

DESIGN AND IMPLEMENTATION OF CLICK ADMIN — A WEB-BASED NETWORK RESEARCH PLATFORM

BY ABDUL HASSAN

A thesis submitted to the
Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Master of Science
Graduate Program in Electrical and Computer Engineering

Written under the direction of
Janne Lindqvist
and approved by

New Brunswick, New Jersey

May, 2014

© 2014

Abdul Hassan

ALL RIGHTS RESERVED

ABSTRACT OF THE THESIS

Design and Implementation of Click Admin — A Web-based Network Research Platform

by Abdul Hassan

Thesis Director: Janne Lindqvist

This thesis describes the design and implementation of Click Admin, a web-based network research platform. Click Admin was designed to be a robust tool for network administrators to be able to research how people would prefer to use their network when presented with different scenarios, e.g., different queueing types, different queueing priorities, individual bandwidth allocation, offered incentives for cooperative users, etc. The main goals of Click Admin are to allow network administrators to (1) gather data on how their users would prefer to use the network in any given scenario, (2) configure the flow of traffic to simulate the given scenario, and (3) monitor and analyze how users actually use the network in the given scenario.

At its core, Click Admin is merely an interface to Click Modular Router, which is a flexible, yet fast modular router toolkit allowing network administrators to reroute traffic in any number of ways. The research capabilities of Click Admin come in the form of a survey system that network administrators can use to design unique experiments and present different scenarios to users, asking them to specify how they would prefer to use the network in each scenario. After determining how each user would prefer to use the network, the Click Modular Router is then dynamically configured to simulate the

given scenario. During each simulation, basic network analysis tools are used to capture the flow of traffic, for the purpose of monitoring and analyzing how users actually use the network in each scenario. With these tools, network administrators can gain insight not only into how users use the network, but how they would prefer to use the network in any number of different real-world scenarios, and can possibly even predict future trends in network usage and improve network performance.

Acknowledgements

First and foremost, I would like to thank my advisor, Professor Janne Lindqvist, for giving me the opportunity to work on this interesting and meaningful research tool. His guidance and encouragement has helped me to successfully complete this thesis, while his patience has allowed me to do so with very little stress. I would also like to thank my mother, Regina, for her unsurpassed love and support. My father, Mahmoud, for his advice and guidance. My entire family as a whole, including my good friend Cynthia, for helping me stay motivated and high-spirited. And lastly, the faculty and staff at WINLAB for all of their help and encouragement over the past few years. I'd like to give a special thanks to Professor Christopher Rose, who has helped me grow and progress not only as a student, but also as an individual.

Dedication

I dedicate this thesis to my family, as well as my good friend Cynthia, for all of their love and support throughout the years. They have helped me succeed by seeing me through some very difficult times.

Table of Contents

Abstract	ii
Acknowledgements	iv
Dedication	v
List of Tables	viii
List of Figures	ix
1. Introduction	1
1.1. Motivation	1
1.2. Thesis Overview	2
2. Background and Related Work	3
2.1. Home Watcher	3
2.2. Click Modular Router	4
2.3. Tcpdump	5
3. Design of Click Admin	8
3.1. Problem Statement	8
3.2. Statement of Goals	9
3.3. Overview of Proposed Solution	9
3.4. Assumptions	10
3.5. System Requirements	10
3.6. Architecture	14
3.6.1. Components	16
3.7. Persistent Data Storage	19
3.7.1. Database Structure	19

3.7.2. File Format	22
3.8. Data Structures	23
3.9. Algorithmic Models	24
3.10. User Interface	25
3.11. Testing	25
4. Implementation of Click Admin	26
4.1. Architecture	26
4.2. Subsystems and Libraries	26
4.3. Persistent Data Storage	32
4.3.1. Database Structure	32
4.4. Testing	36
5. Results	38
5.1. Installing Click Admin	38
5.2. Creating an Experiment	39
5.3. Scheduling an Experiment	43
5.4. Running an Experiment	46
6. Conclusion and Future Work	48
References	49

List of Tables

3.1. Format of PCAP file	23
------------------------------------	----

List of Figures

2.1. Example of a Click configuration file	5
2.2. Example of a Click configuration diagram in Clicky	6
2.3. Data from pcap file displayed in Wireshark	7
3.1. Cluster of load-balanced application servers	15
3.2. Flow of an MVC application	17
3.3. Relationship Diagram of Minimal Database	22
3.4. Representation of a singly linked list	23
3.5. Representation of a doubly linked list	24
4.1. Relationship Diagram of Implemented Database	36
5.1. Create Experiment Page	39
5.2. Create Survey Page	40
5.3. Create Survey Option Page	41
5.4. Create Configuration Page	42
5.5. Edit Experiment Page	44
5.6. View Experiment Page	45

Chapter 1

Introduction

With the assortment of network analysis tools available on the market today [1], network administrators have access to an abundance of data which all together details how traffic flows over the network. This data can be used for many different purposes, for example, detecting problems within the network, monitoring individual bandwidth, spotting patterns and trends, reporting usage statistics, etc. This data can ultimately provide valuable knowledge and insight into how people use the Internet as a whole.

As a network administrator, having access to this information is essential in order to improve certain aspects of the network, predict future load, and detect problems before they arise. However, it's important to note that this information only provides knowledge on how people *currently* use the network. More so, it only describes usage in *typical* network scenarios, e.g., fair queueing, fair bandwidth allocation, quality of service, etc. What if people were subjected to *atypical* network scenarios? How would usage patterns change then? Currently, no tool exists that can provide this sort of knowledge and insight into how people would prefer to use the network in a possibly not so typical scenario.

1.1 Motivation

The motivation behind Click Admin came from Dr. Janne Lindqvist, who wanted to simply gather data on how people would use the network when offered different kinds of incentives. For example, would people be willing to cooperate and share bandwidth with each other if it meant optimal usage and bandwidth for everyone involved? Would

people be willing to sacrifice access to the network during certain parts of the day if it meant better access during other parts of the day? How would traffic be prioritized if it were up to the users themselves. These are the types of questions that we wish to be able to answer with Click Admin.

1.2 Thesis Overview

We begin in chapter 2 with a short discussion on background and related work. This includes further explanation on why Click Admin was built, as well as a description of some of the tools essential to Click Admin. Chapter 3 describes how Click Admin was initially designed — in a format similar to a design document [2]. We explain the initial goals and requirements of Click Admin, as well as the different decisions made for the purpose of meeting those goals. We try to explain, as best we can, the rationale behind some of those decisions. Chapter 4 describes how we implemented a prototype of Click Admin. We explain the different constraints placed upon us while actually building the prototype, and the different modifications made to the design to compensate. We describe the different tools used, why those tools were chosen, and how they integrate with each other. Chapter 5 discusses the final result. We explain how to setup and configure Click Admin, and we go through a sample experiment to illustrate its real-world usage. Finally, chapter 6 concludes this thesis by discussing possible future enhancements and other related work.

Chapter 2

Background and Related Work

2.1 Home Watcher

Research shows that when constraints are placed upon a network (whether home, mobile, or otherwise), usage of the network is impacted. For instance, studies show that placing a cap on bandwidth causes users to be more critical of the applications they use, and to regulate any usage by others when multiple users are involved [3,4]. Chetty et al. [4] studied what would happen in a household if each person had more control over how resources were allocated.

They created a tool called Home Watcher to both display and limit each persons bandwidth usage in real-time [4, p. 3]. They deployed Home Watcher in multiple homes [4, p. 4] and found that usage patterns changed when users were made more aware of how they and others used the network. Having this information, users were able to optimize their use of the network [4, p. 6]. It allowed them to learn more about, e.g., what applications were the most data intensive, what times of the day the network was more/less congested, how much they affected everyone else on the network, etc.

In addition to this, they found that having control over others resources caused confusion, especially in households where there was a defined social order [4, p. 7]. Users found it difficult to decide when it was appropriate to limit someone else's resources — how were they to prioritize usage of the network. While some used the tool playfully (e.g., roommates playing jokes on each other), others were more worried about creating tension within the household (e.g., children were hesitant to limit their parents). Ultimately, the study showed that visualizing resource usage in a home lead to a better

understanding, as well as a difference in usage patterns.

2.2 Click Modular Router

Click Modular Router [5, 6] is a very powerful modular router toolkit, and is the very foundation of Click Admin. It's an open source (MIT license) platform for building flexible, configurable routers. Routers are setup with flat text configuration files, which define a directed graph of elements from input(s) to output(s), as seen in Figure 2.1. Click also comes bundled with Clicky [7], which is a GUI used to view configuration files as block diagrams, as seen in Figure 2.2. Every packet going through the router is passed (not copied) from one element to the next and processed by each element in the graph. Click can be compiled as a user-level program or as a kernel module for Linux, allowing direct interaction with device drivers. Running in the Linux kernel of conventional PC hardware, an unoptimized Click IP router can achieve a maximum loss-free forwarding rate of 357,000 64-byte packets per second, which is only about 5% slower than Linux alone (due to overhead) [5, p. 93].

Click comes with a plethora of elements directly out of the box, and each element is responsible for a specific task. For instance, the Counter element [8] passes packets unchanged from its input to its output, maintaining statistics information about packet count and packet rate. Other useful elements include the IPClassifier element [9] which allows for packets to be classified in a similar way to tcpdump (e.g., filtered by source/destination address), and the BandwidthShaper element [10] which allows for passage of said packets to be limited to a maximum bandwidth.

While Click comes packaged with enough elements to build very complex routers [11], the true power and flexibility comes from the extensibility of Click and the ability to create your own elements. Elements are written in C++ using an API provided by Click. Each element has input and output ports which hook together to create a chain, and can even take arguments to make them more dynamic.

```

// testdevice.click

// Tests whether Click can read packets from the network.
// You may need to modify the device name in FromDevice.
// You'll probably need to be root to run this.

// Run with
//   click testdevice.click
// (runs as a user-level program; uses Linux packet sockets or a similar
// mechanism), or
//   click-install testdevice.click
// (runs inside a Linux kernel).

// If you run this inside the kernel, your kernel's ordinary IP stack
// will stop getting packets from eth0. This might not be convenient.
// The Print messages are printed to the system log, which is accessible
// with 'dmesg' and /var/log/messages. The most recent 2-4K of messages are
// stored in /click/messages.

FromDevice(eth0) -> Print(ok) -> Discard;

```

Figure 2.1: Example of a valid Click configuration file, which defines a directed graph of elements. The graph begins with *FromDevice*, which intercepts packets from a given network device (*eth0* in this case). Packets are then passed on to the *Print* element, which simply prints packet contents. Lastly, packets are passed on to *Discard*, which simply drops all packets.

2.3 Tcpcdump

Tcpcdump [12–14] is a very powerful command line packet analyzer. It's open source (BSD license) and known to be the premier network analysis tool, used by network administrators and security professionals alike, as well as anyone with a desire to better understand TCP/IP in practice [15]. Tcpcdump works by intercepting packets transmitted to or received by the computer it's running on, and thus works best when used on a system acting as a router or gateway, capturing all packets flowing over the network [12].

Tcpcdump comes with a long list of options [16]. Below is a short list of options to highlight it's power.

-i *interface* Listen on *interface*.

-w *file* Write the raw packets to *file* rather than parsing and printing them out.

-r *file* Read packets from *file* (which was created with the **-w** option).

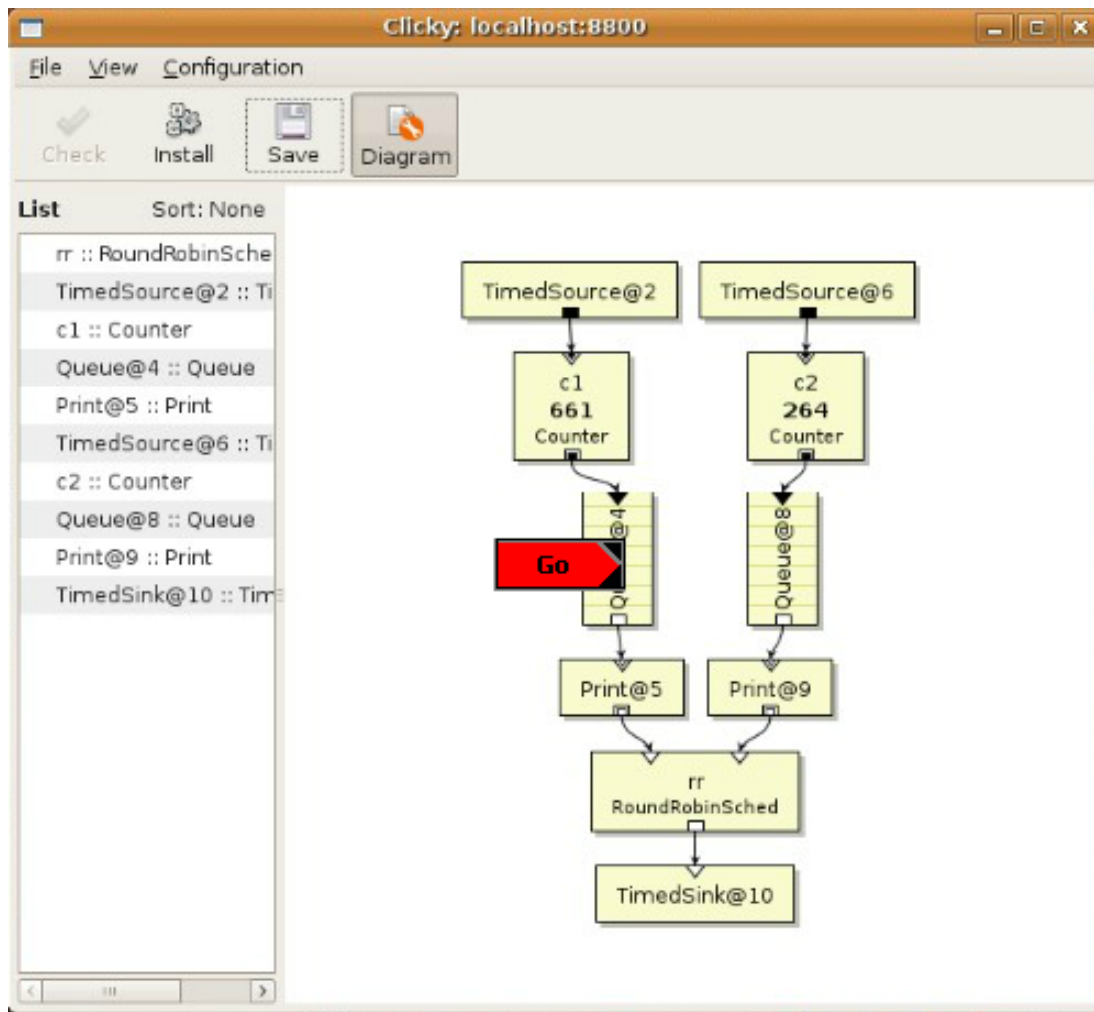


Figure 2.2: Example of a valid Click configuration diagram visualized in Clicky.

- e Capture link-level header.
- n Don't convert addresses to names.
- X Print the contents (in addition to the header) of each packet in both hex and ASCII format.
- v, -vv, -vvv When parsing and printing, produce more and more verbose output – each additional **v** captures more fields.

Although there are many more options [13], using only this short list, an administrator can troubleshoot issues within the network, view communications transferred over the

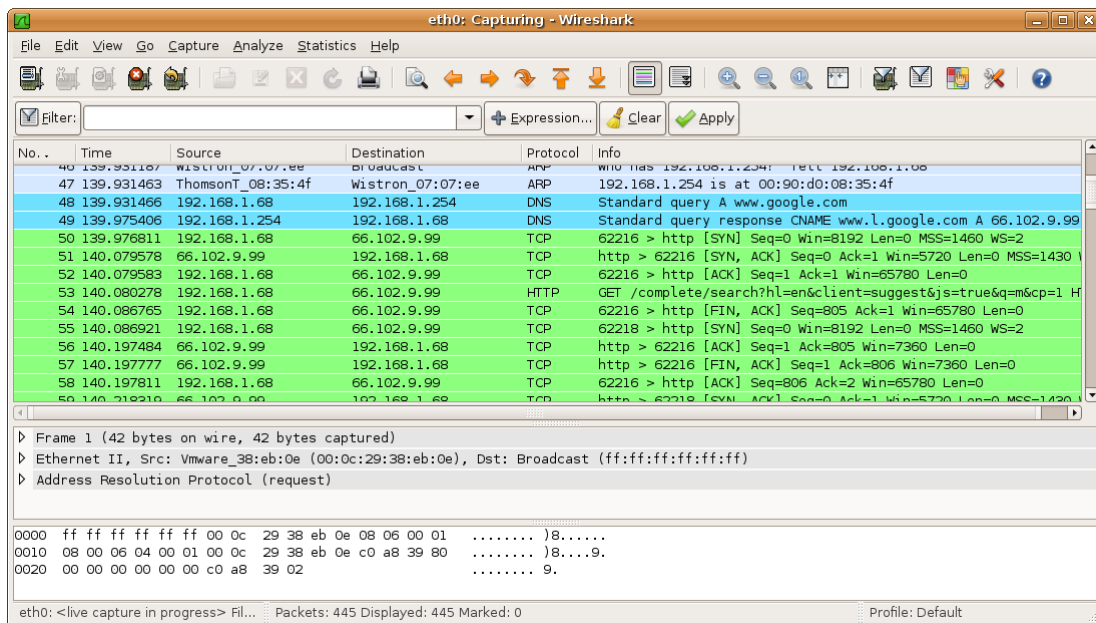


Figure 2.3: Data from pcap file displayed in Wireshark.

network (as long as it's unencrypted, e.g., HTTP), store packets to a file for later analysis, and much more. Tcpdump captures and stores packets in pcap format – which is a fairly natural representation of each packet [17]. However, it doesn't do much more than this, which places the burden of analysis directly on the user. Analyzing pcap files directly can be a daunting and tedious experience, that is, without the use of third party tools such as Wireshark [18], which can read pcap files and assist in the analysis.

Chapter 3

Design of Click Admin

The design of Click Admin took a top-down approach [19] — all goals and requirements were gathered first before any tools were chosen or any code was written. This chapter highlights the architecture and design, listing the compiled goals and requirements of Click Admin, while foregoing any implementation specific details.

3.1 Problem Statement

People use the Internet everyday [20] under a set of basic assumptions. A large majority of users are oblivious to what actually happens "behind the curtains" [21, 22]. They are unaware of how packets are scheduled and prioritized. They don't know that the level of acceptable packet loss is dependent on the application being used – if they know about packet loss at all. They don't know what bandwidth is and why their connection may seem "slow" during certain parts of the day [23].

Usage of the network is often a greedy endeavor, because people fail to realize that their usage ties up resources which may affect other people on the same network. But what if people were more knowledgeable, and were made more aware of how the network was configured? How would usage patterns change then? More so, what if they themselves had more control in how things were configured? How would they choose to set things up? Lastly, what if people had the opportunity to cooperate with each other? Would they be willing to do so if it meant optimal usage for everyone involved?

There are plenty of tools available that network administrators can use to monitor how people *currently* use the network [1], but there is no tool available that can provide

insight into how people would *prefer* to use the network if they had more knowledge and control in how things were configured, or if they knew how others used the same network. We want to build such a tool, allowing network administrators to gain knowledge and insight into how people would choose to use the network when presented with any number of different real-world scenarios.

3.2 Statement of Goals

The main goal of this project is to design and implement a research platform that will allow network administrators to:

1. Gather data on how their users would prefer to use the network in any given scenario.
2. Configure the flow of traffic to simulate the given scenario.
3. Monitor and analyze how users actually use the network in the given scenario.

3.3 Overview of Proposed Solution

To gather data on how people would prefer to use the network in a given scenario, we will create a web-based survey system (website). Each survey will belong to an experiment, and each experiment will consist of multiple (related) survey questions. Each experiment will illustrate a scenario, and each survey will ask users a question related to how they would most likely react in that scenario. The network administrator will be responsible for asking the question as well as providing all possible options to the question. Users will need to register to the website and create an account in order to participate in the experiment and answer each survey question.

To configure the flow of traffic and simulate a given scenario, we will incorporate a configurable router into the system. Some simulations may depend on a users response to the survey questions mentioned above (e.g., minimum acceptable bandwidth), and so router configurations will be dynamic in nature. In order to create these dynamic

configurations, we will incorporate the use of a template engine [24]. The network administrator will be responsible for creating these templates. Each configuration will belong to an experiment, alongside other (related) configurations and surveys.

To monitor how users actually use the network, we will incorporate a network monitoring tool into the system. Users will need to periodically notify the system of their host machine’s [25] network address(es) so that network administrators can later match packets to the users who sent/received them to monitor each users individual network usage.

3.4 Assumptions

While the proposed solution will attempt to cover as many cases as possible, we will operate under the following assumptions:

- A1:** All users belong to the same local network, and all network traffic is routed through the system, which acts as a gateway router [26]. (Actual router setup is beyond the scope of this thesis, but many guides exist online [27].)
- A2:** Each user will have a unique (and static) IP/MAC address and will only access the network using the machine(s) they have registered in the system.
- A3:** Network administrators will be honest when creating an experiment. Packets should be routed exactly as advertised.

3.5 System Requirements

The following requirements are both functional and non-functional [28–30], describing system functionality and behavior, while also suggesting system architecture.

- R-1:** In order to keep track of each user and their responses, user registration is required in order to use the system. To register, users must enter a valid email address and password. When registering, users should read over and agree with the terms of service [31]. They should also read over and consent to the privacy policy [32].

- R-2:** In order to keep track of each machine in the network, host registration is required to use the system. To register, users must enter the IP and MAC address of the machine they plan to use during an experiment — it's assumed they will use this machine and only this machine. This information is used when rendering configuration files and determining network usage for each host, and thus each user.
- R-3:** To keep information up to date, users should be able to update their account and host information. This includes changing their IP address, MAC address, email address, and password.
- R-4:** Administrators should be able to add, review, edit, and delete any content on the site. However, they should not be able to add, edit, or delete any user information, aside from activating a users account, inactivating a users account, adding roles to a user (e.g., make a user an administrator), or removing roles from a user. This is to ensure that all registered users are authentic and have went through the registration process in its entirety.
- R-5:** Host information should always be current. If this information becomes more than two weeks old, users will not be able to use the system until this information is explicitly refreshed.
- R-6:** When creating an experiment, administrators must provide a title. They can optionally add a description to explain the experiment in much more detail, letting users know what they can expect during the experiment.
- R-7:** When creating a configuration, administrators must provide a duration for how long the configuration will run in the router. Durations range from 1 to 24 hours, in 1 hour increments, allowing configurations to run for 1 hour, 2 hours, 3 hours, etc. They can also provide an optional template. Templates will be given access to data related to the experiment and the users involved, such as each users IP/MAC address and their response to each survey, to make configurations dynamic. Templates should render to a valid Click router file. If no template is

provided, a default template should be used which simply passes packets along untouched. When adding multiple configurations, an order must be specified in such a way that each configuration runs once and only once, one after the other.

R-8: Surveys can be added to an experiment if necessary. When creating a survey, administrators must provide the question as well as all possible options to the question. For example, a question could be *"How important is email to you?"*, and options for such a question could be (1) *very important*, and (2) *not very important*. The only requirement is that at least two options be provided for each survey. An optional description can also be added to the survey to explain the survey in more detail, and to explain the possible consequences of a response. For instance, the consequence of choosing *not very important* to the previous question could lead to email packets being made a lower priority and more susceptible to packet loss during the simulation.

R-9: Experiments can be scheduled to run at a specified time. Administrators can only schedule and run experiments containing at least one configuration. They must be scheduled to start at a future time and can't start within 24 hours of scheduling the experiment — allowing users adequate time to familiarize themselves with the experiment and answer any and all survey questions. Experiments must start at the top of the hour, and they must not conflict with any other scheduled experiments.

R-10: Administrators should be able to fully edit any experiment that is not yet scheduled, or is scheduled but has not yet started. This includes the experiment itself, any and all of its configurations, surveys, and options. Once an experiment has started, it can not be edited or deleted, for archiving purposes.

R-11: Users should review the details of an experiment and answer any and all survey questions before it starts. A reminder to do so should be sent out prior to the start of the next experiment, reminding users to familiarize themselves with the experiment and answer any and all survey questions.

- R-12:** Users should be able to view all scheduled experiments, past, present, and future. They should be able to view statistics and a breakdown of user responses for any and all surveys. However, users should have no knowledge of an experiment that is not yet scheduled.
- R-13:** During times when no experiment is running, the system should continue to function as a basic gateway router so that traffic can continue to flow.
- R-14:** All traffic flowing through the system should be logged, regardless of whether or not an experiment is running, to continuously monitor each users use of the network.
- R-15:** Administrators should be able to set and change the terms of service and privacy policy to account for any possible future changes made to the code and how the data is collected and used.
- R-16:** Reliability should be the highest priority, being that the system is meant to be a gateway router. Any downtime or interruptions, especially during an experiment, is disastrous.
- R-17:** Extensibility should be a high priority, being that the system is intended to be a research *platform*.
- R-18:** The code base should remain lean, to allow for easy and quick navigation when trying to extend the code. With this in mind, features should be kept to a minimum.
- R-19:** Feature settings should be as configurable as possible. For instance, the administrator should be able to easily change the max duration a configuration can run from 24 hours to 48 hours, if necessary.
- R-20:** Security does not need to be a major concern, aside from basic and common sense security measures. System administrators can implement more security features as needed, depending on their own needs.
- R-21:** Aside from the speed in which Click routes packets — which is more or less

dependent on hardware — speed should not be a major concern. The website is intended to be fairly simple, and most of the heavy lifting occurs outside of any web context. Any resource intensive processes should still be performed in the background.

R-22: The install and setup process should be quick, easy, and automated.

3.6 Architecture

According to the IEEE, software architecture refers to the “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution” [33]. It defines that which is fundamental to the system as a whole, determining its form, function, value, cost, and risk. Although many applications exist, most of which can be considered different and unique, the system requirements for these applications often share some commonalities, such as data storage, reliability, security, etc. Software architectures provide basic blueprints to apply to these problems, and depending on the nature of the problem, there is likely an existing blueprint that one can apply. In our case, the following blueprints make the most sense and should be followed as closely as possible.

Request/Response

Being web-based, the system will naturally adhere to the request/response architecture [34], where clients make requests and the system processes those requests, performing any necessary actions and returning any necessary data. This means that the system will need to be accessible to clients and documented to help clients easily navigate through the system. A more typical service architecture [35], however, is rather unnecessary as there’s no need for the application to be agnostic.

Shared Nothing

The system should be based on a shared-nothing architecture [36]. This means that the application should be written in such a way that each instance (node) would be independent and self-sufficient, to the point where a node can be added or removed without worry. Structuring it this way would allow us to load-balance [37] between the nodes, making the system more efficient and reliable.

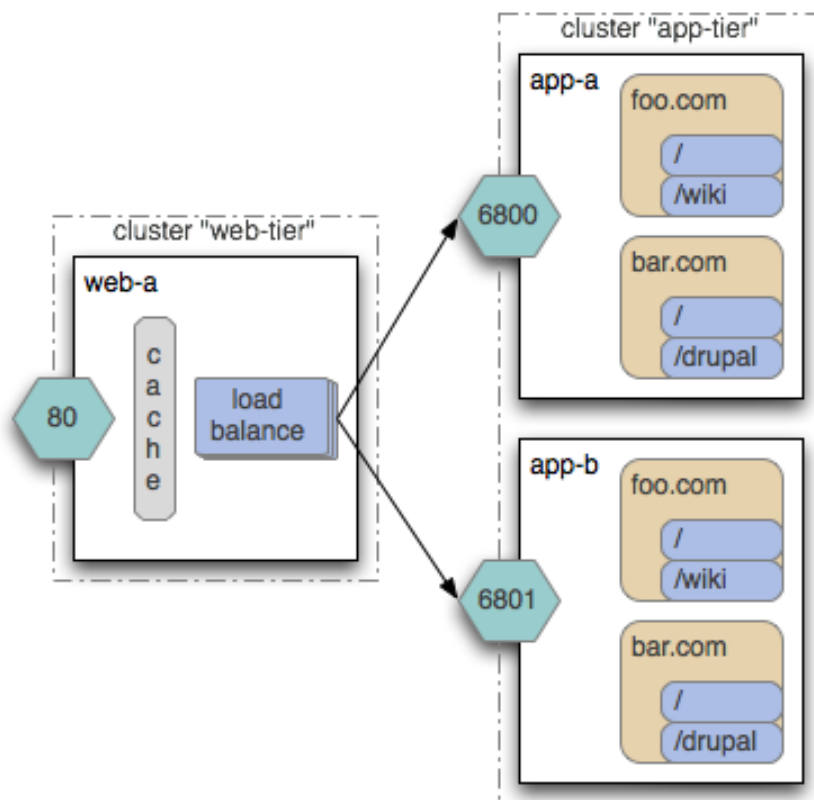


Figure 3.1: A cluster of application servers being load-balanced by a web-server.

Multi-Tiered

Because reliability is such a major concern, the system should be based on a multi-tiered [38,39] architecture. This means that different components should be physically separated and reside on different machines, which means more resources available for

each component. At the very least, the router should reside on its own machine, with possible backups in place for redundancy, so that errors in other components will not affect the network. Adhering to the shared-nothing architecture will also allow instances of the application to be separate and reside on different machines. As a side effect, performance should also increase, although that is not a major concern.

MVC

Because extensibility and code simplicity is a major concern, the application should be based on an MVC (Model - View - Controller) [40, 41] architecture. This means dividing the application into three interconnected parts, where each part deals with its own (major) concerns. This makes navigating and editing the codebase much easier. The *model* represents knowledge. It holds the business logic of the application and is responsible for interacting with the database, storing and querying data. The model should closely resemble a real-world object (e.g., in our case, a survey). The *view* is a visual representation of the model, and is responsible for building the user interface, taking data from the model and arranging it to fit the proper format (e.g., in our case, HTML). The *controller* is the link between the model and the view. It is responsible for processing incoming requests from clients, validating and formatting user input, and calling the necessary functions on the model, passing any data to the correct view which gets sent back to the client.

3.6.1 Components

The system will consist of several major components (layers), each separate and responsible for their own set of concerns. Being a web-based system, the following components are necessary at the very least.

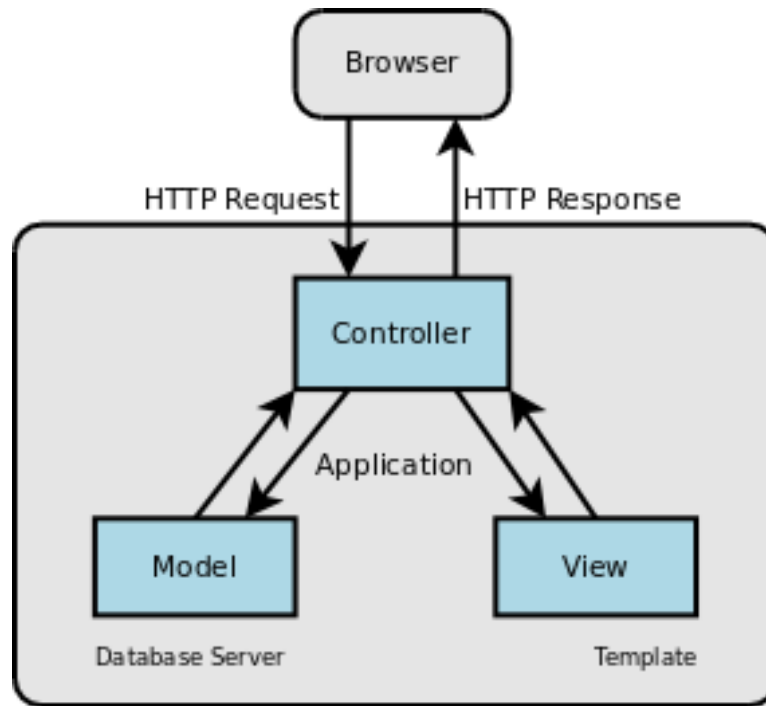


Figure 3.2: Flow of an MVC application.

Configurable Router

A configurable router will take on arguably the biggest responsibility in the system. It will be responsible for routing packets over the network. All simulations presented in the experiments will be performed by the router. As mentioned, it's assumed the router will function as a gateway router [26], so it's important that the router chosen be both efficient and reliable.

Network Monitor

A network monitor, also known as a packet analyzer [42], will take on another big responsibility. It will be responsible for monitoring and recording packets routing through the system, and will run at all times to keep track of how people use the network. Because the system is meant to operate as a gateway router, *all* traffic over the network should route through the system and be logged.

Application Server

An application server [43] can be seen as a framework providing a structured approach to the application. This is where our application code will reside and run to process client requests, serving as the core of our web-based survey system. The application server will interact with most other components directly. It will interact with the database server to store and query data. It will interact with the configurable router to update its configuration. It will push emails to the mail server to deliver them, and push tasks to the job scheduler to run certain tasks at specified times. Because extensibility is a high priority, the framework chose should be simple and easy to use.

Database Server

A database server [44] will make up our back-end and will be responsible for storing all of our application related data (experiments, users, etc.). It will take requests from the application server to store, update, and query data quickly and efficiently.

Web Server

A web server [45] will be responsible for accepting incoming requests from clients and forwarding them to the application server. As mentioned, the application server will process these requests and return a response for the web server to return back to the client.

Mail Server

A mail server [46] is necessary to deliver email notifications. It will be egress only, as there is no need for the system to receive emails.

Job Scheduler

A job scheduler [47] is necessary to control background execution of different tasks, such as updating the router at the start of a new experiment, or sending reminder email notifications to users and administrators.

3.7 Persistent Data Storage

As mentioned, the system will need to store several sets of data. All together, we need to keep track of experiment related data (experiment titles, survey questions, etc.), user related data (email addresses, ip addresses, etc.), and packet data (traffic over the network captured by the network monitor). All data submitted to the application should be stored in a database, as previously mentioned, while all data captured by the network monitor should be stored in binary files. There is no need to store packet data in the database, as it's up to the administrator to decide if and how this data is actually used, as well as what tools are used to analyze it.

3.7.1 Database Structure

At the very least, the database structure should resemble Listing 3.1.

Listing 3.1: Designed using SQLite 3.7.17

```
CREATE TABLE users (
    id INTEGER NOT NULL,
    email VARCHAR(255) NOT NULL,
    password CHAR(60) NOT NULL,
    -- Host information is required, but nullable, since it's not
    -- required at sign up
    ip_address VARCHAR(39), -- IPv4 or IPv6
    mac_address VARCHAR(17),
    -- Keep track of the last time host information was updated,
    -- since it should be updated every two weeks
```

```

        last_updated_at DATETIME,
        PRIMARY KEY (id),
        UNIQUE (email)
    );

```

```

CREATE TABLE experiments (
    id INTEGER NOT NULL,
    title VARCHAR(140) NOT NULL,
    description TEXT,
    -- Null until schedule is set
    start_time DATETIME,
    PRIMARY KEY (id),
    UNIQUE (title)
);

```

```

CREATE TABLE configurations (
    id INTEGER NOT NULL,
    experiment_id INTEGER NOT NULL,
    -- Self-referencing table
    previous_configuration_id INTEGER,
    -- Written in template language
    template TEXT,
    -- In hours
    duration INTEGER NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY(previous_configuration_id) REFERENCES
        configurations (id),
    FOREIGN KEY(experiment_id) REFERENCES experiments (id) ON
        DELETE cascade
);

```

```

CREATE TABLE surveys (
    id INTEGER NOT NULL,
    experiment_id INTEGER NOT NULL,

```

```

        question VARCHAR(140) NOT NULL,
        description TEXT,
        PRIMARY KEY (id),
        FOREIGN KEY(experiment_id) REFERENCES experiments (id) ON
            DELETE cascade
    );

CREATE TABLE options (
    id INTEGER NOT NULL,
    survey_id INTEGER NOT NULL,
    response VARCHAR(140) NOT NULL,
    PRIMARY KEY (id),
    UNIQUE (survey_id, response),
    FOREIGN KEY(survey_id) REFERENCES surveys (id) ON DELETE
        cascade
);

-- Stores each users chosen response to a survey question
CREATE TABLE user_choices (
    user_id INTEGER NOT NULL,
    survey_id INTEGER NOT NULL,
    option_id INTEGER NOT NULL,
    PRIMARY KEY (user_id, survey_id, option_id),
    UNIQUE (user_id, survey_id),
    FOREIGN KEY(user_id) REFERENCES users (id) ON DELETE cascade,
    FOREIGN KEY(survey_id) REFERENCES surveys (id) ON DELETE
        cascade,
    FOREIGN KEY(option_id) REFERENCES options (id) ON DELETE
        cascade
);

-- Stores miscellaneous data, such as the terms of service and
    privacy policy
CREATE TABLE pages (

```

```

    id INTEGER NOT NULL,
    name VARCHAR(80) NOT NULL,
    title VARCHAR(255) NOT NULL,
    content TEXT,
    PRIMARY KEY (id),
    UNIQUE (name)
);

```

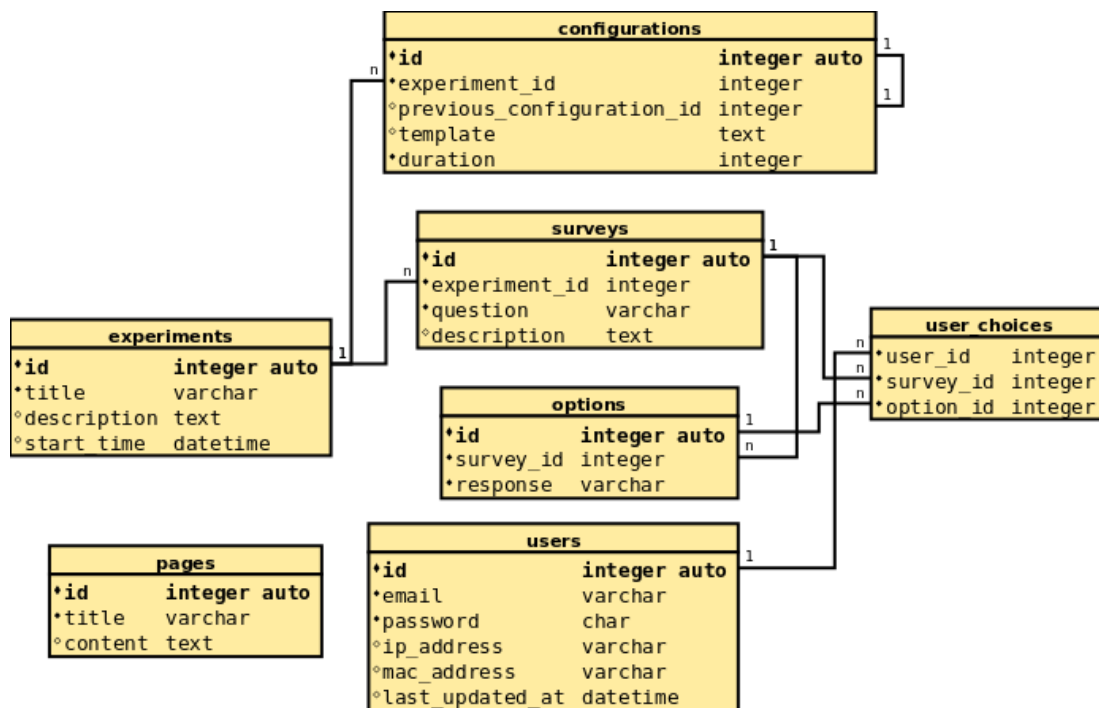


Figure 3.3: Relationship diagram of the minimal database.

3.7.2 File Format

As mentioned, packet data should be stored in binary files. Many open source network monitoring tools are implemented using the PCAP API (libpcap) [17], which defines the structure used to store packet data. Details of the PCAP API are beyond the scope of this thesis, but Table 3.1 shows a visual representation of the PCAP file format.

Global Header	Packet 1 Header	Packet 1 Data	Packet 2 Header	Packet 2 Data	...	Packet N Header	Packet N Data
---------------	-----------------	---------------	-----------------	---------------	-----	-----------------	---------------

Table 3.1: Format of PCAP file [48]. Packet headers are added by libpcap (or winpcap), packet data is what is actually captured (raw data). The global header is added once by libpcap (or winpcap).

3.8 Data Structures

Within the application, data should be stored in very basic structures (lists, sets, dictionaries, etc.). Aside from what's provided by the application framework, there is almost no need for any complex data structures, as the design doesn't call for such. The only need for something complex stems from the requirement that multiple configurations within an experiment run and progress in a specified order set by the administrator. Configuration sets should be represented in the application as a linked list [49] to keep track of the order and progression of configurations from one to the next. A linked list is simply a chain of similar objects, where each object contains its own set of data, as well as a reference to the next object in the chain, as seen in Figure 3.4.

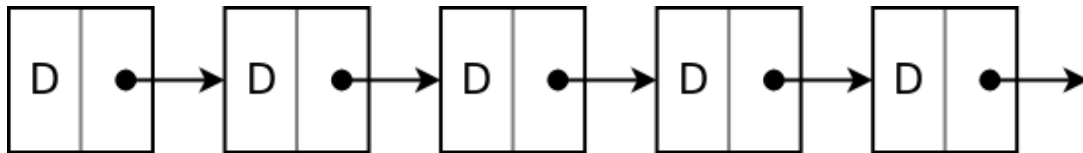


Figure 3.4: A singly linked list containing five nodes. Each node has its own set of data, as well as a reference to the next node in the chain.

Because configuration data is stored in a database, the order of configurations must also (primarily) be stored and represented in the database. With this in mind, it's simpler if, in the database, each configuration contain an explicit reference to the *previous* configuration, as opposed to the *next* configuration. Doing it this way would make finding the first configuration much easier, as it would be the only configuration with no explicit reference to another.

Also, because the data is stored in a database, it's very simple to find the next configuration through a simple query. So while there is no explicit reference, in the database,

to the next configuration, this information can be considered known. With this in mind, it makes more sense for configuration sets to be represented in the application as a doubly linked list, with references to both the previous and the next configuration in the chain, as seen in Figure 3.5.

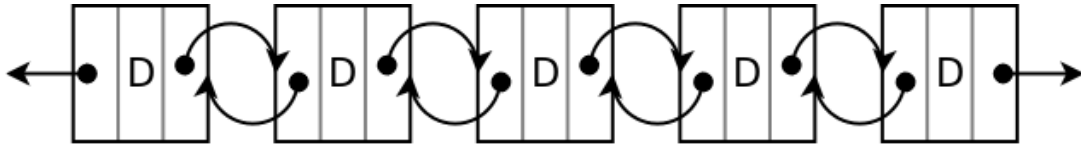


Figure 3.5: A doubly linked list containing five nodes. Each node has its own set of data, as well as a reference to both the next and previous node in the chain.

3.9 Algorithmic Models

Because of the simplicity of our design, the application does not call for any complex models or algorithms. The only need for something somewhat complex stems from the requirement that each configuration within an experiment run once and only once, one after the other. The need for this algorithm goes hand in hand with the need for a linked list data structure. Since the administrator is the one to specify the order of configurations, there needs to be a way to check and ensure the validity and unambiguity of the chosen order. There must be no gaps, circular references, or any other ambiguity in the chain of configurations.

The algorithm to check for this is pretty simple. First, identify the head of the chain. Starting at the head, traverse the chain from one configuration to the next, keeping a list of each configuration visited. For every configuration encountered, check the list of configurations already visited. If it's determined that a configuration is visited more than once, it's because there is a circular reference in our chain, which is an error. If not, and the end of the chain is reached, the next thing is to make sure that all configurations were visited. The length of visited configurations should be equal to the number of total configurations in the set. If this is not the case, it's because there is a gap in our chain, which is an error.

3.10 User Interface

In keeping with the minimalism and simplicity of the system, the user interface should also be simple and easy to use for users and administrators alike. It should adhere to the principles of user interface design [50] — structure, simplicity, visibility, feedback, tolerance, and reuse. From a user’s perspective, the application should essentially be nothing more than a collection of experiments and survey questions. The interface should allow them to log on, review experiments, and answer questions, while hiding most other functionality. For administrator’s, the interface should also allow them to manage content, similar to a CMS (content management system) [51].

3.11 Testing

Testing ensures the quality of the final product, and there are many different levels of software testing, including unit testing, integration testing, component testing, system testing, and acceptance testing [52]. In any system, testing is encouraged and welcome. In this case, at the very least, system testing is a must to check the system as a whole, and to ensure that all requirements are met. Second to that, component testing should be done to ensure that each component does what it should do, and that information and commands are passed between components as expected.

Chapter 4

Implementation of Click Admin

While building the prototype for Click Admin, we were presented with a few different constraints, such as time, cost, resources, and even our own requirements, which forced our hand and caused us to deviate slightly from the design. This chapter describes the details of our implementation, highlighting any deviations in design. (Where not explicitly mentioned, it can be assumed that the implementation follows the design presented in chapter 3.)

4.1 Architecture

For the most part, the prototype was built to adhere to the architectures discussed in chapter 3. Click Admin was designed to be multi-tiered [39], however, due to limited resources, the prototype is single-tiered [53] — all components (including the router) run on the same machine.

4.2 Subsystems and Libraries

Our choice of tools was based on both our constraint for time as well as our requirement for extensibility. In order to build a system that others could freely add upon or make changes to, we were forced to use open source software with flexible and permissive licenses. To build the prototype as quickly as possible, we used tools that were easy to learn and configure, and which integrated well with each other.

Linux/Ubuntu

Linux [54] is a free and open source operating system. Contrary to popular belief, Linux is very user-friendly, with many utilities that make everyday tasks simple. It's very reliable, with many servers experiencing years worth of uptime [55]. It's also very secure, with most security threats being patched by one of the many contributors worldwide as soon as they are discovered.

Ubuntu [56] is one of the many distributions [57] of Linux. Ubuntu adds to the user experience, making it one of the easiest distributions to use. Ubuntu, specifically version 12.04 (Precise Pangolin), was chosen as our operating system because of its ease of use, and the fact that most open source tools are made to work primarily on Linux.

Click

As mentioned in chapter 2, Click [5, 6, 58] is a powerful modular router toolkit, used to build flexible and configurable routers. We use Click, specifically version 2.0.1 to configure the network and simulate our experiments.

Tcpdump

As mentioned in chapter 2, Tcpdump [13, 14] is a powerful command line packet analyzer, used to intercept packets sent over the network. We use Tcpdump to log and monitor all traffic on the network.

Python

Python [59, 60] is a general-purpose, high-level dynamic programming language allowing for quick prototyping. It's philosophy emphasizes readability and simplicity in code. While the language is easy to use, it's powerful nonetheless, with an extensive list of useful libraries [61] which make programming even easier. Python, specifically version 2.7 (CPython), was chosen to power our website.

Flask

Flask [62, 63] is a lightweight, yet powerful microframework for Python web development. It's considered a microframework because it's simple, yet extensible, leading to less dependencies and more freedom on how to write and organize code. As opposed to more monolithic frameworks, such as Django, which is very tightly coupled and makes certain decisions for you [64]. The power of Flask comes from its extensibility. Currently, Flask supports many different extensions, providing such functionality as database integration, form validation, upload handling, authentication, authorization, etc. Flask is based on Werkzeug and Jinja2, these being its only dependencies. We chose to use Flask as our framework to make building our website quick and easy.

Flask-Admin

Flask-Admin [65, 66] is a Flask extension providing a framework for building flexible administrative interfaces for Flask applications. We use Flask-Admin to build a CMS [51], providing administrators with the ability to create and edit experiments, as well as perform any other admin related tasks.

Flask-Mail

Flask-Mail [67, 68] is a Flask extension providing the ability to send emails from within a Flask application. We use Flask-Mail to send reminder emails to users, as well as warning/error messages to administrators when necessary.

Flask-Security

Flask-Security [69, 70] is a Flask extension providing security functionality, such as user registration and login, to a Flask application. We use Flask-Security to ease the user management aspect of our system, and to restrict access and functionality to users based on their privileges.

Flask-SQLAlchemy

Flask-SQLAlchemy [71, 72] is a Flask extension that adds SQLAlchemy [73, 74] support to an application. SQLAlchemy is an ORM (object relational mapper) [75] for Python with support for many different SQL databases, such as MySQL and PostgreSQL. An ORM converts data found in the database to objects (and vice versa) for easy data retrieval and manipulation. It's an abstraction layer which allows the use of objects directly, removing the need to write, by hand, most SQL queries. We use Flask-SQLAlchemy to aid in the development of our models and data access layer.

Flask-WTF

Flask-WTF [76, 77] is a Flask extension that adds WTForms [78, 79] support to a Flask application. WTForms is a form and validation library for Python, allowing developers to (abstractly) build complex HTML forms and validate user input on submission. We use Flask-WTF to build our HTML forms and validate user input.

Jinja2

Jinja2 [80, 81] is a powerful, full featured template engine [24] for Python. It's one of the most used template engines for Python, with full unicode support, template inheritance, JIT compilation, and many other features. We use Jinja2 within our Flask application to build our HTML and Click router templates.

PostgreSQL

PostgreSQL [82, 83] is a powerful and advanced ORDBMS (object-relational database management system) [84]. It's fully featured and debated as one of the best database systems available, used by such prominent users as Reddit and Yahoo. PostgreSQL offers more data types than most other database systems, such as the *interval* type, which holds time intervals (e.g., 5 hours, 6 days). This comes in handy when storing

configurations, as we no longer need to store the duration as an integer. SQLAlchemy maps a PostgreSQL *interval* to a Python *timedelta*, which is an added benefit for quick and easy datetime calculations. We use PostgreSQL to store all of our user and experiment related data and archives.

Nginx

Nginx [85, 86] is a high-performance reverse proxy server which works for HTTP, HTTPS, SMTP, POP3, etc. It's also a high-performance web server, load balancer, and cache server. It has a strong focus on high concurrency and is one of a handful of servers which addresses the C10K problem [87]. Nginx currently serves nearly 12.18% of active sites across all domains, powering such sites as Netflix, Pinterest, and even Facebook. We use Nginx as a reverse proxy for our application server(s), allowing for load balancing between them. We also use it as a static file server, serving all of our CSS, JavaScript, and media files.

uWSGI

uWSGI [88, 89] is an ambiguously named high-performance application server, which implements the similarly named WSGI/uWSGI protocol for Python and other programming languages. (Any mention of the name uWSGI in this document will refer to the application server and not the protocol.) Being an application server, uWSGI usually runs behind a reverse proxy (e.g., Nginx) which load-balances between multiple instances of uWSGI, but it is functional enough to work as a standalone server for small-traffic websites. We use uWSGI to run and serve our Flask application.

Celery

Celery [90, 91] is a task queue allowing tasks to be performed asynchronously in the background, on a separate server. Tasks can also be scheduled to run at a specific time or in a specific interval, similar to a cron job [92]. We use Celery primarily as a job

scheduler [47] to run and control our scheduled tasks, such as sending reminder emails to users before the start of a new experiment.

Postfix

Postfix [93,94] is a simple SMTP server used as a mail transfer agent. It is very easy to setup and administer, making it a great alternative to the more popular Sendmail [95]. We use Postfix to deliver messages sent from our application, and configure it to be egress only.

Redis

Redis [96,97] is a very fast and advanced in-memory NoSQL [98] database. It is a key-value store (similar to a dictionary), supporting many different data types, such as strings, lists, sets, hashes, etc. Redis is used by Celery as a backend broker to store background tasks.

Bootstrap

Bootstrap [99,100] is a front-end framework for building responsive, interactive, mobile-first websites. It contains different HTML, CSS, and JavaScript utilities to aid in the quick development of beautiful and interactive websites. We use Bootstrap to build a responsive interface, able to run on multiple devices.

jQuery

jQuery [101,102] is a fast, feature-rich, cross-platform JavaScript library used to simplify many different JavaScript tasks, such as DOM traversal, event handling, and AJAX. It's the most popular JavaScript library in use today, accounting for a 93% market share [103]. We use jQuery to help make our interface more interactive and dynamic.

Supervisor

Supervisor [104, 105] is a supervision tool that allows processes to be monitored and controlled. Processes can be configured to be run and controlled by Supervisor, after which they can be restarted automatically if they fail to ensure they stay up and running. Supervisor has plenty of other features as well, such as a web server allowing remote control of processes. Other alternatives were considered, mainly Monit [106], but Supervisor was chosen for its ease of use and simple configuration.

Ansible

Ansible [107, 108] is a simple configuration management tool, used to automate the setup and configuration of systems. It allows control over many machines simultaneously, so complex systems can be deployed quickly and precisely. Ansible is used to automate the setup and deployment of our system, allowing quick and easy installation/updates. Other alternatives were considered, mainly Fabric [109], but Ansible was chosen for its large set of features.

4.3 Persistent Data Storage

4.3.1 Database Structure

The database structure implemented, seen in Listing 4.1, is slightly different from the structure in the design, seen in Listing 3.1. Some types were changed, mainly the use of *interval* instead of *integer* to hold durations. Some fields were added, such as *users.active*, mainly because some Flask extensions require them. Some fields were moved, such as *users.ip_address*, to make the application more flexible.

Listing 4.1: Designed using PostgreSQL 9.1.13

```
CREATE TABLE users (
    -- Autopopulated field storing time of creation
```

```

        created_at timestamp without time zone NOT NULL,
        id integer NOT NULL,
        email character varying(255) NOT NULL,
        password character(60) NOT NULL,
        host_last_updated_at timestamp without time zone,
        -- Fields added by Flask-Security
        active boolean NOT NULL,
        last_login_at timestamp without time zone,
        current_login_at timestamp without time zone,
        last_login_ip character varying(15),
        current_login_ip character varying(15),
        login_count smallint
    );

-- Separate table allows users to have multiple IP addresses (i.e
    ., multiple host machines)
CREATE TABLE user_ip_addresses (
    created_at timestamp without time zone NOT NULL,
    id integer NOT NULL,
    user_id integer NOT NULL,
    ip_address character varying(39) NOT NULL
);

-- Separate table allows users to have multiple MAC addresses (i.
    e., multiple host machines)
CREATE TABLE user_mac_addresses (
    created_at timestamp without time zone NOT NULL,
    id integer NOT NULL,
    user_id integer NOT NULL,
    mac_address character varying(17) NOT NULL
);

-- Read table containing roles users can have (e.g.,
    administrator)

```

```

CREATE TABLE roles (
    created_at timestamp without time zone NOT NULL,
    id integer NOT NULL,
    name character varying(80) NOT NULL,
    description character varying(255),
    permanent boolean NOT NULL
);

-- Join table allowing users to have multiple roles
CREATE TABLE users_roles (
    created_at timestamp without time zone NOT NULL,
    user_id integer NOT NULL,
    role_id integer NOT NULL
);

CREATE TABLE experiments (
    created_at timestamp without time zone NOT NULL,
    id integer NOT NULL,
    title character varying(140) NOT NULL,
    description text,
    -- Aggregate field holding total duration of configurations
    duration interval NOT NULL,
    start_time timestamp without time zone,
    -- Calculated field, based on start_time and duration
    end_time timestamp without time zone
);

CREATE TABLE configurations (
    created_at timestamp without time zone NOT NULL,
    id integer NOT NULL,
    previous_configuration_id integer,
    experiment_id integer NOT NULL,
    text text,
    duration interval NOT NULL

```

```
);
```

```
CREATE TABLE surveys (  
    created_at timestamp without time zone NOT NULL,  
    id integer NOT NULL,  
    experiment_id integer NOT NULL,  
    question character varying(140) NOT NULL,  
    description text  
);
```

```
CREATE TABLE options (  
    created_at timestamp without time zone NOT NULL,  
    id integer NOT NULL,  
    survey_id integer NOT NULL,  
    response character varying(140) NOT NULL  
);
```

```
CREATE TABLE user_choices (  
    created_at timestamp without time zone NOT NULL,  
    user_id integer NOT NULL,  
    survey_id integer NOT NULL,  
    option_id integer NOT NULL  
);
```

```
CREATE TABLE pages (  
    created_at timestamp without time zone NOT NULL,  
    id integer NOT NULL,  
    name character varying(80) NOT NULL,  
    title character varying(255) NOT NULL,  
    content text,  
    permanent boolean NOT NULL  
);
```

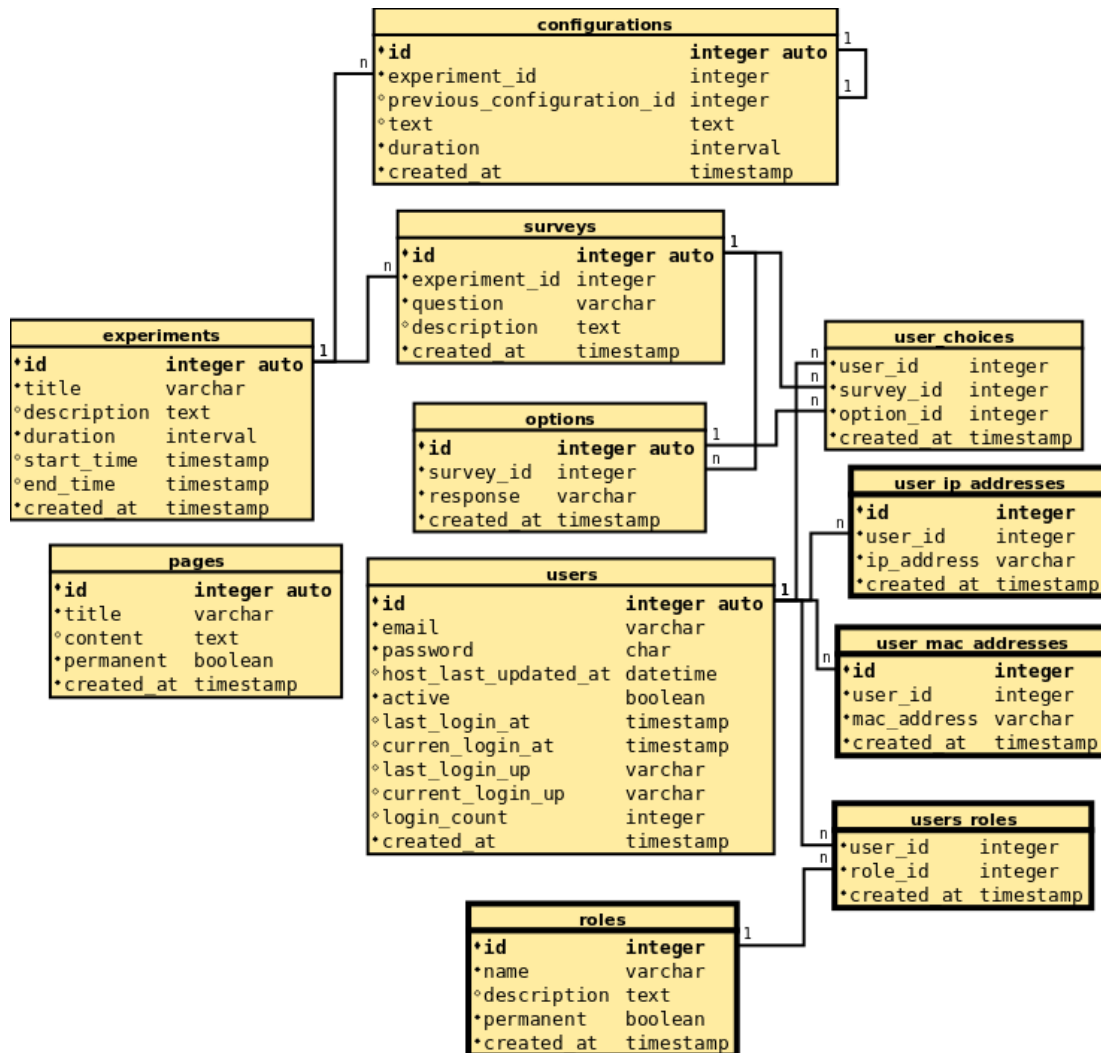


Figure 4.1: Relationship diagram of the implemented database.

4.4 Testing

Due to time constraints, the prototype was not tested at all levels. System testing was done to ensure that all requirements were satisfied. However, admittedly not **all** edge cases were tested. The following processes were tested manually.

Installing Click Admin

The installation process was tested on a virtual machine. Installation was done on a barebone machine, to ensure Click Admin was installed successfully, along with all of

it's dependencies. As per the requirements, the installation process proved to be quick and simple.

Creating an Experiment

The process of creating an experiment was tested manually. Multiple experiments were created, testing different edge cases, to ensure valid experiments were permitted and saved, and invalid experiments triggered the correct error messages. All requirements were satisfied.

Running an Experiment

Tcpdump log files were periodically checked and cross-referenced with actual usage to ensure packet data was being logged correctly. To ensure that experiments run as expected, a test experiment was scheduled. A configuration was added to the test experiment for the purpose of dropping all packets. As expected, all packets were dropped by the router while the experiment ran.

Chapter 5

Results

5.1 Installing Click Admin

Click Admin comes packaged with deployment scripts in the form of Ansible playbooks [107]. The first step to installing Click Admin is to configure it. Ansible expects to find a ‘vars.yaml’ file in the ‘deployment/secrets/’ directory with database credentials and other private keys.

Listing 5.1: deployment/secrets/vars.yaml

```
---  
  
# Database credentials  
db_name: clickadmin  
db_user: clickadminuser  
db_pass: clickadminpass  
  
# Used for signing cookies - should be set to something random  
secret_key: 0980423lkjfe093042kjhkg  
  
# Used for salting passwords - should be set to something random  
security_password_salt: 93802lkksmlvkenf089232n
```

Certain settings in ‘deployment/vars.yaml’ should also be changed. At the very least, the list of default administrators should be changed and/or added to.

Listing 5.2: deployment/vars.yaml


```
# Site administrators - accounts will be created for each
admins:
  - admin@clickadmin.com
```

After configuring Click Admin, run Ansible to install it. This will create an administrator account for each user listed, each whose password is *password* by default. Log on using this password and immediately change it on the **Change Password** page, found in the drop-down menu.

5.2 Creating an Experiment

After logging on as an administrator, head over to the **Create Experiment** page, found in the **Admin Section** of the site. Enter an experiment title and (optionally) a description, as seen in Figure 5.1. Submit the form to create the experiment.

Title *Is email a priority?*

Description *This experiment will aim to determine if email is a priority to our users.*

The screenshot shows a web browser window with the URL `33.33.33.33/admin/experiment/new/?url=%2Fadmin%2Fexperiment%2F`. The page has a navigation bar with links: Click Admin, Admin Home, Site Related, User Related, Experiment Related, Site Home, and Logout. The main content area has two tabs: List and Create. The Create tab is active, showing a form with the following fields:

- Title**: A text input field containing "Is email a priority?".
- Description**: A rich text editor with a toolbar (bold, italic, underline, link, unlink, list, ul, ol, table, table-bordered, table-striped, table-hover, table-condensed, table-responsive, table-dark, table-light, table-primary, table-secondary, table-success, table-danger, table-warning, table-info, table-dark, table-light, table-primary, table-secondary, table-success, table-danger, table-warning, table-info) and a text area containing "This experiment will aim to determine if email is a priority to our users.".
- Start Time**: A text input field.

At the bottom of the form are three buttons: **Submit** (blue), **Save and Add** (grey), and **Cancel** (grey).

Figure 5.1: Create experiment page.

After creating the experiment, head over to the **Create Survey** page. Select the experiment it belongs to in the drop-down list, enter a survey question and (optionally) a description, as seen in Figure 5.2. Submit the form to create the survey.

Experiment *Is email a priority?*

Question *Would you be willing to rate limit your email traffic?*

Description *Selecting yes will result in your emails being rate limited to 300 kB/s.*

Figure 5.2: Create survey page.

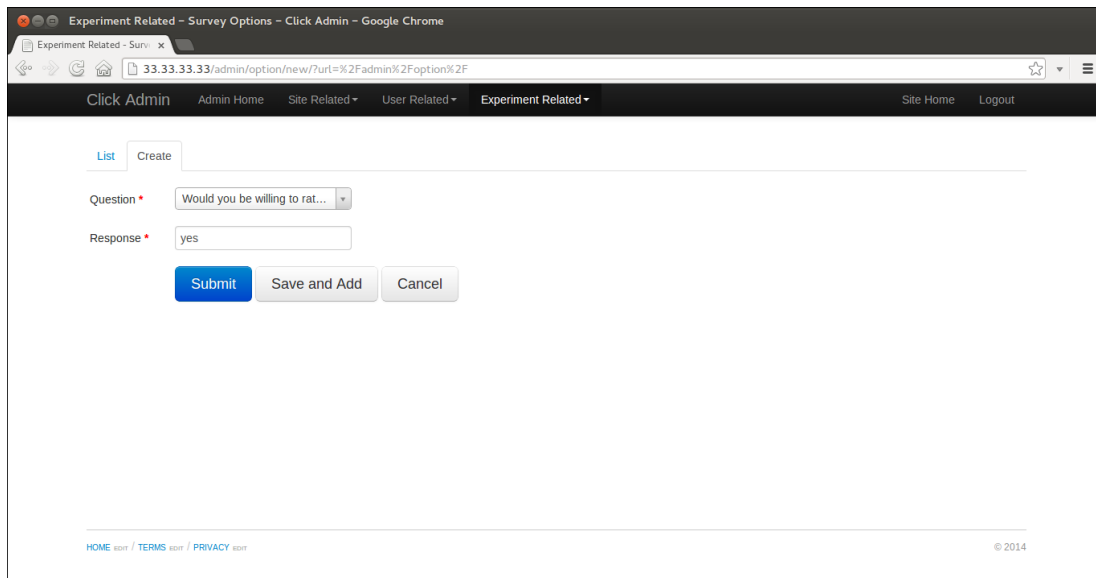
After creating the survey, head over to the **Create Survey Option** page. Select the survey it belongs to in the drop-down list and enter a response. Surveys must have at least two options, so click **Save and Add** to create the first survey option and repeat the steps to create another, as seen in Figure 5.3.

Survey *Would you be willing to rate limit your email traffic?*

Response (Option 1) *yes*

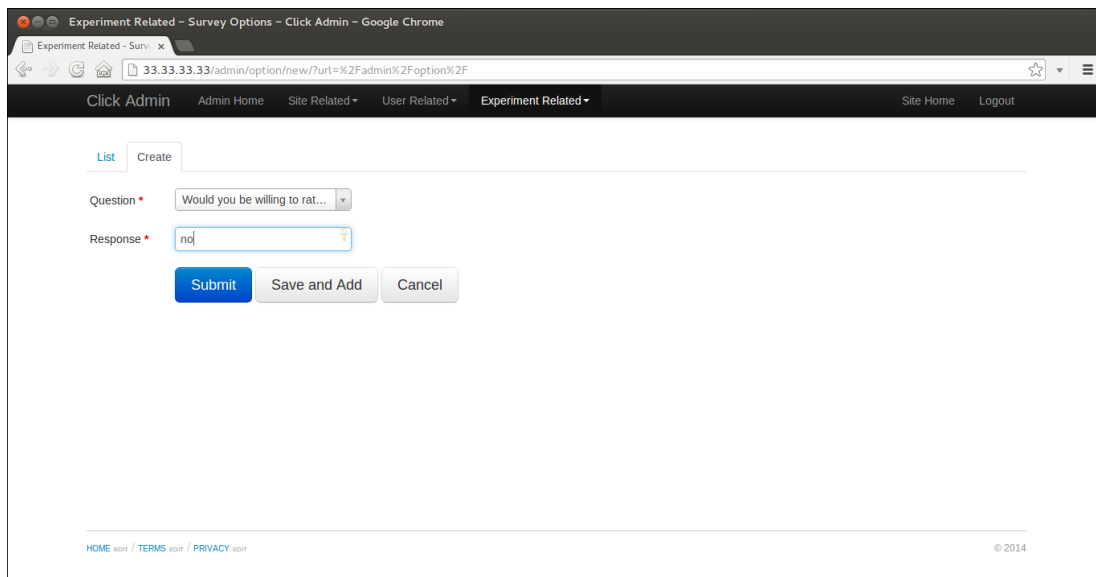
Response (Option 2) *no*

After creating the survey, head over to the **Create Configuration** page. Select the experiment it belongs to in the drop-down list, select the duration it should run for



The screenshot shows a web browser window with the title "Experiment Related - Survey Options - Click Admin - Google Chrome". The address bar shows the URL "33.33.33.33/admin/option/new?url=%2Fadmin%2Foption%2F". The page has a dark navigation bar with links: "Click Admin", "Admin Home", "Site Related", "User Related", "Experiment Related", "Site Home", and "Logout". Below the navigation bar, there are two tabs: "List" and "Create". The "Create" tab is active. The form contains a "Question" field with the text "Would you be willing to rat..." and a "Response" field with the text "yes". There are three buttons: "Submit", "Save and Add", and "Cancel". At the bottom of the page, there are links for "HOME", "TERMS", and "PRIVACY", and a copyright notice "© 2014".

(a) Option 1



The screenshot shows the same web browser window as in (a). The "Create" tab is active. The "Question" field still contains "Would you be willing to rat...". The "Response" field now contains the text "no". The "Submit", "Save and Add", and "Cancel" buttons are still present. The footer links and copyright notice remain the same.

(b) Option 2

Figure 5.3: Create survey option page.

in the drop-down list, and enter the content of the configuration, as seen in Figure 5.4.

Experiment *Is email a priority?*

Duration *a day*

Content Refer to Listing 5.3.

Figure 5.4: Create configuration page.

Listing 5.3: Configuration Template (Jinja2)

```

1 {% set ports = [110, 995, 143, 993, 25, 465, 587, 2525] %}
2 input :: FromDevice(eth0);
3 output :: ToDevice(eth0);
4 sched :: PrioSched;
5 bw :: BandwidthShaper(300000);
6
7 ip :: IPClassifier(
8     (
9         {% for user in option_1.consenting_users %}
10             host {{ user.ip_address }} {{ ' ' if loop.last else 'or' }}
11         }}

```

```

11     {% endfor %}
12 )
13 and
14 (
15     {% for port in ports %}
16         port {{ port }} {{ ' ' if loop.last else 'or' }}
17     {% endfor %}
18 ),
19 -
20 );
21
22 input -> ip[0] -> Queue -> bw -> [0] sched;
23 input -> ip[1] -> Queue -> [1] sched;
24 sched -> output;

```

5.3 Scheduling an Experiment

After creating the experiment with all of its surveys and configurations, the last step is scheduling a time for everything to run. As per the requirements, users are not made aware of experiments until they are scheduled. To do so, find the experiment in the **List Experiments** page and click edit to go to the **Edit Experiment** page. Enter the start time of the experiment, as seen in Figure 5.5, and submit the form to schedule the experiment.

Once the experiment is scheduled, users will be able to view the details of the experiment and answer any and all survey questions, as seen in Figure 5.6. As mentioned, a users response to a survey question may alter the rendered configuration. As it was written, our configuration will ensure that all users who responded *yes* to the question "*Would you be willing to rate limit your email traffic?*" will have all of their email packets (packets sent to/from standard email ports) rate limited to 300 kB/s, as seen in Listing 5.4.

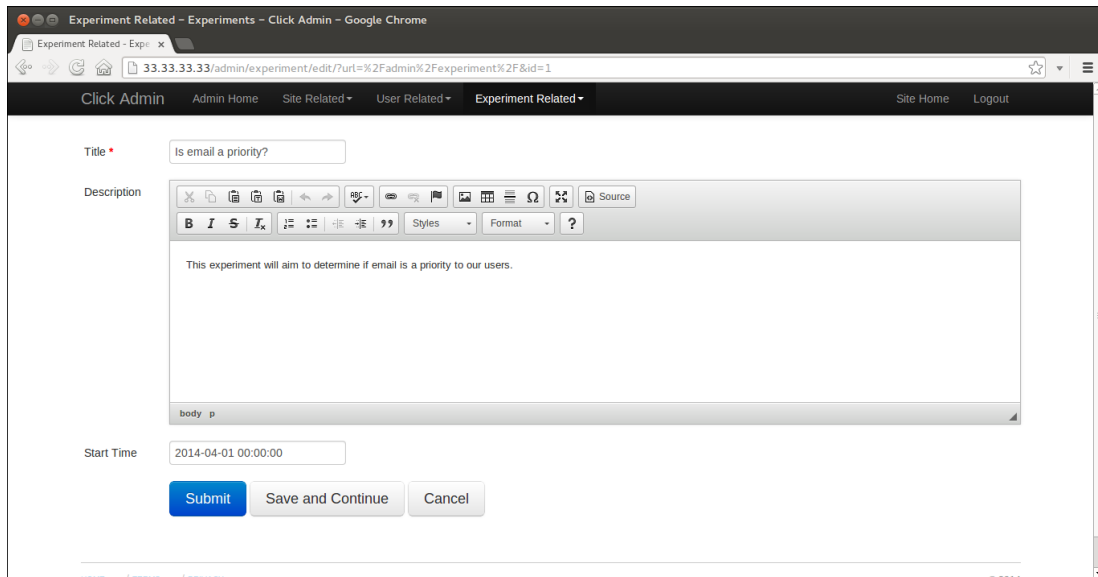


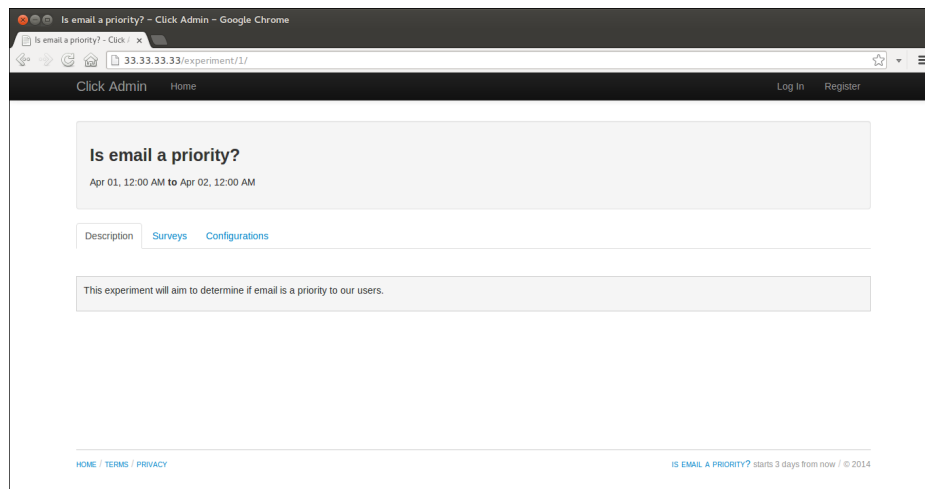
Figure 5.5: The edit experiment page has the same format as the create experiment page. However, a start time can only be set in this context.

Listing 5.4: Rendered Configuration

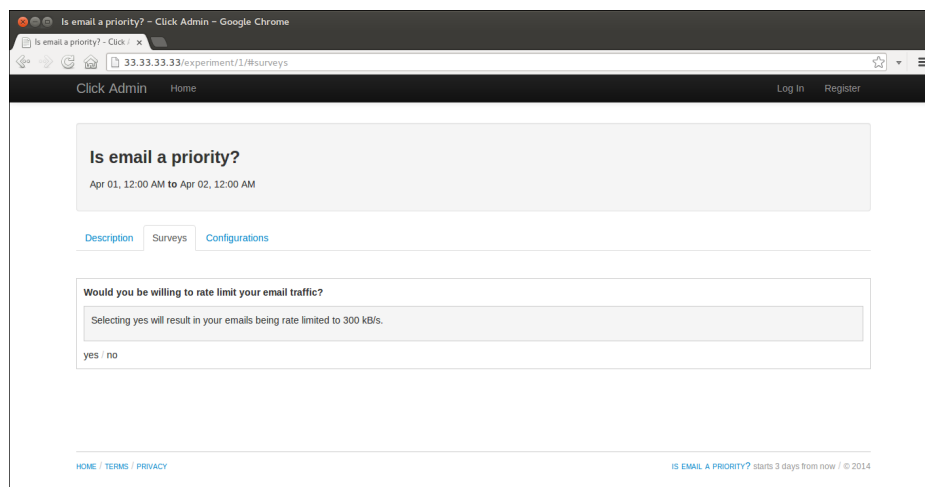
```

1 input :: FromDevice(eth0);
2 output :: ToDevice(eth0);
3 sched :: PrioSched;
4 bw :: BandwidthShaper(300000);
5
6 ip :: IPClassifier(
7     (
8
9         host 33.33.33.1
10
11     )
12     and
13     (
14
15         port 110 or
16
17         port 995 or

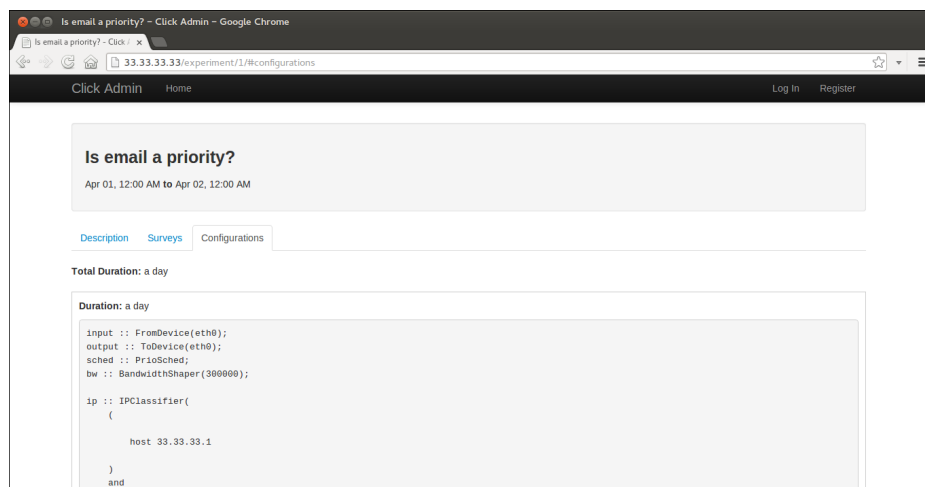
```



(a) Description



(b) Surveys



(c) Configurations

Figure 5.6: View experiment page. Users can view experiment details and answer survey questions.

```

18
19         port 143 or
20
21         port 993 or
22
23         port 25 or
24
25         port 465 or
26
27         port 587 or
28
29         port 2525
30
31     ),
32     -
33 );
34
35 input -> ip[0] -> Queue -> bw -> [0] sched;
36 input -> ip[1] -> Queue -> [1] sched;
37 sched -> output;

```

5.4 Running an Experiment

Once an experiment is scheduled, it will run all on it's own (assuming no errors are found). Our celery server runs a function every hour to check any upcoming experiments for errors. If any errors are found, a warning email is sent to each administrator to notify them. It's up to the administrator to address each and every error before the experiment runs. Our celery server also runs a function to check for any users who have yet to respond to the survey questions and reminds them to do so. If no errors are found, the experiment will start on schedule and each configuration will run for it's specified duration, one after the other. If the system is running on a gateway machine,

as is assumed, everyone on the network will be subjected to the configuration of the router.

Chapter 6

Conclusion and Future Work

Click Admin has the potential to be a valuable tool for network administrators. Although its intended use is for performing research into determining usage patterns in different scenarios, the extensibility of Click Admin should lead to some interesting use cases. One of the things Click Admin would greatly benefit from is more testing. As mentioned, only system testing was done. However, unit tests would provide much more code coverage, and should uncover any hidden bugs in the code. It would also help in testing different edge cases. There is an extension in Flask used to assist with writing and running tests called Flask-Testing [110].

An interesting feature that Click Admin could greatly benefit from is real-time analysis and feedback — allowing users to view how they and others are using the network, in real-time. This would provide instant feedback to users which would allow them to make instant changes and optimize their use of the network. For instance, they may choose not to use data intensive applications during peak hours.

Another interesting feature that Click Admin could benefit from is the ability to inform/remind network users of Click Admin's presence. As per the requirements, users must explicitly register to the system in order to participate and answer survey questions. However, everyone on the network is subjected to the configuration of the router. It would make sense for experiments to be advertised to all users on the network periodically.

References

- [1] A Survey of Network Traffic Monitoring and Analysis Tools. Available at http://www.cse.wustl.edu/~jain/cse567-06/ftp/net_traffic_monitors3.pdf.
- [2] Software Design Document — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Software_design_document.
- [3] Marshini Chetty, Richard Banks, A.J. Bernheim Brush, Jonathan Donner, and Rebecca E. Grinter. You're Capped! Understanding the Effects of Bandwidth Caps on Broadband Use in the Home. Available at http://marshini.net/uploads/2/0/6/8/20687796/capped_camera_ready.pdf, 2012.
- [4] Marshini Chetty, Richard Banks, Richard Harper, Tim Regan, Abigail Sellen, Christos Gkantsidis, Thomas Karagiannis, and Peter Key. Whos Hogging The Bandwidth?: The Consequences Of Revealing The Invisible In The Home. Available at http://marshini.net/uploads/2/0/6/8/20687796/home_watcher.pdf, 2010.
- [5] Eddie Kohler. *The Click Modular Router*. PhD thesis, Massachusetts Institute of Technology, February 2001. Available at <http://pdos.csail.mit.edu/papers/click:kohler-phd/thesis.pdf>.
- [6] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click Modular Router. Available at <http://pdos.csail.mit.edu/papers/click:tocs00/paper.pdf>, 2000.
- [7] Clicky GUI for Click. Available at <http://read.cs.ucla.edu/click/clicky>.
- [8] *Counter Element Documentation*. Available at <http://www.read.cs.ucla.edu/click/elements/counter>.
- [9] *IPClassifier Element Documentation*. Available at <http://www.read.cs.ucla.edu/click/elements/ipclassifier>.
- [10] *BandwidthShaper Element Documentation*. Available at <http://www.read.cs.ucla.edu/click/elements/bandwidthshaper>.
- [11] Click Element List. Available at <http://www.read.cs.ucla.edu/click/elements>.
- [12] Tcpdump — Wikipedia, the free encyclopedia. Available at <http://en.wikipedia.org/wiki/Tcpdump>.
- [13] *Manpage of TCPDUMP*. Available at http://www.tcpdump.org/tcpdump_man.html.

- [14] Tcpdump Source Code. Available at <https://github.com/the-tcpdump-group/tcpdump>.
- [15] A tcpdump Tutorial and Primer. Available at <http://www.danielmiessler.com/study/tcpdump/>.
- [16] Jeremy Stretch. Tcpdump Commands. Available at <http://packetlife.net/media/library/12/tcpdump.pdf>.
- [17] PCAP — Wikipedia, the free encyclopedia. Available at <http://en.wikipedia.org/wiki/Pcap>.
- [18] *Wireshark User's Guide*. Available at http://www.wireshark.org/docs/wsug_html_chunked.
- [19] Top-Down and Bottom-Up design — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Top-down_and_bottom-up_design.
- [20] U.S. Census Bureau. Computer and Internet Use in the United States. Available at <http://www.census.gov/prod/2013pubs/p20-569.pdf>, 2013.
- [21] Jamie Bartlett and Carl Miller. Truth, Lies and The Internet — A Report Into Young Peoples Digital Fluency. Available at http://www.demos.co.uk/files/Truth_-_web.pdf, 2011.
- [22] Eszter Hargittai, Lindsay Fullerton, Ericka Menchen-Trevino, and Kristin Yates Thomas. Trust Online: Young Adults Evaluation of Web Content. *International Journal of Communication*, 4:468–494, 2010. Available at <http://ijoc.org/index.php/ijoc/article/view/636/423>.
- [23] Marshini Chetty, David Haslem, Andrew Baird, Ugochi Ofoha, Bethany Sumner, and Rebecca E. Grinter. Why Is My Internet Slow?: Making Network Speeds Visible. Available at <http://www.marshini.net/uploads/2/0/6/8/20687796/kermi.pdf>, 2011.
- [24] Web Template System — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Web_template_system.
- [25] Host (network) — Wikipedia, the free encyclopedia. Available at [http://en.wikipedia.org/wiki/Host_\(network\)](http://en.wikipedia.org/wiki/Host_(network)).
- [26] Default Gateway — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Default_gateway.
- [27] Setting up an Ubuntu Router. Available at <https://help.ubuntu.com/community/Router>.
- [28] Functional Requirement — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Functional_requirements.
- [29] Non-Functional Requirement — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Non-Functional_Requirements.
- [30] FURPS — Wikipedia, the free encyclopedia. Available at <http://en.wikipedia.org/wiki/FURPS>.

- [31] Terms of Service — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Terms_of_service.
- [32] Privacy Policy — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Privacy_policy.
- [33] ISO/IEC/IEEE 42010 Systems and software engineering — Architecture description.
- [34] Request-Response — Wikipedia, the free encyclopedia. Available at <http://en.wikipedia.org/wiki/Request-response>.
- [35] Heather Kreger. Web Services Conceptual Architecture. Available at <http://cs.uoi.gr/~pvassil/courses/ptyxiakes/miscellaneous/WebServicesConceptualArchitecture.pdf>, 2001.
- [36] Shared Nothing Architecture — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Shared_nothing_architecture.
- [37] Load Balancing (computing) — Wikipedia, the free encyclopedia. Available at [http://en.wikipedia.org/wiki/Load_balancing_\(computing\)](http://en.wikipedia.org/wiki/Load_balancing_(computing)).
- [38] Channu Kambalyal. 3-Tier Architecture. Available at <http://channukambalyal.tripod.com/NTierArchitecture.pdf>.
- [39] Multitier Architecture — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Multitier_architecture.
- [40] Trygve Reenskaug. The Model-View-Controller (MVC) — Its Past and Present, August 2003. Available at http://heim.ifi.uio.no/~trygver/2003/javazone-jaoo/MVC_pattern.pdf.
- [41] Model-View-Controller — Wikipedia, the free encyclopedia. Available at <http://en.wikipedia.org/wiki/Model-view-controller>.
- [42] Packet Analyzer — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Packet_analyzer.
- [43] Application Server — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Application_server.
- [44] Database Server — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Database_server.
- [45] Web Server — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Web_server.
- [46] Mail Transfer Agent — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Mail_server.
- [47] Job Scheduler — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Job_scheduler.
- [48] *LibPCAP File Format*. Available at <http://wiki.wireshark.org/Development/LibpcapFileFormat>.

- [49] Linked List — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Linked_list.
- [50] Principles of User Interface Design — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Principles_of_user_interface_design.
- [51] Content Management System — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Content_management_system.
- [52] Software Testing — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Software_testing.
- [53] One-tier architecture. Available at <http://www.techopedia.com/definition/17374/one-tier-architecture>.
- [54] Linux — Wikipedia, the free encyclopedia. Available at <http://en.wikipedia.org/wiki/Linux>.
- [55] System Uptime Personal Record. Available at <http://blog.wsensors.com/2012/11/system-uptime-personal-record-4-75-years-and-growing-debian-linux/>.
- [56] Ubuntu. Available at <http://www.ubuntu.com/>.
- [57] Linux Distribution — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Linux_distribution.
- [58] Click Source Code. <https://github.com/kohler/click>.
- [59] *Python Documentation*. Available at <http://docs.python.org/2.7/>.
- [60] Python 2.7 Source Code. Available at <https://github.com/python/cpython/tree/2.7>.
- [61] Top 400 Python Projects on Github. Available at <http://pythonhackers.com/open-source/>.
- [62] *Flask Documentation*. Available at <http://flask.pocoo.org/docs/>.
- [63] Flask Source Code. Available at <https://github.com/mitsuhiko/flask>.
- [64] Kenneth Reitz. Why Django Sucks. <https://speakerdeck.com/kennethreitz/flasky-goodness>, September 2012.
- [65] *Flask-Admin Documentation*. Available at <https://flask-admin.readthedocs.org/en/latest/>.
- [66] Flask-Admin Source Code. Available at <https://github.com/mrjoes/flask-admin>.
- [67] *Flask-Mail Documentation*. Available at <https://pythonhosted.org/Flask-Mail/>.
- [68] Flask-Mail Source Code. Available at <https://github.com/mattupstate/flask-mail/>.
- [69] *Flask-Security Documentation*. Available at <https://pythonhosted.org/Flask-Security/>.

- [70] Flask-Security Source Code. Available at <https://github.com/mattupstate/flask-security>.
- [71] *Flask-SQLAlchemy Documentation*. Available at <http://pythonhosted.org/Flask-SQLAlchemy/>.
- [72] Flask-SQLAlchemy Source Code. Available at <https://github.com/mitsuhiko/flask-sqlalchemy>.
- [73] *SQLAlchemy Documentation*. Available at http://docs.sqlalchemy.org/en/rel_0_9/.
- [74] SQLAlchemy Source Code. Available at <https://github.com/zzzeek/sqlalchemy>.
- [75] Object-Relational Mapping — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Object-relational_mapping.
- [76] *Flask-WTF Documentation*. Available at <https://flask-wtf.readthedocs.org/en/latest/>.
- [77] Flask-WTF Source Code. Available at <https://github.com/lepture/flask-wtf/>.
- [78] *WTForms Documentation*. Available at <http://wtforms.readthedocs.org/en/latest/>.
- [79] WTForms Source Code. Available at <https://github.com/wtforms/wtforms>.
- [80] *Jinja2 Documentation*. Available at <http://jinja.pocoo.org/docs/>.
- [81] Jinja2 Source Code. Available at <https://github.com/mitsuhiko/jinja2>.
- [82] *PostgreSQL Documentation*. Available at <http://www.postgresql.org/docs/9.1/interactive/index.html>.
- [83] PostgreSQL Source Code. Available at <https://github.com/postgres/postgres>.
- [84] Object-Relational Database — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/Object-relational_database.
- [85] *Nginx Documentation*. Available at <http://wiki.nginx.org/Main>.
- [86] Nginx Source Code. Available at <https://github.com/nginx/nginx>.
- [87] C10k Problem — Wikipedia, the free encyclopedia. Available at http://en.wikipedia.org/wiki/C10k_problem.
- [88] *uWSGI Documentation*. Available at <http://uwsgi-docs.readthedocs.org/en/latest/index.html>.
- [89] uWSGI Source Code. Available at <https://github.com/unbit/uwsgi>.
- [90] *Celery Documentation*. Available at <http://celery.readthedocs.org/en/latest/index.html>.
- [91] Celery Source Code. <https://github.com/celery/celery>.

- [92] *Cron Linux Manual*. Available at <http://man7.org/linux/man-pages/man8/cron.8.html>.
- [93] *Postfix Manual*. Available at <http://www.postfix.org/start.html>.
- [94] Postfix Source Code. Available at <https://github.com/vdubrovni/postfix>.
- [95] Sendmail. Available at <http://www.qmailinfo.org/sendmail.html>.
- [96] *Redis Documentation*. Available at <http://redis.io/>.
- [97] Redis Source Code. Available at <https://github.com/antirez/redis>.
- [98] NoSQL — Wikipedia, the free encyclopedia. Available at <http://en.wikipedia.org/wiki/NoSQL>.
- [99] *Bootstrap Documentation*. Available at <http://getbootstrap.com/>.
- [100] Bootstrap Source Code. <https://github.com/twbs/bootstrap>.
- [101] *jQuery Documentation*. Available at <http://jquery.com/>.
- [102] jQuery Source Code. Available at <https://github.com/jquery/jquery>.
- [103] Usage Statistics and Market Share of JavaScript Libraries for Websites. Available at http://w3techs.com/technologies/overview/javascript_library/all.
- [104] *Supervisor Documentation*. Available at <http://supervisord.org/>.
- [105] Supervisor Source Code. Available at <https://github.com/Supervisor/supervisor>.
- [106] Monit — Wikipedia, the free encyclopedia. Available at <http://en.wikipedia.org/wiki/Monit>.
- [107] *Ansible Documentation*. Available at <http://www.ansible.com/>.
- [108] Ansible Source Code. Available at <https://github.com/ansible/ansible>.
- [109] *Fabric Documentation*. Available at <http://docs.fabfile.org/>.
- [110] *Flask-Testing Documentation*. Available at <https://pythonhosted.org/Flask-Testing/>.
- [111] Ivan Marsic. Software Engineering. Available at http://www.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf, 2012.
- [112] Katherine L. Milkman, Dolly Chugh, and Max H. Bazerman. How Can Decision Making Be Improved? Available at <http://www.hbs.edu/faculty/Publication%20Files/08-102.pdf>, 2008.
- [113] *Python Data Structures*. Available at <http://docs.python.org/2/tutorial/datastructures.html>.
- [114] Click Admin Source Code. Available at <https://bitbucket.org/BDuelz/click-admin>.
- [115] Middleware — Wikipedia, the free encyclopedia. Available at <http://en.wikipedia.org/wiki/Middleware>.