SERVICE STORE MODEL AND TOOLS FOR FRESCO

An Undergraduate Research Scholars Thesis

by

ARBIN BHUIYAN, MICHAEL CHACKO

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisor: Dr. Guofei Gu

May 2021

Major: Computer Science

RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

We, Arbin Bhuiyan and Michael Chacko, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Advisor prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

TABLE OF CONTENTS

ABSTRACT

Service Store Model and Tools for Fresco Applications

Arbin Bhuiyan, Michael Chacko
Department of Computer Science and Engineering
Texas A&M University

Research Faculty Advisor: Dr. Guofei Gu
Department of Computer Science and Engineering
Texas A&M University

As the number of network connected devices grows, the necessity to secure them and the networks they reside on increases in lockstep. In recent years, software defined networking (SDN) has grown from its infancy and has slowly established itself as a network security solution for commercial applications. To ease the burden of tedious packet level configurations via the SDN flow table, FRESCO was created by Texas A&M's SUCCESS Lab in collaboration with SRI International. FRESCO serves as a SDN scripting language that facilitates manipulation of the SDN control panel via modules that manipulate flow tables. To increase the adoption of SDN and FRESCO on the part of everyday consumers, we present the FRESCO Service Store, a centralized resource for community created FRESCO modules, applications, and tutorials. In addition to the service store, we will present the FRESCO AppBuilder, a GUI application which will allow the general user to create FRESCO Applications more easily. Finally, we present select FRESCO Applications that address key network security topics in addition to the ones that have already been created during the initial development of FRESCO in 2017.

DEDICATION

This work is dedicated to our friends and families who supported us during the COVID pandemic. We would also like to thank all frontline healthcare workers who risked their livelihood to combat this pandemic.

ACKNOWLEDGEMENTS

Contributors

We would like to thank Dr. Guofei Gu for his dedication to maintaining the SUCCESS Lab and fostering an environment of growth and learning.

Development for the FRESCO AppBuilder was aided through forking existing GitHub projects by Michał Ochman [1], and Jeremy Dombrowski [2]. Both repositories contain license files which give written permission for their code to be copied and modified for commercial use, private use, or distribution.

Peter Chacko offered significant help with Django and REACT, which aided in the development of the FRESCO Service Store and FRESCO AppBuilder.

Special thanks to Dr. Gu for allowing the use of figures from previous FRESCO publications (Figure 1.1.1 & Figure 1.1.2).

Special thanks to Taylor Vick for the use of his photography for the FRESCO Service Store through a free use license (Figure 3.1.1).

Funding Sources

Undergraduate research was supported by Dr. Guofei Gu and the SUCCESS Lab at Texas A&M University. There was no funding for this research.

NOMENCLATURE

SDN	Software Defined Network
FRESCO	Modular Composable Security Services for Software-Defined Networks
REACT	An open-source Javascript library maintained by Facebook.
OVS	Open vSwitch

1. INTRODUCTION

1.1 Background

1.1.1 Software Defined Networking

Introduced in 2008 [3], software defined networking is a nontraditional approach to networking that decouples the two planes involved in networking: the data plane, and the control plane. The data plane is the aspect of network most obviously felt by a user; its role is to carry network flow and ensure their delivery. The control plane manages the data plane and determines what traffic it will carry and how to route it within the network [4]. In traditional networking, these two planes are connected, which means that all networking devices such as switches and routers have their own data and control planes [5]. This means that in order to make a network security configuration, tedious packet level configurations must be made on the control plane of each router or switch in the network in order to implement the desired configurations.

However, with software defined networking, a control plane is not needed on every network device. Instead, a centralized software defined network controller serves this purpose, enforcing programmable configurations that sit one layer above the control plane in the application layer, on the data planes of each network device [6], as shown in Figure 1.1.1. This inherently removes much of the hassle in implementing network configurations, an unsavory bottleneck in traditional networks, and as a result, bolster an organizations network security as they can quickly make appropriate network changes to deal with potential and incoming threats.

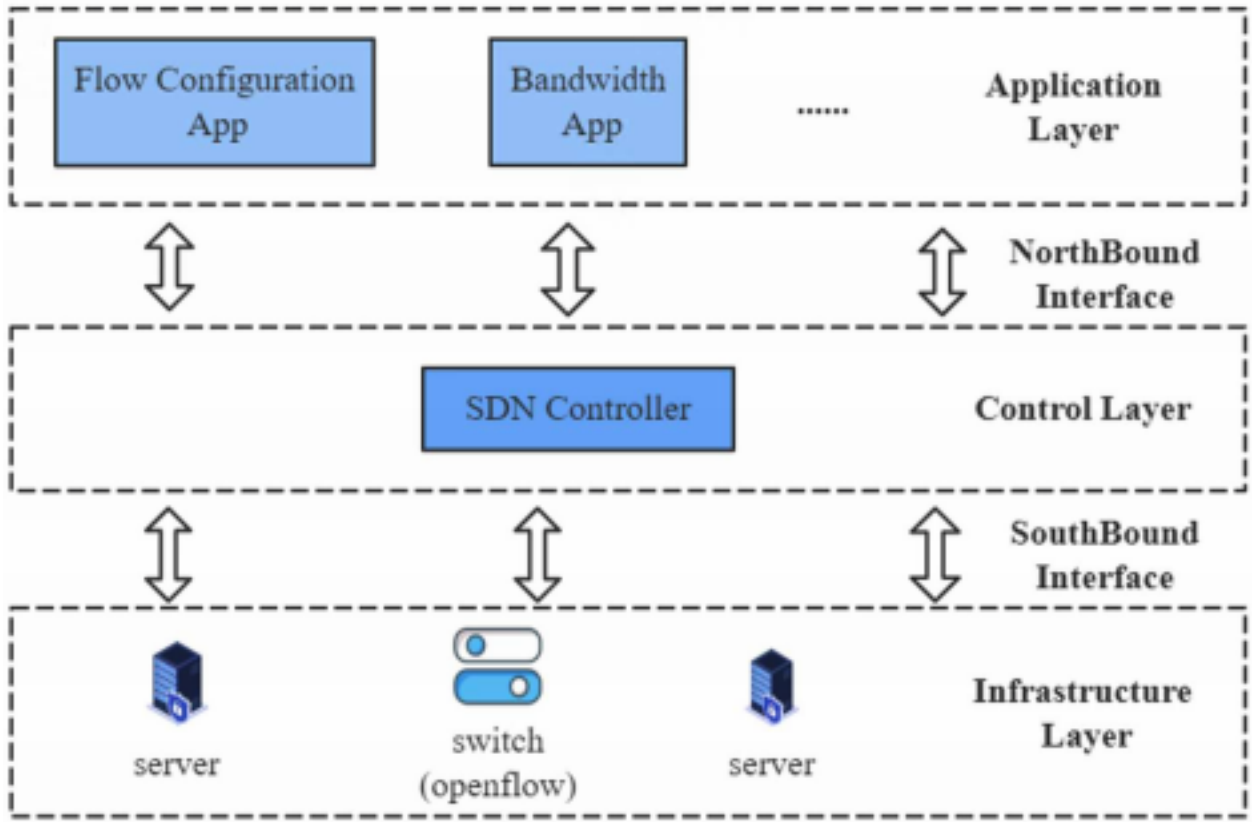

Figure 1.1.1 – Overarching layout of a software defined network [8]

1.1.2 FRESCO

While the basic methodology behind SDN network configurations is far better than its predecessor, the process can be further simplified. Although the central control plane eliminates the need to configure every switch or router on the network, the process of configuring the flow tables for the controller is still a somewhat arduous task. To easily manage the control plane, FRESCO was created by Texas A&M's SUCCESS Lab, led by Dr. Gu, in collaboration with SRI International. FRESCO is a SDN application development framework that delivers on the Open Networking Foundations goal to make software defined networks truly programmable by allowing users to manipulate pre-configured modules that achieve the function of flow rules without the labor [6]. FRESCO Modules are operative functions written in Java, and are the building blocks for FRESCO Applications, which are scripts written in JSON-like manner, using

said modules [8]. The lightweight nature of FRESCO Modules and FRESCO Applications give flexibility and portability to a wide variety of SDN security solutions, and they serve as an excellent tool for the open-source community to share and distribute their own custom solutions for SDN security. Figure 1.1.2 below illustrates the interactions between FRESCO modules to create FRESCO applications that run on a SDN Controller.

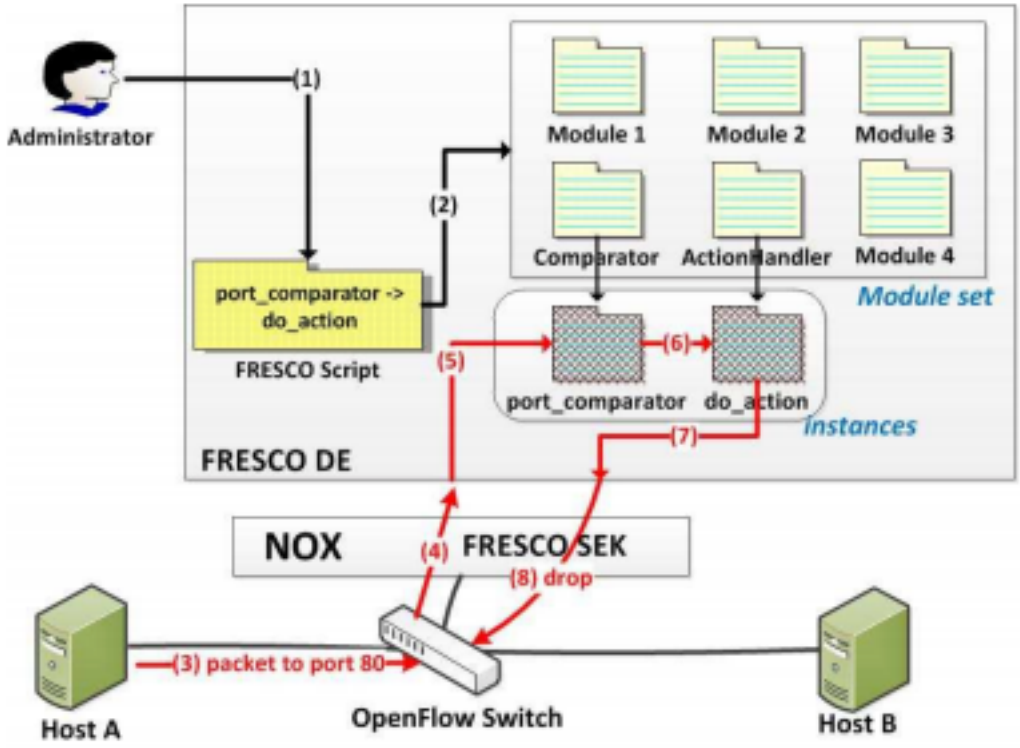

Figure 1.1.2 – Illustration of a SDN using FRESCO (NOX is a type of SDN controller) [8]

1.2 Motivation

As presented above, FRESCO is a powerful tool in democratizing the use of software defined networks. For consumer and commercial users, it is easy and free to implement in existing networks. For developers, its modular nature affords limitless opportunity to expand the module base and create new apps to fit one's needs. Combined with the ever-present need for

effective network security protocols in the world today, we believe that software defined networking and FRESCO is the most efficient coupling to start securing a network. As such, our research is motivated to increase the adoption of FRESCO on the part of users and developers via the following methods outlined in the sections below.

1.2.1 Method 1: FRESCO Service Store

Currently there is no central resource with the infrastructure for storing, hosting, and distributing FRESCO Applications. To serve this need, we have created a browser-based FRESCO Service Store that will serve as a central resource for users looking to utilize FRESCO. Via the service store, users can discover new FRESCO Apps that they wish to employ on their network. Developers can use the service store to distribute their FRESCO Apps after a short approval process to find and submit new modules for other developers to work with. The service store also hosts a variety of tutorials ranging from how to setup FRESCO on a network, to developer resources like how to write a FRESCO Application. The goal of this is to provide users and developers all the tools that they need in order to spread the reach of FRESCO and SDN.

1.2.2 Method 2: FRESCO AppBuilder

As FRESCO Applications are written in a JSON style language, apps with advanced functionality can become quite complex, with many interconnected modules that can be hard to manage. For this, we have created the FRESCO AppBuilder, a GUI based alternative for users looking to create more complicated FRESCO Applications. This tool can aid developers in their construction of FRESCO Apps by providing a GUI workflow for building new FRESCO Applications, making it a more attractive option for individuals looking for a networking solution. The FRESCO AppBuilder can also be used as a tool for beginners to dabble in the

development of FRESCO Applications. We believe that this is an important aspect, as not all network needs are the same, and the ability for all to create meaningful network configurations for their use case is essential to bolster overall network security.

1.2.3 Method 3: New FRESCO Modules & Applications

To make FRESCO an attractive option for prospective users, it is essential for there to be a strong catalog of FRESCO Applications ready to be used. While we hope that community development will grow the collection of Apps in the future, we have developed new general purpose FRESCO Modules and Applications that provide key functionality and address important security concerns, specifically concerning Denial-of-Service attacks. These FRESCO Applications will supplement the existing Applications made during the initial development of FRESCO to provide a solid set of solutions to common network security concerns.

2. METHODS

2.1 FRESCO Service Store

The FRESCO Service Store serves a multifunctional role in our project. It embodies a "home base" for the other two aspects of our work, while also serving as the main venue to grow the appeal of FRESCO. The sections below go over the reasoning behind the FRESCO Service Store model, as well key implementation details.

2.1.1 Inspiration and Philosophy

A service store model was chosen as the best avenue to create a central hub for all things related to FRESCO for a myriad of reasons. A primary inspiration that drew us to this approach was the overwhelming popularity of mobile based app stores like the Google Play Store and Apple App Store [9] . As a result of their success, an inherent connection has been made between users and the word "application". When people think of applications, they often think of a plug-and-play solution that provides a meaningful service to them [10]. This mold fits the definition of a FRESCO Application. Using this association to our benefit, we wanted to create an experience that made finding, downloading, and deploying FRESCO Applications as easy as possible. The same general idea has been employed by popular services like Slack and Microsoft Teams, in which a centralized spot allows users to discover and use third party applications and connections with the service in order to improve the functionality of the overall product.

Possible alternatives could have been to mimic popular open-source projects and create an open-source community on a platform like GitHub. However, although many developers are comfortable with GitHub and its equivalents, it is likely that the lack of clarity will push away potential users when trying to download specific applications or when navigating through dense

wikis for documentation and tutorials. As such, a service store model, with functionality for downloading and uploading FRESCO Applications and modules, GUI FRESCO AppBuilder support, and easy to navigate documentation and tutorials for developers and consumers, was decided on.

2.1.2 Functionality of the FRESCO Service Store

The FRESCO Service Store was developed with an HTML & CSS frontend, and a Django backend. The primary requirements for the FRESCO Service Store were to 1) authenticate user identity via logging in 2) allow users to submit their own custom modules or applications 3) allow moderators to approve, delete, and update user submissions 4) allow all features to be accessible through a web browser 5) allow users to search for applications. Development of the backend with Django allowed us to implement all these features.

2.1.2.1 User Log In

Users can log into the Service Store by clicking “Submit App” at the top of the page. From here, they will need to authenticate their identity with a GitHub Oauth login. We chose to use GitHub Oauth because it is a convenient log-in method for developers and allows Service Store administrators to avoid storing any sensitive user credentials.

2.1.2.2 User submission

Once a user has logged in, they can create a user submission by clicking “Submit App” at the top of the page. There, users must input the name, description, thumbnail, submission, and submission type. Once complete, the submission will be pending approval by a store administrator.

2.1.2.3 Moderator Features

All user submitted content must first be approved by an administrator before it can be seen publicly. The administrators can gain access to the administrator portal by logging in with a GitHub account that has been previously marked as an administrator or logging in with a special username & password pair.

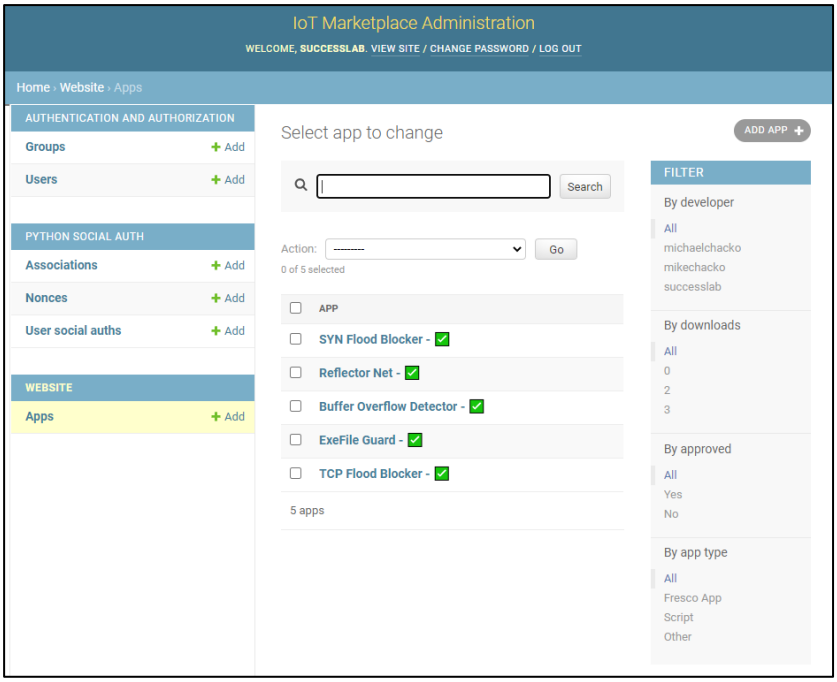

Figure 2.1.2.3 – The administrator portal for user and content moderation

User submissions can be sorted by approved/unapproved, and approved submissions are denoted with a green check mark as seen in Figure 2.1.2.3.

2.1.2.4 Web Browser

The entirety of the Service Store is displayed through the web browser. During the early stages of this project, we considered developing a Command Line Interface package manager (like “apt”, used in Debian based Linux). It was decided later that we should maintain a browser-based submission system to allow ease of maintenance and use for new users.

2.1.2.5 Submission Search

Users can quickly find existing submissions through the search bar in the top right corner of the Service Store. The search bar will search both the title and description of applications.

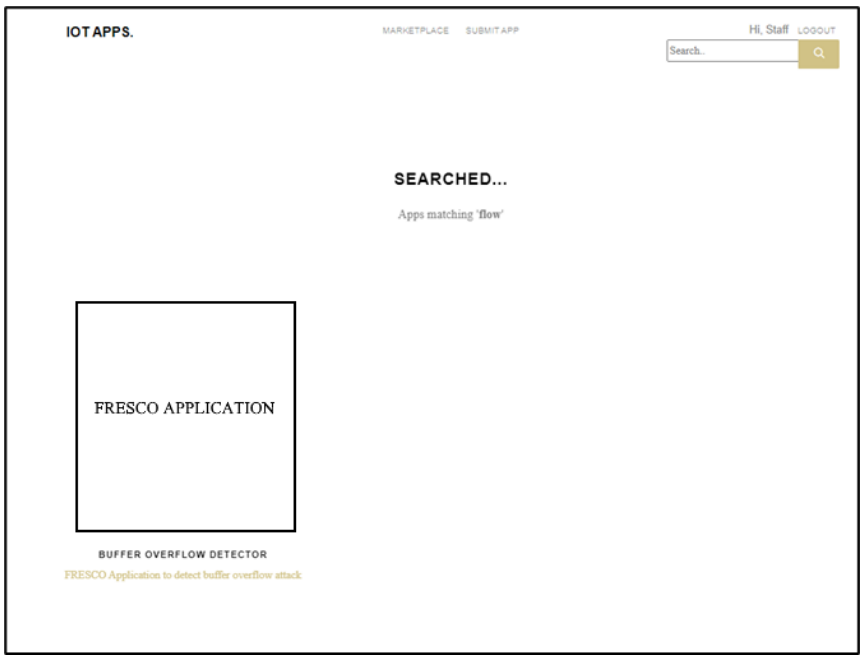

Figure 2.1.2.5 – User search for "Flow" in the Service Store. One result appeared.

As of May 2021, The FRESCO Service Store is hosted on Google Cloud and administrator access resides with Dr. Guofei Gu. The current plan for website hosting is to integrate the existing FRESCO TAMU subdomain to point to the FRESCO Service Store.

2.2 FRESCO AppBuilder

The purpose of the FRESCO AppBuilder is to simplify the process of creating large, complex FRESCO Applications, as well as providing a graphical medium for new developers to get their hands on and build FRESCO Applications. The following sections will explore the inspirations and implementations of this feature.

2.2.1 Inspiration and Philosophy

Although the FRESCO scripting language is far more preferable than manual configurations of the flow table, when developing complex applications, it is often cumbersome to keep up with the many interconnected inputs and outputs of the FRESCO modules. Compared to existing scripting languages, this requirement is cumbersome and difficult and may inhibit the growth of FRESCO by discouraging developers from creating complex applications. For this reason, we hypothesized a solution for this issue, with the primary goal being to create a system in which the relationships between the FRESCO modules that comprise a FRESCO Application could be easily represented.

Scratch is a visual programming language created by MIT Media Lab that is commonly used as an education tool to introduce people to program. It uses a block-based approach in which users manipulate blocks, each with their own utility and function, via their mouse and "scratch" them together in order to create programs [11]. For the purposes of our research, we recognized that Scratch is a perfect model for the problems we wanted to solve. Not only does the graphical approach clearly represent the relationships between the modules, including input and output, but it is also intuitive to use and generates the accompanying plain text code that the blocks represent. Furthermore, Scratch is a very beginner friendly tool that demystifies programming, enabling novices to learn quickly. As such, the FRESCO AppBuilder is heavily inspired by the function of Scratch in order to decrease complexity for developers when creating FRESCO Applications and provide a beginner friendly environment for novice programmers to develop their own applications.

2.2.2 Functionality of FRESCO AppBuilder

The FRESCO AppBuilder is a REACT application that runs on JavaScript in the browser. The primary requirements for FRESCO AppBuilder are to 1) allow users to visually drag and connect FRESCO Modules and 2) allow users to see a live update of the corresponding FRESCO Application JSON script which they can download. Both of these features are apparent in the main window of the application screen, where the modules that can be dragged are in the center, and the JSON script on the right pane. This JSON script can be downloaded as a FRESCO script file (.fre) for real world use. Users can also save and restore their FRESCO AppBuilder project by saving the full URL of their project.

2.2.2.1 FRESCO AppBuilder Development

Development of the FRESCO AppBuilder was done in React with the React Flow library [12]. Our work was forked from Michał Ochman React Web Audio Graph GitHub project [1] and Jeremy Dombrowski's React Flow Test GitHub project [2]. The FRESCO AppBuilder is hosted on GitHub and will be easily accessible from the FRESCO Service Store through a source pane.

2.2.2.2 FRESCO AppBuilder Layout

When opening the FRESCO AppBuilder, users can right-click to see the menu of accessible FRESCO Modules. Users will select the desired modules, fill their parameters, and connect them in the necessary logical fashion. The right-hand pane can be opened to view the corresponding FRESCO JSON output, as seen in Figure 2.2.2.2.

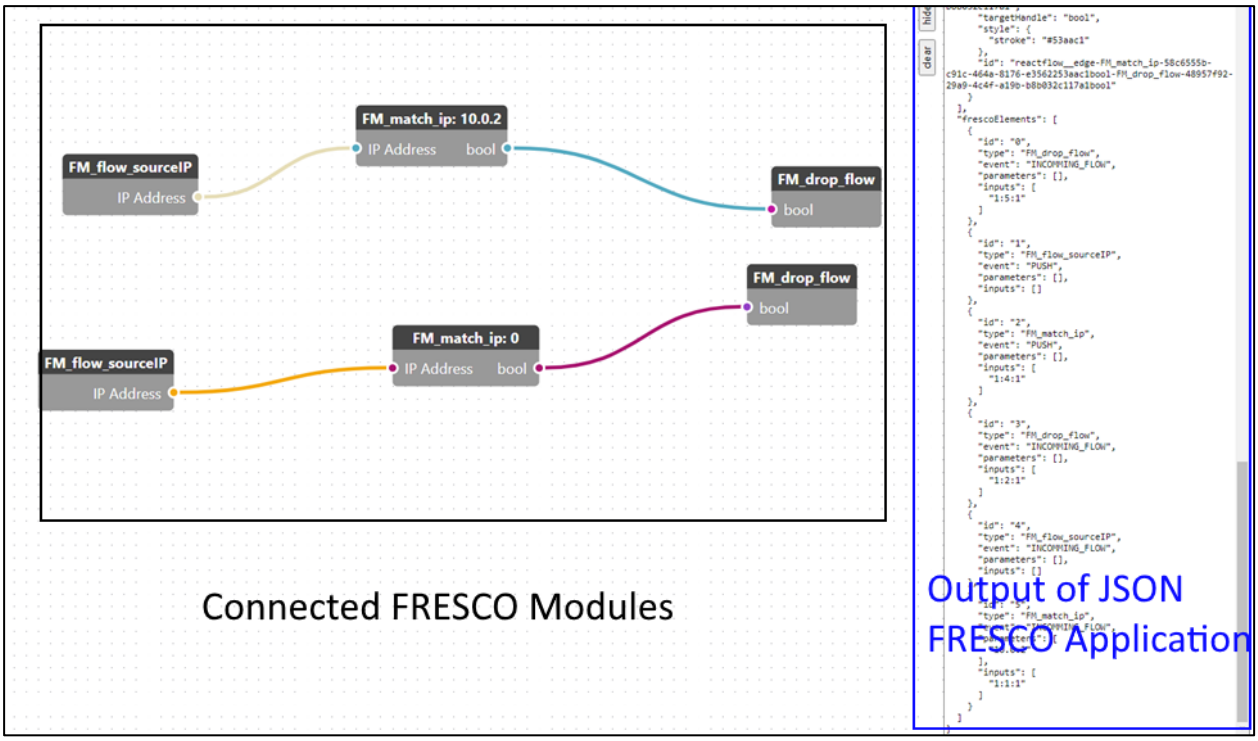

Figure 2.2.2.2 – Annotated screenshot of the FRESCO AppBuilder page.

2.3 New FRESCO Modules & Applications

In order to increase the usage of FRESCO, meaningful security applications that provide value to users need to be developed in order to attract adopters. During the initial development of FRESCO in 2017, nine exemplar FRESCO Applications were created to showcase the functionality of FRESCO and provide utility to adopters. Although all nine serve meaningful security purposes, we felt that we needed to expand upon them, and thus created two more applications, as well as a module, to supplement them and address more network security threats. The following sections explain the motivation behind the apps, as well as details on their implementation.

2.3.1 Inspiration and Philosophy

The most common type of network based attack is known as a Distributed Denial-of-Service Attack (DDoS), which is a "malicious attempt to disrupt the normal traffic of a targeted

server, service or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic" [13]. DDoS attacks can take many forms, but we chose to focus on the two most common variants. A "traditional" DDoS adds little variance from the definition and perpetuates the attack by flooding the target network with traffic from one or multiple IP addresses. As such, we have created the Dynamic Flood Blocker FRESCO Application that uses a whitelist-based approach to protect the network from this type of attack. Furthermore, as a result of the send/reply nature of network traffic, this application has the added benefit of intervening in cases in which the network itself may unknowingly be perpetuating a DDoS. There have been multiple instances over the past decade of IoT devices being undetected slaves in botnets that inflict DDoS attacks, the most notable being the Mirai botnet that infected hundreds of millions of devices, leveraging them as pawns for the attacks.

The second most common type of DDoS attack is known as a SYN Flood, or half-open attack. When establishing a TCP connection, a main protocol involved in networking, a client sends a SYN packet to a destination to initiate the connection. The destination replies to the client via SYN/ACK packet, acknowledging the connection, and keeps a port open for the client to reply with their own ACK that is used to acknowledge the acknowledgment. Once this complex dance is done, the TCP connection between the two is established, allowing for both to send and receive data from each other [14]. Attackers exploit this process in a SYN Flood attack by spoofing the IP's associated with a large amount of SYN packets directed at a target, which the target then replies to. Because the SYN packets were sent with spoofed IP's, the target is left with multiple open ports awaiting a response from a source that will never answer. Although these open ports do eventually time out without a response, the attacker bombards the target with so many requests that eventually, all available ports are tied up in the attack. This leaves the

target unable to respond to credible network connections, rendering it useless in performing it given function. To prevent this type of DDoS attack, we have created the SYN Flood Attack Mitigation FRESCO Application. This app takes a two-pronged approach to combat this intrusion by sensing when a potential SYN Flood attack is occurring and enacting specific countermeasures to lessen the load on the network. In the case of extreme attacks, the app is equipped with an ability to temporarily limit all traffic to the network, allowing time for the network to open enough ports and recover.

2.3.2 Functionality

2.3.2.1 Dynamic Flood Blocker

To create the Dynamic Flood Blocker application, in addition to using the existing modules in FRESCO, we created our own module that serves as a Boolean negation function. Using this module, we simply used Boolean logic via a sequence of OR's and NOT's to determine if the source/destination IP is directed to a whitelisted IP. If it is, the traffic is not altered. However, if it is not, and the application determines that an abnormal amount of traffic has been received or sent to a particular IP, subsequent flow to or from that location will be dropped until the traffic normalizes. Specifically, the application keeps track of how many times a certain IP has been interacted with on the network over the course of a minute. If the IP address is whitelisted, it does not matter, but if it is not and exceeds the allowed amount of interactions within a minute, the flow is dropped.

To enable the whitelist mechanic, the application is accompanied by a Python script we have created that is used to generate the specific whitelist a user wants. The user enters the IP addresses they wish to whitelist into the script, either as program arguments, or through a text file, as well as the allowed number of interactions for non-whitelisted IP addresses. The script

then goes through and creates the specific application the user should install onto their FRESCO network. Without this Python script, the user would have to go through a tedious process of modifying the application to fit their whitelist, so instead, we created this script to make this application as easy as possible to use.

2.3.2.2 SYN Flood Attack Mitigation

Because SYN Flood attacks are often perpetuated using spoofed IP's, the approach used in the Dynamic Flood Blocker will not work as the traffic will be masqueraded from multiple sources. The SYN Flood Attack Mitigation app uses two user provided "sensitivities", similar to the previous app. If a SYN Flood attacks is suspected via triggering the first sensitivity, the application will enable countermeasures to modify the sysctl.conf to prevent IP Spoofing, reduce the amount of SYN_ACK retires, and decrease SYN timeout, in attempt to lessen the load on the network. These countermeasures are built into the kernels of most networking devices, and are effective in lessening the load on the network by filtering out legitimate and illegitimate requests [14]. However, they come at a performance cost if enabled [15]. By dynamically triggering these countermeasures when needed, the network does not suffer from performance issues when a half-open attack is not in progress, but is able to fight-back and mitigate the effects of the attack if needed. In the case of an egregious attack, triggered by the second sensitivity, the application will temporarily limit all traffic, just like the Dynamic Flood Blocker.

3. RESULTS

3.1 FRESCO Service Store Results

Using the methodology laid out in Section 2.1, the FRESCO Service Store Website resulted in the layout shown in Figure 3.1.1. On the home page, the top ten most popular uploads are displayed for quick access, as shown in Figure 3.1.2. Users can find and search for FRESCO Applications or Modules in the search pane. Additionally, they can log in with their GitHub account to submit new apps for approval, as shown in Figure 3.1.3. The administrator portal shown in Figure 2.1.2.3 gives administrators complete control over all website content.

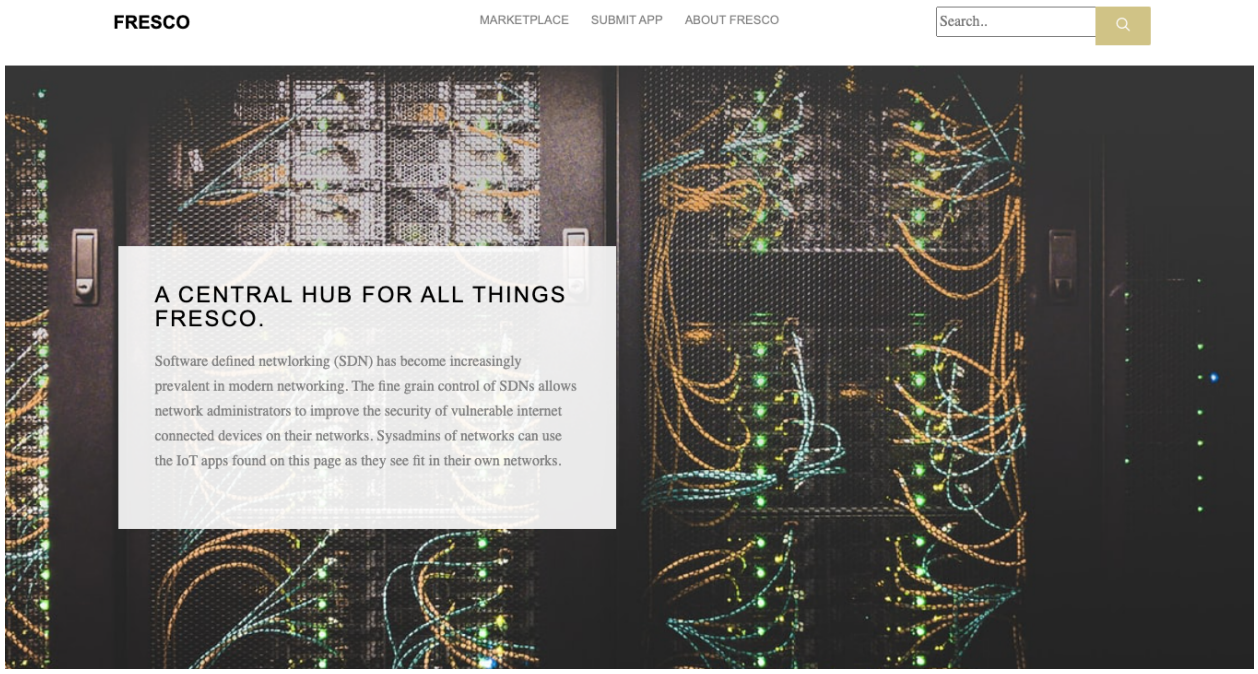

Figure 3.1.1 – FRESCO Service Store Homepage

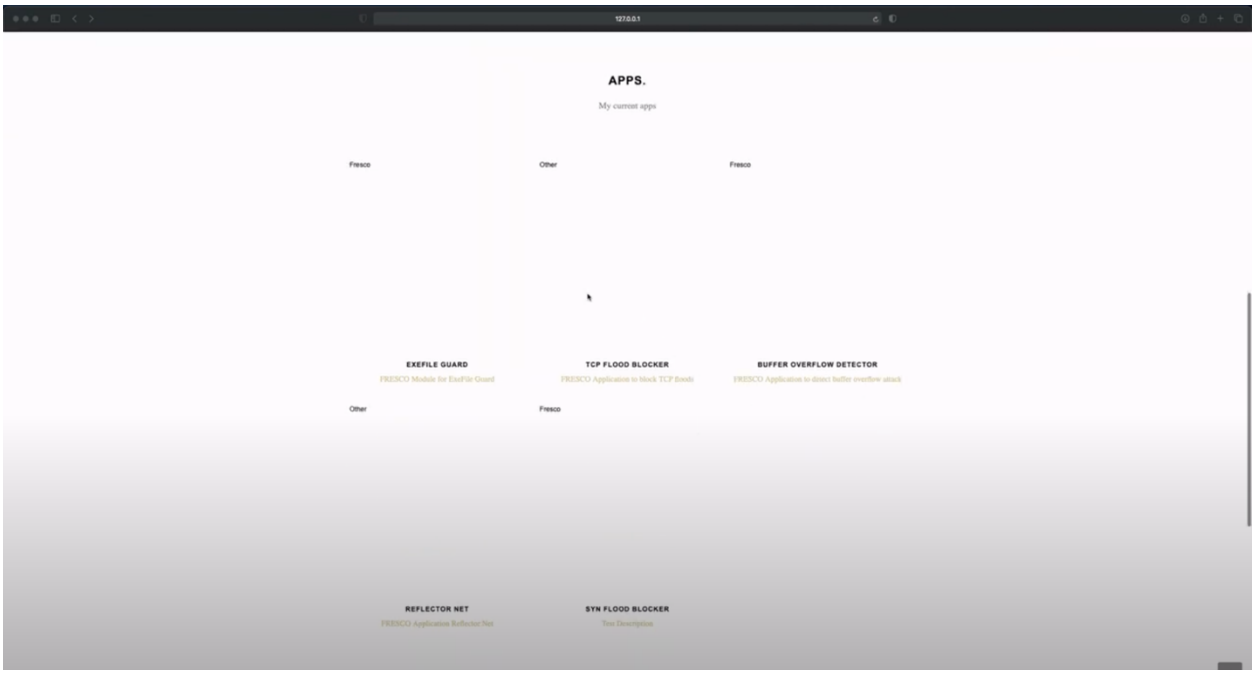

Figure 3.1.2 – Page for users to download FRESCO Applications

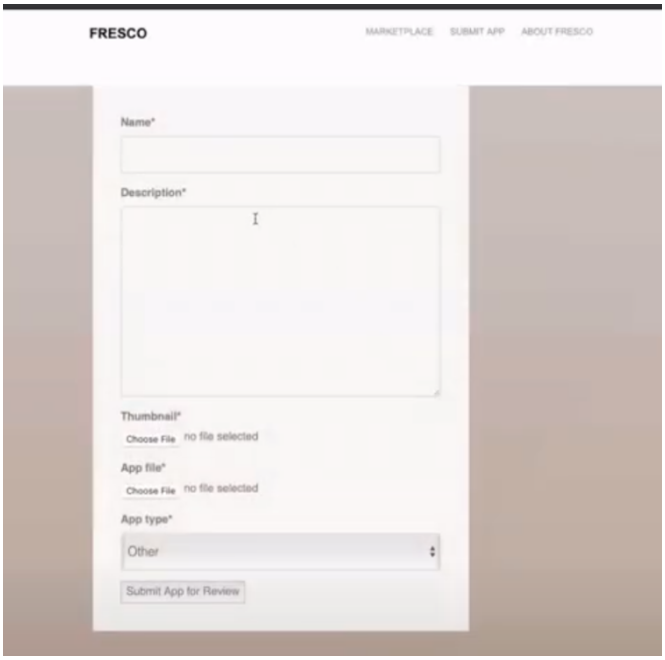

Figure 3.1.3 – Page for users to submit their own FRESCO Application for approval

Administrators are also sent an e-mail whenever a new submission has been made. Figure 3.1.4 shows the real approval request sent to the website administrator after a new user submission.

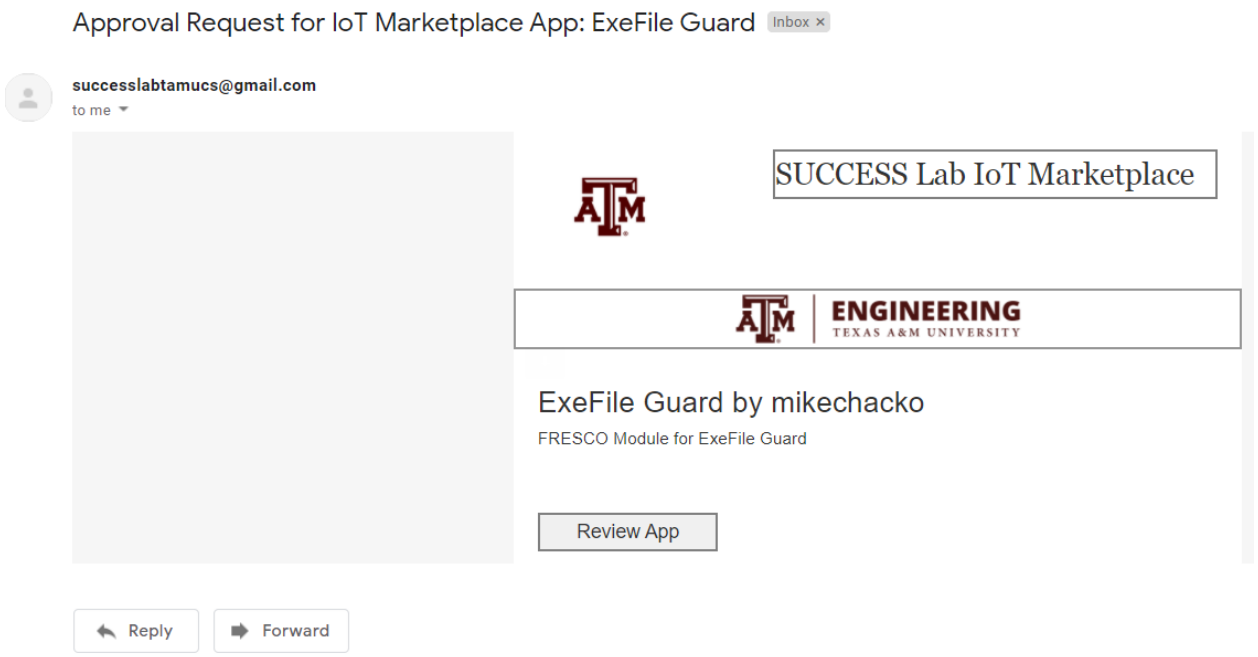

Figure 3.1.4 – Approval Request sent to administrators when an upload is submitted

The FRESCO Service Store also hosts tutorials as resource for users to quickly jumpstart their FRESCO projects. Figure 3.1.5 shows the "About FRESCO" page which will host links to instructional videos that provide users with knowledge on using the FRESCO Service Store, FRESCO AppBuilder, and other developer resources.

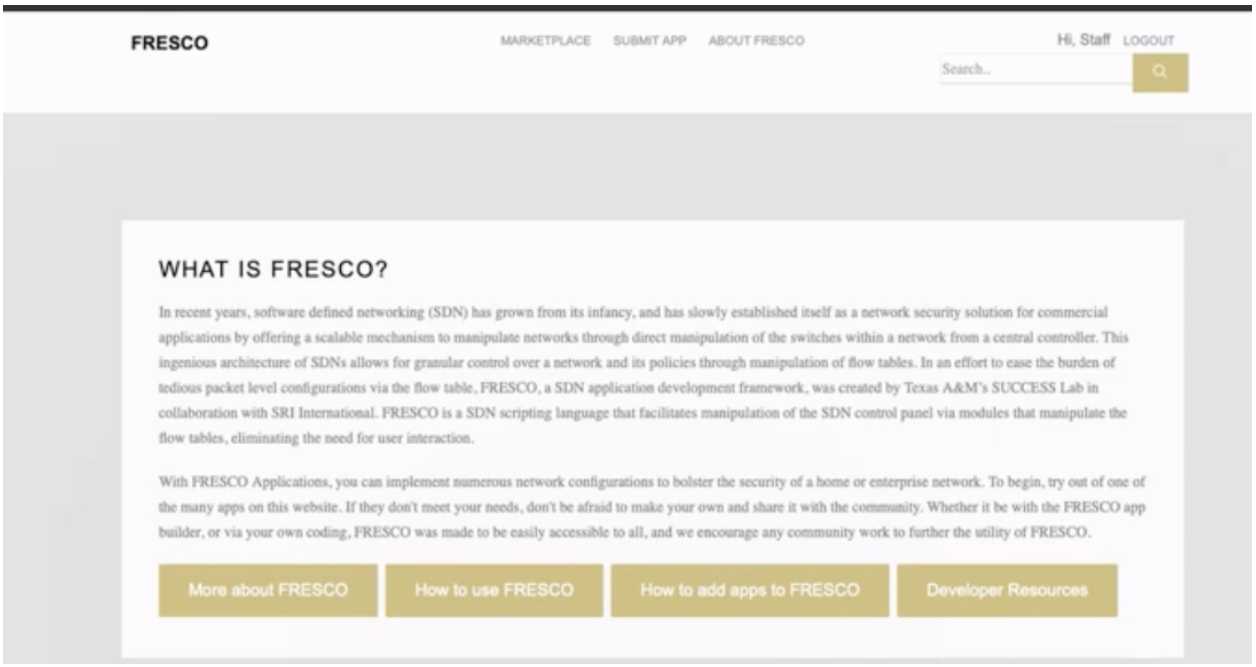

Figure 3.1.5 – FRESCO Service Store Tutorial Page

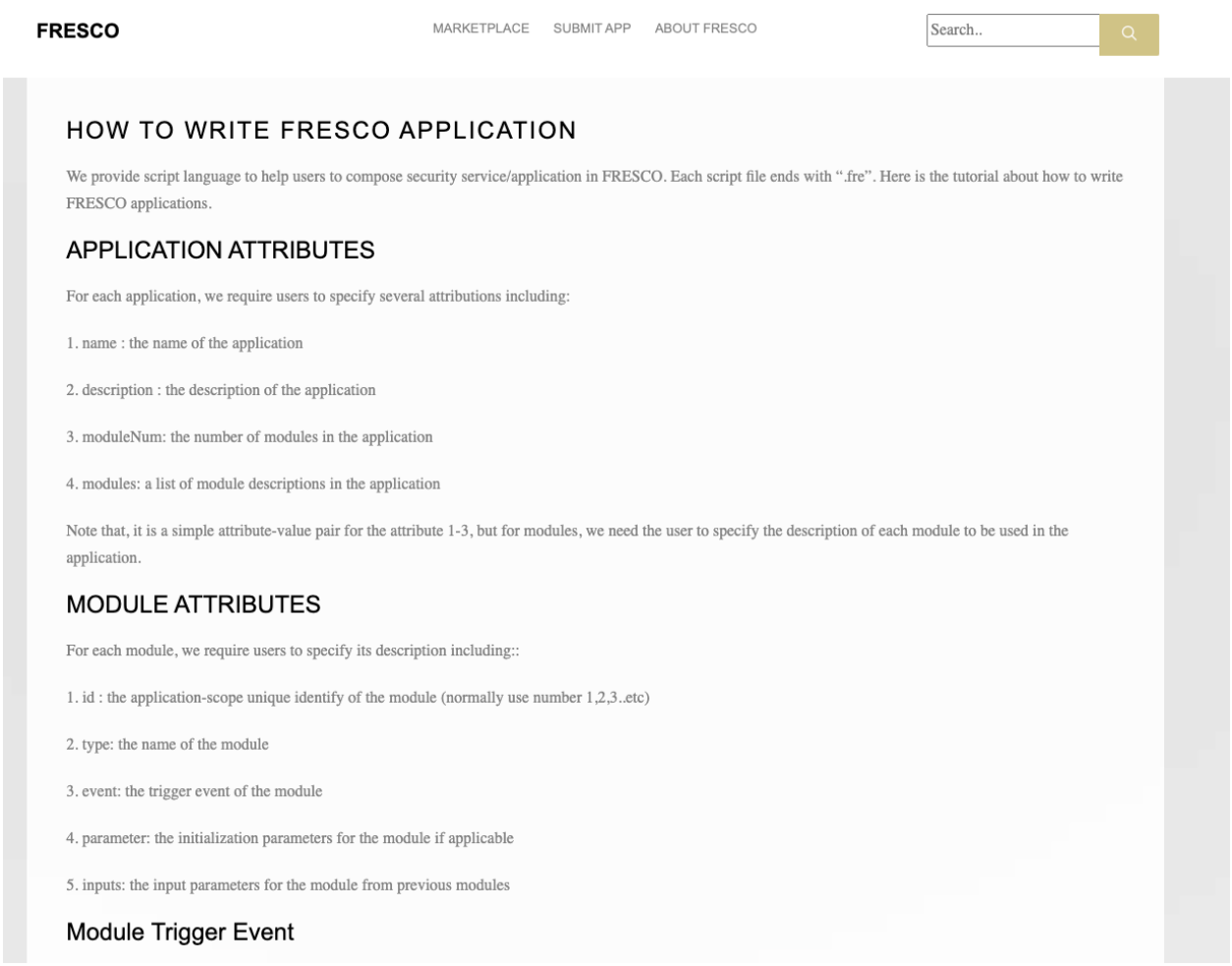

Figure 3.1.6 – FRESCO Developer Resources: How to Write a FRESCO App

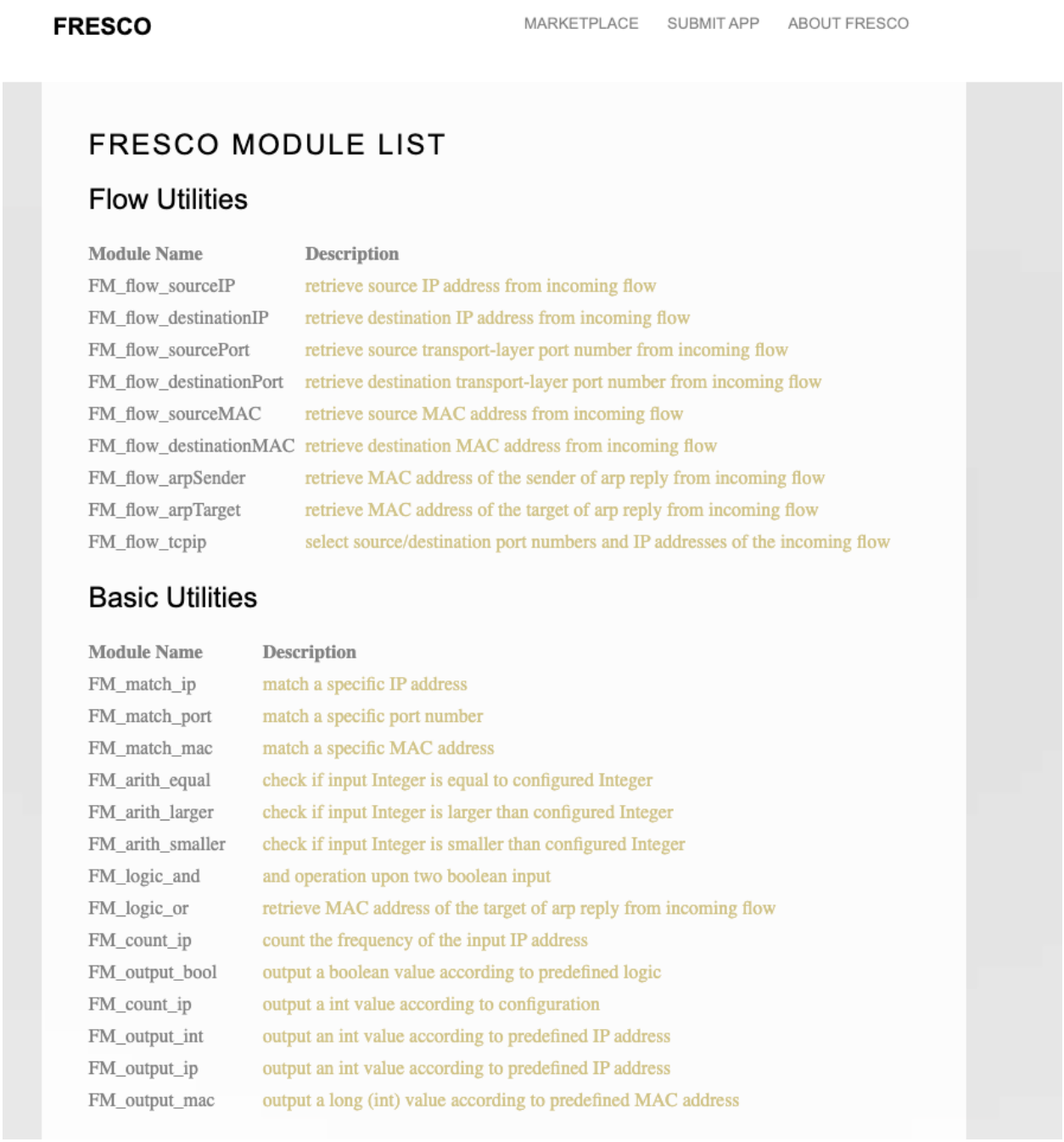

Figure 3.1.6 – FRESCO Developer Resources: Module List

3.2 FRESCO AppBuilder Results

Showing the functionality laid out in Section 2.2, the FRESCO AppBuilder resulted in the layout shown in Figure 3.2.1. Users can drag and connect FRESCO modules to generate the corresponding FRESCO JSON script on the right-hand pane. The download button shown on the

right-hand pane will download the JSON script to a *.fre* file that can be used in FRESCO projects. Users can also create a custom node as a placeholder for any new FRESCO modules they wish to use or create. All these features contribute to a robust web application that encourages the easier creation of new FRESCO Applications.

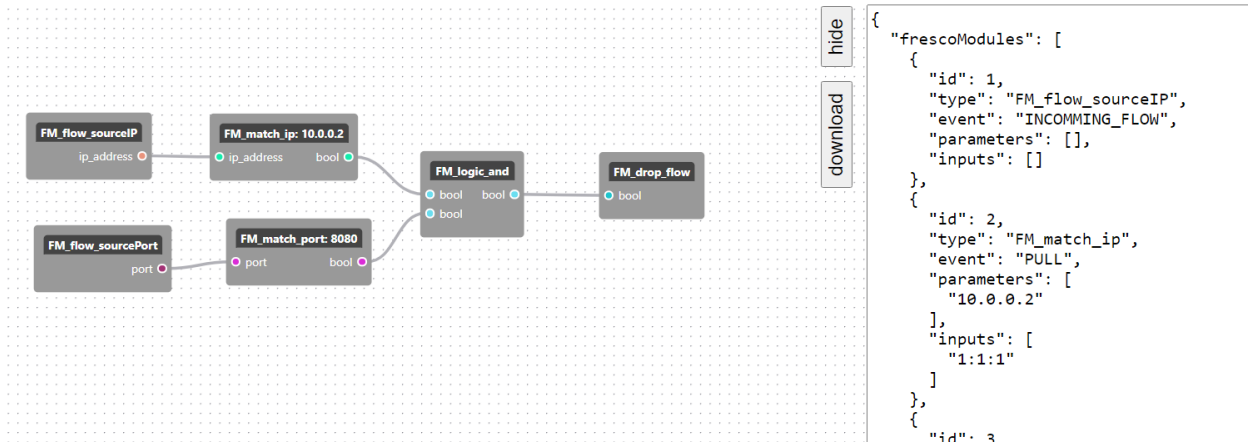

Figure 3.2.1 – FRESCO AppBuilder with an example application

Additionally, users can import and export their FRESCO AppBuilder projects by saving the full URL of the project, as shown in Figure 3.2.2. The URL encodes all information about the project so that users can regain access to their projects through a hyperlink.

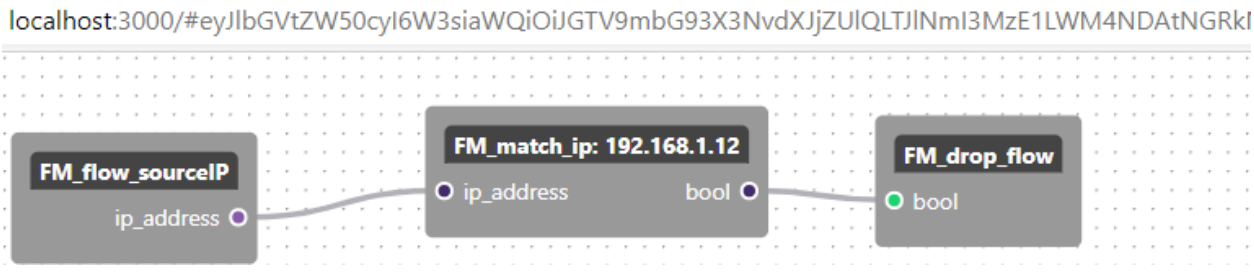

Figure 3.2.2 – FRESCO AppBuilder encodes the project state in the URL

When creating new FRESCO Applications, the interface provided by the FRESCO AppBuilder is extremely intuitive when compared to manually scripting new Applications. All of

the features listed in this section contribute to an experience that encourages ease-of-use for the user.

3.3 FRESCO Modules & Applications Results

As our research focused on increasing the adoption of FRESCO, we focused primarily on the previous two aspects of our work. While we were not able to fully evaluate the efficacy of the two new applications created, we have however verified that they indeed work, and provide the function promised. In the regards to the Dynamic Flood Blocker, we have verified that the included Python script creates a FRESCO application with the correct whitelist and confirmed using a virtual network via Mininet that the application does indeed block excessive traffic from non-whitelisted sources. In the same vein, using Mininet, we have confirmed that under network condition similar to the those of a SYN Flood attack, the SYN Flood Attack Mitigation Application does indeed enable the proper kernel countermeasures and network fail safes as detailed in 2.3.2.2.

3.4 Future work

To further improve upon the FRESCO Service Store, new content needs to be added by the community to increase the number of useful plugins available. Next, the website itself can benefit from a visual overhaul that is more appealing and space efficient. The FRESCO AppBuilder can be improved through increased features, including a toolbar in the top for more customization options, color coded connection types between nodes, and tighter integration with the FRESCO Service Store for direct uploads.

4. CONCLUSION

As our society becomes more dependent on the security of network-connected devices, we hope that robust security solutions keep pace to combat any malicious actors. To best accomplish this goal, we have created three key tools introduced in this paper: the FRESCO Service Store, the FRESCO AppBuilder, and new FRESCO applications to fight DDoS attacks. We are very excited to release these tools due to their ability to improve the adoption of FRESCO, which has the potential to become a mainstream resource in the world of network security.

The FRESCO Service Store will serve as a convenient hub for developers to find and share new FRESCO Applications. Tutorials and other resources will also be available for developers, making the FRESCO Service Store the go-to resource for developers looking to utilize FRESCO in their work.

With community applications readily available through the FRESCO Service Store, developers can also create their own Applications with the FRESCO AppBuilder. The GUI method of building FRESCO Applications will be extremely beneficial to both novice and advanced developers, as both simple and complicated Applications can be visualized and modified much more intuitively when compared to scripting Applications.

Finally, the release of new FRESCO Applications to protect against DDoS attacks will give users even more tools to protect their own networks, while also inspiring developers with new ideas for their own potential FRESCO Applications. Together, all these tools will contribute to a safer interconnected digital world.

REFERENCES

[1] M. Ochman, *React Web Audio Graph*. https://github.com/michalochman/react-web-audio-graph (accessed Apr. 01, 2021).

[2] J. Dombrowski, *React Flow Test*. https://github.com/meatflavourdev/react-flow-test (accessed Apr. 01, 2021).

[3] N. McKeown *et al.*, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008, doi: 10.1145/1355734.1355746.

[4] S. Sezer *et al.*, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013, doi: 10.1109/MCOM.2013.6553676.

[5] "SDN Versus Traditional Networking Explained," Feb. 11, 2021. https://www.ibm.com/services/network/sdn-versus-traditional-networking (accessed Mar. 01, 2021).

[6] "Software-Defined Networking (SDN) Definition," *Open Networking Foundation*. https://opennetworking.org/sdn-definition/ (accessed Mar. 01, 2021).

[7] Y. Gao, Y. Chen, X. Hu, H. Lin, Y. Liu, and L. Nie, "Blockchain based IIoT data sharing framework for SDN-enabled Pervasive Edge Computing," *IEEE Trans. Ind. Inform.*, pp. 1–1, 2020, doi: 10.1109/TII.2020.3012508.

[8] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular Composable Security Services for Software-Defined Networks," p. 16.

[9] "App Download and Usage Statistics (2020)," *Business of Apps*, Sep. 01, 2017. https://www.businessofapps.com/data/app-statistics/ (accessed Mar. 01, 2021).

[10] M. Dakić, “What Makes a Great App Great,” *Medium*, Dec. 01, 2020. https://medium.datadriveninvestor.com/what-makes-a-great-app-great-6e15d6c36b26 (accessed Mar. 01, 2021).

[11] “Scratch - About.” https://scratch.mit.edu/ (accessed Mar. 01, 2021).

[12] “webkid - React Flow.” https://webkid.io/projects/react-flow/ (accessed Mar. 01, 2021).

[13] “What Is a Distributed Denial-of-Service (DDoS) Attack?,” *Cloudflare*. https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/ (accessed Mar. 01, 2021).

[14] “SYN Flood DDoS Attack,” *Cloudflare*. https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/ (accessed Mar. 01, 2021).

[15] B. Al-Duwairi, E. Al-Quraan, and Y. AbdelQader, “ISDSDN: Mitigating SYN Flood Attacks in Software Defined Networks,” *J. Netw. Syst. Manag.*, vol. 28, no. 4, pp. 1366–1390, Oct. 2020, doi: 10.1007/s10922-020-09540-1.

[16] J. Kurose and K. Ross, “Computer Networking: A Top Down Approach,” 7th ed., Pearson/Addison Wesley, 2016.