

# Chapter 6

# Testing and Validation

The official testing and validation process will be done in the virtual network provided by National Cyber-security Lab (NCL). The Autraff framework will be installed and deployed in this network with one of the server being the controller and others being the clients. Autraff will be tested mainly for its reliability and stability to generate continuous and authentic web traffic, which could help to eliminate all the human intervention and replace all the human efforts described in section 1.2.

However, for now, there may lack a universally accepted way to validate whether the web traffics generated by our "mod_human_web_browsing" indeed present a high similarity to the human footprints while browsing a website. So the validation work for this module may become a future work for developers or researchers who are going to continue with this topic.

# Chapter 7

# Conclusions

In this chapter, we are going to discuss about some limitations about our current framework as well as the mathematical model that we used to simulate human web browsing behaviors.

## 7.1 Limitations and Future Work

### 7.1.1 Autraff Orchestration Framework

1. In order to configure and initialize each client, the user of Autraff may still need to go to each client machine and manually start up the client-side program, which may cause a lot of asynchronous processes happening at the same time, introducing another potential point of failure. Therefore, to eliminate this issue, some provisioning management tools such as Ansible could be used in future development, to configure and start up the client-side program right after the controller-side turns up.

2. In the current version of Autraff, it is not possible to set the start time of a job as “now” in the high-level script. It is because of the truth that the controller may not be able to communicate with all the clients right after it is turned up (the previous limitation has already explained why). Solving this issue is also something that could dramatically improve the user experience in Autraff. The solution to this issue could vary from initiating all client-side program using provisioning management tools as mentioned in the previous item, to modifying the way of how the client and controller establish communication channel , so that each time the

client is up, it can establish the connection with the controller and the controller can send the scheduling request accordingly.

3. Currently, Autraff does not support any changes on jobs during the actaully generation of network traffics. Each change will imply the deletion of a current job and the creation of a new job. Therefore, in future development or enhancement, dynamic planning or scheduling of jobs could be implemented and supported.

4. As autraff now is mainly designed to produce normal network traffic during the CDX, it could be probably extended to support automated attacking process as well since it was designed with high extensibility and there is no restriction on the capabilities of each single module used on the client-side server. Therefore, in the future, we could extend our framework to support similar functions as Cryton (Nutar, 2018), or make it backward compatible with Cryton in terms of its high-level scripts used.

### 7.1.2 Human Web Browsing Simulation Module

The major limitation of the module we proposed here is that pure mathematical model may be difficult to validate and easy to be out-of-date as we mentioned in the last chapter. Therefore, we have sketched a machine learning solution which will make use of all the factors we introduced for content analysis as training features. In such manner, the result and the accuracy will be easier to be judged, and it will also give a more intuitive way for the researchers to refine the model.

## 7.2 Summary

As the needs of generating sufficient and human behavior-like web traffic in the CDX have been increasing fast these days, especially for the needs from NCL, we have designed and implemented an orchestration framework, Autraff, for CDX operators to easily automate and manage all the traffic generation actions. Autraff supports singular job scheduling as well as job scheduling that follows a behavior tree of actions. Moreover, we also implemented a client-side module by structuring a mathematical model that would be used to simulate human web browsing behaviors to further help Autraff to generate some background traffic during the CDX. Though some limitations may exist for current work, we have addressed some of them and provide with some potential solutions and methods of enhancement in the previous section, which will leave some future work for further study and development.

# Bibliography

Antoniou, C., Koutsopoulos, H. N., and Yannis, G. (2007). Traffic state prediction using markov chain models. published in 2007 European Control Conference (ECC).

Barabsi, A.-L. (2005). The origin of bursts and heavy tails in human dynamics. *Nature*, 435:207–211.

Busari, M. and Williamson, C. (2002). Prowgen: A synthetic workload generation tool for simulation evaluation of web proxy caches. *Computer Networks*, 38:779–794.

Chen, H. and Chau, M. (2005). Web mining: Machine learning for web applications. published by University of Arizona.

Gavrilis, D., Chatzis, I., and Dermatas, E. (2007). Flash crowd detection using decoy hyperlinks. published in IEEE International Conference on Networking, Sensing and Control.

Gebert, S., Pries, R., Schlosser, D., and Heck, K. (2012). Internet access traffic measurement and analysis. In Pescap, A., Salgarelli, L., and Dimitropoulos, X., editors, *Traffic Monitoring and Analysis*, pages 29–42.

Grnholm, A. (2019). Apscheduler user manual. `https://apscheduler.readthedocs.io/en/latest/`.

Hirner, E. (2018). Web browsing simulator for needs safety exercise. unpublished.

Kuljis, J. (2010). Applying content analysis to web-based content. *Journal of Computing and Information Technology - CIT 18*, 4:369–375.

Mammadov, S., Mehta, D., Stoner, E., and Carvalho, M. M. (2017). High fidelity adaptive cyber emulation.

Mihalceaand, R., Corley, C., and Strapparava, C. (2006). Corpus-based and knowledge-based measuresof text semantic similarity. published in: Proceeding AAAI'06 Proceedings of the 21st national conference on Artificial intelligence - volume 1, pages 775-780 .

Nicolau, M., Perez-Liebana, D., ONeill, M., and Brabazon, A. (2017). Evolutionary behavior tree approaches for navigating platform games. published by IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI IN GAME.

Nutar, I. (2018). Automation of complex attack scenarios. unpublished.

Open Sistemas de Informacin Internet S.L. (2019). osbrain user manual. `https://osbrain.readthedocs.io/en/stable/about.html`.

Schechtera, S., Krishnanb, M., and D.Smithc, M. (1998). Using path profiles to predict http requests. *Computer Networks and ISDN Systems*, 30:457–467.

Shen, X., Dumais, S., and Horvitz, E. (2005). Analysis of topic dynamics in web search.

Shoulson, A., Garcia, F. M., Jones, M., Mead, R., and Badler, N. I. (2011). Parameterizing behavior trees. *Motion in Games. MIG 2011. Lecture Notes in Computer Science*, pages 144–155.

The Apache Software Foundation (2018). ab - apache http server benchmarking tool documentations. urlhttp://httpd.apache.org/docs/2.2/programs/ab.html.

Tyagi, N. K., Solanki, A. K., and Wadhwa, M. (2010). Analysis of server log by web usage mining for website improvement. *IJCSI International Journal of Computer Science*, 7:17–21.

Unknown (2017). Dumpsterfire toolset user manual. `https://github.com/TryCatchHCF/DumpsterFire/tree/master/FireModules`.

Unknown (2018). Autottp user manual. `https://github.com/jymcheong/AutoTTP`.

Yang, Y. (2016). Modeling of humanweb browsing based on theory of interest-driven behavior. published by York University.