Trace Driven Traffic Generator for Self-Driving Networks

Nolan Rudolph
University of Oregon
ngr@uoregon.edu

ABSTRACT

Our nation has perpetually progressed toward the era of fully automated devices. Obliging by the expectations of the future, we intend to implement a network capable of running entirely by itself, eliminating the need for a network engineer to constantly scrutinize their network's stability. As expected, the conceptualization of such a concept entails the consideration of an overwhelming number of components and services to allow such a device to run with perfection. This project serves as the first step in creating a basis for this understanding.

1 INTRODUCTION

From the plain old telephone service originating in 1876 to this current decade of machine learning, it has become clear that our nation is constantly advancing in the direction of automation. To allow tools and services to run independently without worry of corruption or fault would not only serve wonders for the efficiency of our technology, but also provides the ability to ensure the security of our confidential information. Of course, the current automated and self learning machines, such as Google's search query system, are constantly making their own advancements, but there are still many devices today that remain unautomated. The concept presented in this paper regards a fully automated networking device referred to as a self-driven network.

A self-driven network in the most rudimentary of definitions is a network that completely runs by itself. Approaching a scope outside of the basis understanding of a self-driven network, no one currently understands how such a device could operate with full self-diagnosing features. Both Juniper Networks and Cisco Systems have attempted to make progress on the subject, however the concept of self-driven networks has fallen outside the scope of a majority of these organization's projects, thus many of the pages hinting toward the production of a self-driven network haven't been updated since 2017. This leaves me with the opportunity to fully conceptualize the creation of such a device through a multitude of methods, the first of which will be the subject of this paper.

An important introductional step to such an esoteric device is to confine ones research to a particular region before attempting universal applications. In this paper, the region of focus will be the University of Oregon (UO), and all netflow data analyzed will be from this location and the requests of users and faculty members encompassed by their domain. I'll begin by presenting a rundown of how, using Python module Scapy, I was able to synthesize packets based on the netflow data that I was granted. Next, the data will be analyzed in multiple data graphing softwares, and then monitored through demonstrations representing the practicality of each softwares usage toward visualizing the characteristics and topology of a network. After attempting the usage of Gephi, Excel, and Tableau for the visualization of the netflow data, I decided to stick with a private hosted graphing mechanism dependent on MySQL queries called Grafana. I then used the software to create visuals for segregated time periods to derive anomalies from the dataset.

Furthermore, I validated the usage of Grafana to understand the applicability of its visualization capabilities to that of malicious events such as the Slow Loris Denial of Service (DoS) attack. I was able to find that the service is fully able to visually replicate any malicious attacks one wishes to pursue, introducing the idea of using such depictions to deploy as test cases in the future testing of a self-driven network. Lastly, I utilize my findings to establish how Grafana is the perfect graphing mechanism for conceptualizing the basis of a self-driven network. For future work, I consider testing even further abilities of Grafana to monitor real-time activity as suggested by the conclusive segment of a research paper regarding the DoS ecosystem written by Mattijs Jonker [1]. I then plan to test and apply anomaly detection strategies on this real-time data to produce real time analysis and diagnosis of ongoing events.

2 BACKGROUND

A self-driven network is the first of its field, that is, minimal approaches have been taken on the subject. The two companies who have tried their hand on the idea have been Juniper Networks, a multinational corporation which markets networking products, and Cisco Systems, a multinational conglomerate which manufactures networking hardware and other high-technology services and products.

Prior to my production of a visualization for the conceptualization of a self-driven network, I thoroughly researched the concept of self-driven networks to get a jump start on the production. I decided I'd first look into the achievements of Juniper Networks. In my findings, the most commendable of advancements toward the subject of self-driven networks is deployable bots that ensure network hosts a particular component of network maintainability [2]. This is no discouragement to my project, however, being as the bots are segregated into three different functionalities, and furthermore, to implement all bots as a means of maintenance for your network wouldn't allow a host to walk away from monitoring its performance and robustness as a self-driven network aims to accomplish. Secondly, the bots continue to rely on this wellknown software defined networking architecture [3], whereas self-driven networks aspires to advance beyond the scope of software defined networking.

Next we observe Cisco Systems and the progress they've made toward self-driven networks. On January 26, 2017, they announced an anticipated release of a self-driving network [4]. Fortunately for my research, the outcome was a lot like Juniper Networks' in the sense that Cisco Systems created a self-driven service for ensuring the most optimized performance of a network with the greatest quality of service. Essentially, Cisco Systems' depiction of a self-driven network is a closed loop system that handles Internet of Things (IoT) application inclusion through instances of virtual segments [5]; This differs greatly from our end goal expectations of a self-driven network.

My project is geared to acquire leverage over these competitors by taking a step back and allowing the ability to perfectly visualize the netflow data of a specified region that will be used for later implementations of a self-driven network. As I have a whole Autonomous System (ASN) as my arsenal, namely the UO, I will be able to perfectly decipher the conditions and characteristics of the traffic and network usage of this ASN all while shaping a device into the image that we assume fit to become a self-driven network. The issue with the enterprise projects released by Juniper Networks and Cisco Systems is they began with tools designed for other networking projects in an attempt to push their projects out quicker for commercial reasons. In my case, I will be starting from the ground up in an attempt to forge this concept of self-driven networks in the most efficient and methodical way possible.

3 PROJECT

3.1 Acquiring the Dataset. Before all else, I needed to acquire the vast dataset of the UO netflow. Through a secure shell, I was able to transfer the data from a UO server to my own personal computer where I discovered the two datasets were comprised of over 280 million entries between 1 AM on May 14th, 2018 and 1 AM on May 15th, 2018. Since these two datasets were recorded within the same time window and were segregated solely because the recording occured at two distinct border router vantage points, I decided to concatenate the two datasets and issue a sort command on the start timestamp via command line interface to acquire my complete netflow dataset. After close observation alongside a description of the netflow data granted to me by the capturer, I found that the categories measured, delimited by commas, were as follows:

1. Start Timestamp in Epoch Format
2. End Timestamp in Epoch Format
3. Source IP Address
4. End IP Address
5. Source Port Number
6. Destination Port Number (or a float type.code if ICMP/IGMP/IPv6 ICMP)
7. IP Protocol Number
8. Type of Service (ToS)
9. Transmission Control Protocol Flags (defaulted to 0 if IP protocol is not TCP)
10. Number of Packets
11. Number of Bytes
12. Router Ingress Port
13. Router Egress Port
14. Source ASN
15. Destination ASN

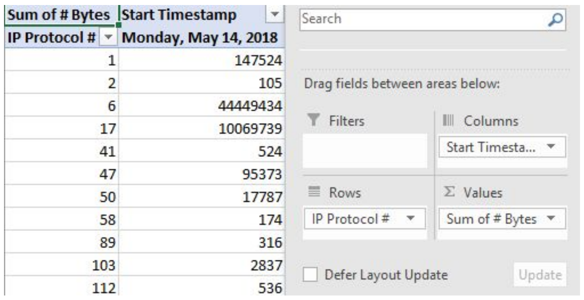
After running a custom built average and summation script on the dataset, the average bytes/packet came to be 1088, the most common IP protocol was TCP, and the most common source and destination ports were HTTPS. The statistics derived from the dataset is on par with what one would expect from a university, thus I was convinced I had a legitimate dataset.

3.2 Packet Synthesizing. After creating a Python script that isolates specified categories of the netflow data [6], I ran the script on the dataset to analyze the IP protocol list. I discovered that the top eight protocols were 1 (ICMP), 2 (IGMP), 6 (TCP), 17 (UDP), 47 (GRE), 50 (ESP), 58 (IPv6 ICMP), and 89 (OSPF_Hello).

With this newly found data, I took to Python, and utilizing the Scapy Python module [7], I was able to create a UO netflow data to packet converter [8]. This converter accounts for these eight protocols alongside an `else` case for generating generic IPv4 packets including the specified protocol and a payload calculated to replicate the entry at hand utilizing the byte and packet attributes.

3.3 Finding the Best Graphing Software. To begin my hunt for the best visualization software to use for my dataset, I began with a program called Gephi. The process for installing and configuring Gephi is quite simple, however the second I began playing with the offered tool set, I quickly realized that it would not be compatible with graphing my particular data points. Implementing a miniscule fraction of my dataset practically crashed the software and upon loading, I received a black square supposedly representing the multitude of points offered by my subset. I reconfigured the software to display my points with outline, and after an hour sort of ensuring no two points touched each other to provide a sense of topology, I soon realized the filtration system was flawed as I attempted and failed to group like IP protocols.

Therefore I switched to an application I was familiar with: Excel. Unfortunately, the preliminary problem of importing my dataset was Excel's table cap of 1.04M entries. Nonetheless, I decided to aggregate my dataset into five minute intervals utilizing an aggregation script [9], and the resulting subsets consisted of an average of 100K entries. Upon attempting to create a pivot table from the data, I quickly realized that Excel had a difficult time understanding float point precision epoch time formatting.



Sum of # Bytes	Start Timestamp
IP Protocol #	Monday, May 14, 2018
1	147524
2	105
6	44449434
17	10069739
41	524
47	95373
50	17787
58	174
89	316
103	2837
112	536

Search

Drag fields between areas below:

Filters

Columns: Start Timesta...

Rows: IP Protocol #

Values: Sum of # Bytes

Defer Layout Update

Update

Figure 1: Excel Pivot Table attempting to segregate data by timestamp

This comes to no surprise as Excel is generally used for Business rather than networking analytics.

Next, I decided to use Tableau, a software designed for interactive data visualization, in effort to represent my data. Upon importing an aggregated subset consisting of approximately 50M entries, I immediately noticed Tableau was unresponsive. Once again I reduced the size of the subset, however this time to 1M entries as I knew Tableau's capabilities of effectively producing visualizations exceeded that of Excel. Importing the subset, I received a two minute timer which concerned me since this was only 1/280th of my total dataset. When the data registered, I instantly made a new dashboard and graphed two segregated subsets of 1M entries side by side.

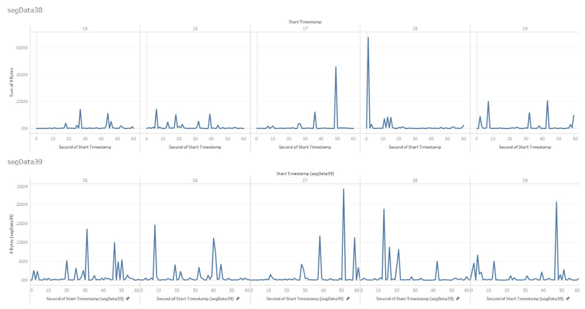


Figure 2: Tableau graphing of two aggregated subsets consisting of 1M entries.

This was much closer to the direction I wanted to take my visualizations toward, however I was discontent with the way Tableau links different subsets based on like variables (i.e. source port). Additionally, if I preferred to graph ten 5M entry subsets, I realized I could be waiting as long as hours for Tableau to produce a graph of my request.

3.4 Discovering Grafana. I decided to retreat to an idea pitched to me by my mentor, Ramakrishnan Durairajan, as a means of graphing the data. The software called Grafana was released in late 2018, therefore I expected the setup and configuration to be long and tedious with minimal documentation. Nonetheless, after applying alterations of Grafana's initialization file, I was able to have Grafana services running on local port 3000.

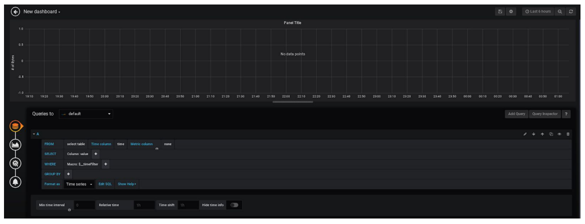


Figure 3: Grafana services' interface hosted on local port 3000.

I quickly discovered that the software utilizes MySQL search queries to isolate data and produce graphs, therefore I knew I'd have to come up with a methodical approach to providing Grafana with the proper dataset.

3.5 Methodical Importation. The first step in this process was designing a MySQL table that fit my netflow data. For brevity, I included the final table produced for netflow dataset importation:

```
CREATE TABLE UONETFLOW(
      startTime double,
      endTime double,
      srcIP varchar(16),
      dstIP varchar(16),
      srcPort varchar(5),
      destPort varchar(5),
      IPProt varchar(3),
      TOSVal varchar(2),
      TCPFlags varchar(3),
      packets int(8),
      bytes int(16),
      routerInPort varchar(5),
      routerOutPort varchar(5),
      srcASN varchar(5),
      dstASN varchar(5)
);
```

Previous models initialized a majority of numeric values as type `int`, `float` or `double`, however Grafana only recognizes strings as potential filters for data graphing. Next I utilized a bash shell script [10] to iteratively concatenate variations of "UONETFLOW" titles with identical fields to allow the initialization of a series of tables with minimal effort. After installing these tables into MySQL server, I used another bash shell script [11] to saturate said tables with the aggregated subsets of my choosing. After this final step, my data was ready to be graphed.

3.6 Graphing Subsets. My first step in effectively graphing the netflow data in each MySQL table was to find the first and last start timestamps. This can be done in a very simple manner by issuing two MySQL commands:
First entry: `SELECT startTime FROM [TABLE] LIMIT 1;`
Last entry: `SELECT startTime FROM [TABLE] ORDER BY startTime DESC LIMIT 1;`
Making sure to choose the proper time zone, I converted each epoch startTime to an interpretable format and inputted the interval as Grafana's time frame. I then selected the subset whose time interval I just implemented, and the first graph of my project was born.

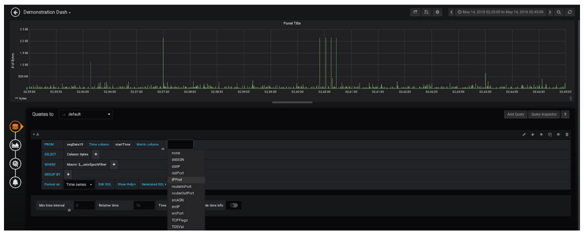


Figure 4: First Grafana graph displaying the list of potential filters.

I found the most optimal way to graph data is through thirty minute subsets, where Grafana then displays ten minutes at a time. Therefore, a salient specification of the rest of this paper is the usage of this method for graphing by time.

3.7 Segregation of Data by Time of Day. After graphing various time periods of the dataset, I theorized that a potential way to discover characteristics of the recorded traffic was to segregate the dataset into subsets by time of day. Using the method defined above, I divided the data into four new tables called MORNING, AFTERNOON, EVENING, and NIGHT. MORNING consisted of netflow entries with a startTime between 8:00 AM and 8:30 AM with a net total of 5.3 M entries. AFTERNOON consisted of times between 12:00 PM and 12:30 PM with a total of 6.8 M entries. EVENING consisted of times between 6:00 PM and 6:30 PM with a total of 6.9 M entries. Lastly, NIGHT consisted of times between 10:00 PM and 10:30 PM with a total of 5.5 M entries. The number of entries per dataset is reassuring as we expect to see an increase in user traffic during the middle of the day.

The next step in the analysis of time of day netflows was to graph each subset.

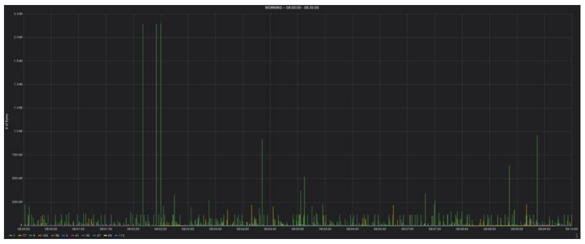


Figure 5: Graph of MORNING subset.

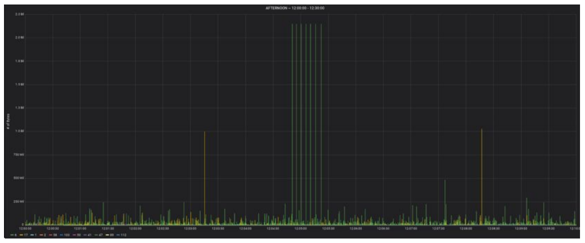


Figure 6: Graph of AFTERNOON subset.

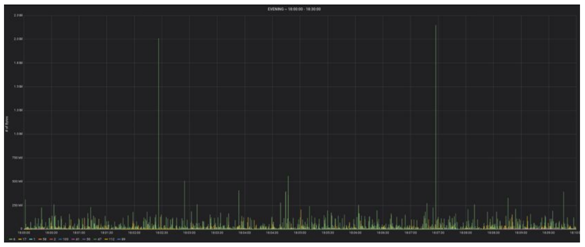


Figure 7: Graph of EVENING subset.

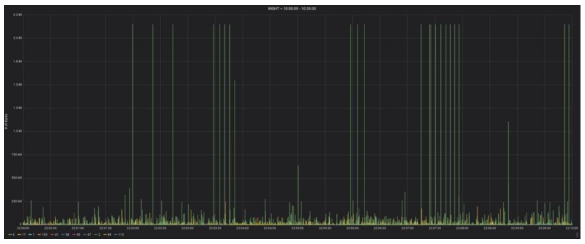


Figure 8: Graph of NIGHT subset.

As green is representative of the TCP protocol which happens to be the protocol most common amongst the primary dataset, and being that a majority of our graph is considerably green, we instill further confidence in our graph being an accurate depiction of our dataset. Since the AFTERNOON and EVENING subsets held the most entries, it makes sense to see the packets a little more clustered than MORNING and NIGHT. Perhaps the most interesting aspect of our charts is how the NIGHT subset was responsible for a large portion of the spikes in the graphs. I decided to take a deeper look at this aspect.

3.8 Analyzing Netflow Spikes. Due to the large number of spikes found in the subsets, I knew I had to forge another method to clarify the purpose of such spikes in an efficient and adaptable manner. I decided the best strategy to approach this problem was to create a CSV file of start and end times containing the spikes alongside a threshold that, once passed, would indicate an anomaly (to isolate spikes from normal packets).

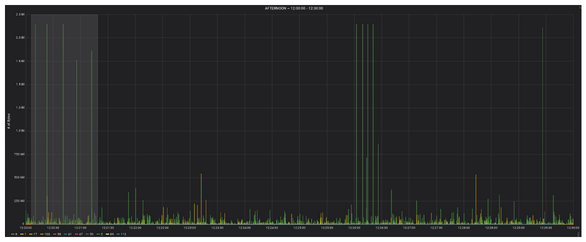


Figure 9: Truncating time range to highlighted region 12:20:10-12:21:15.

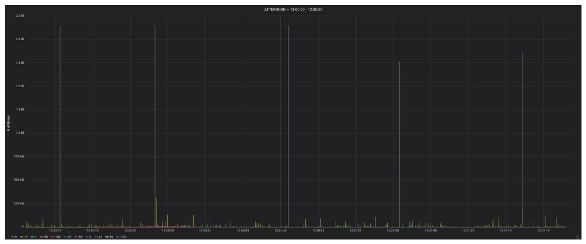
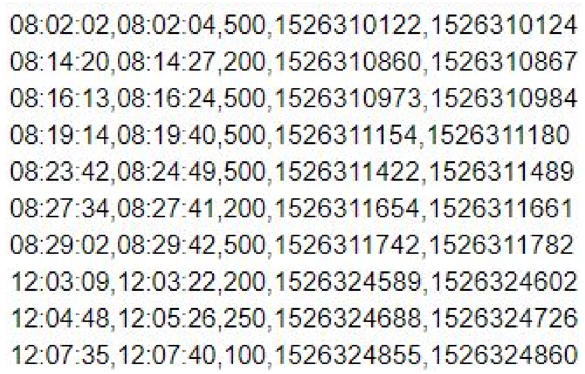


Figure 10: Resulting graph enlargement from time range truncation.

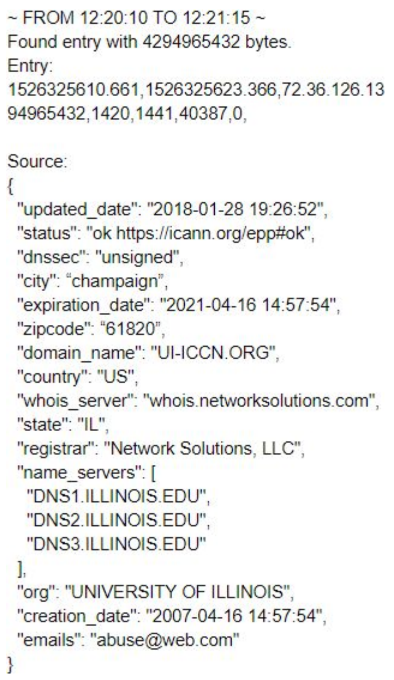
The resulting CSV entry for this particular example became: `22:04:49,22:05,07,150,1526360689,1526360633`, denoting respectively: standard format start time, standard format end time, threshold in millions, epoch start time, and epoch end time. After searching through all three ten minute sections of the thirty minute subsets above, I was able to derive a list of 38 spikes to analyze.



```
08:02:02,08:02:04,500,1526310122,1526310124
08:14:20,08:14:27,200,1526310860,1526310867
08:16:13,08:16:24,500,1526310973,1526310984
08:19:14,08:19:40,500,1526311154,1526311180
08:23:42,08:24:49,500,1526311422,1526311489
08:27:34,08:27:41,200,1526311654,1526311661
08:29:02,08:29:42,500,1526311742,1526311782
12:03:09,12:03:22,200,1526324589,1526324602
12:04:48,12:05:26,250,1526324688,1526324726
12:07:35,12:07:40,100,1526324855,1526324860
```

Table 1: First ten entries of the thirty-eight spikes in the CSV spike file.

I utilized an anomaly detection script [12] able to decipher this CSV file to first locate the spikes and then give details on the sender and receivers of the packets using the Python module *whois* [13].



```
~ FROM 12:20:10 TO 12:21:15 ~
Found entry with 4294965432 bytes.
Entry:
1526325610.661,1526325623.366,72.36.126.13
94965432,1420,1441,40387,0,

Source:
{
  "updated_date": "2018-01-28 19:26:52",
  "status": "ok https://icann.org/epp#ok",
  "dnssec": "unsigned",
  "city": "champaign",
  "expiration_date": "2021-04-16 14:57:54",
  "zipcode": "61820",
  "domain_name": "UI-ICCN.ORG",
  "country": "US",
  "whois_server": "whois.networksolutions.com",
  "state": "IL",
  "registrar": "Network Solutions, LLC",
  "name_servers": [
    "DNS1.ILLINOIS.EDU",
    "DNS2.ILLINOIS.EDU",
    "DNS3.ILLINOIS.EDU"
  ],
  "org": "UNIVERSITY OF ILLINOIS",
  "creation_date": "2007-04-16 14:57:54",
  "emails": "abuse@web.com"
}
```

Figure 11: Example of an outputted entry from spike analysis script, regarding Figure 10.

The outputted text file contained 233 pages of data which allowed me to discover some interesting statistics about the kinds of anomalies that UO currently allows.

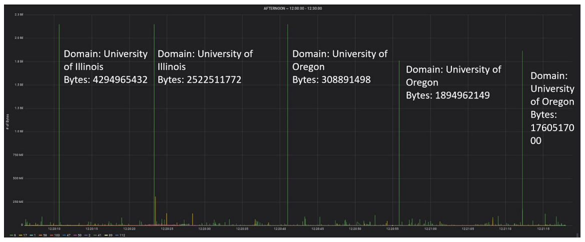


Figure 12: Domain and byte labels applied to anomalies in Figure 10.

Though remaining classified, I was able to derive a handful of domains that the UO allows to bypass anomaly detection. This is just one example of how Grafana could be used to find data for specified fields of interest.

3.9 DoS Attack Applications. A final trial I wanted to run was to see if Grafana could provide a useful visualization for the effect of DoS attacks. For this segment, I decided to use the Slow Loris DoS attack to be the choice of influence on my dataset. The Slow Loris DoS works by saturating all TCP and UDP ports by establishing a connection through spoofed IPs on every port and maintaining this denial of service by sending approximately twenty bytes a minute to the server in attempt to simulate an extremely slow connection. The server then maintains this connection as it acknowledges that the client still requests information, and thus all services are under the domain of the perpetrator. To emulate this attack, I used a Slow Loris DoS script [14] with a briefing period of 1M entries to influence my MORNING subset, and then graphed both the DoS'd MORNING subset and original MORNING subset within the same time period side by side using Grafana's dashboard interface.

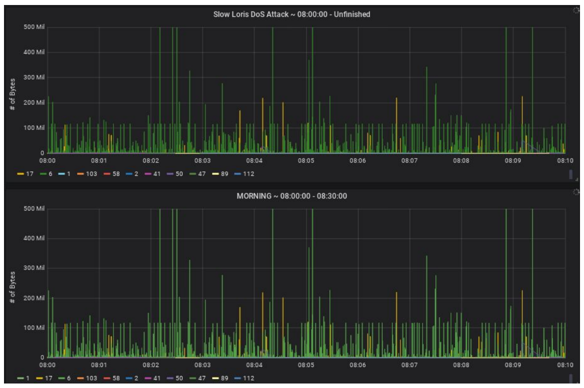


Figure 13: First ten minutes of DoS Attack (Top: DoS Influenced, Bottom: Original)

In the first ten minutes of the DoS attack, the two subsets still look identical. This is due to the fact that I set a briefing period of 1M entries to allow the DoS attack to slowly take effect, as a realistic DoS attack would.

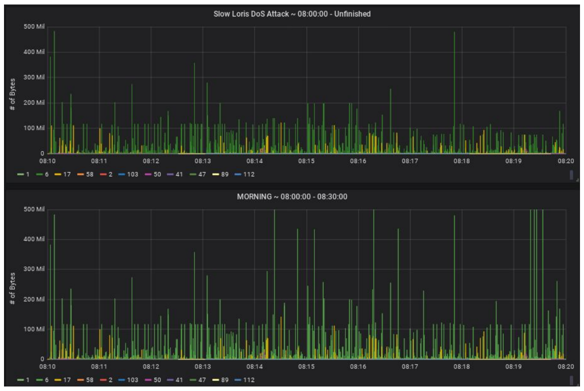


Figure 14: Second ten minutes of DoS Attack (Top: DoS Influenced, Bottom: Original)

Around the ten to twenty minute range of the DoS Attack, I begin to witness changes in the DoS'd subset which trends a loss of packets and bytes with respect to our original subset.

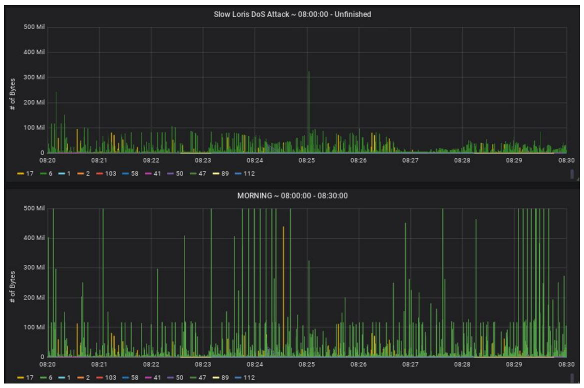


Figure 15: Third ten minutes of DoS Attack (Top: DoS Influenced, Bottom: Original)

Viewing the twenty to thirty minute range, we can clearly see the effect that our DoS attack is having on our DoS'd subset. The original subset might as well be considered completely distinct from our DoS'd subset with no remote similarities apart from the single spike at 08:25:00. This perfectly demonstrates the capabilities of Grafana and promotes the idea of trying to visualize even further examples of DoS attacks and other malicious infiltrations of a network.

4 SUMMARY

The conceptualization of an innovative cornerstone project such as a self-driven network is heavily dependent on the ability to visualize particular events and scenarios. By starting from the ground up, I have allowed myself to create a solidified and impenetrable basis for my project, leveraging myself above other competitors such as Juniper Networks and Cisco Systems. I began by receiving a dataset chalk full of 280 million netflow entries of user and external traffic at the University of Oregon. After many trials of different softwares, I came across Grafana, an open source visualization tool for metric analytics which is heavily compatible with netflow data. Once fully configured, Grafana was capable of handling millions of entries while simultaneously indicating each packet's IP protocol number. I decided to test Grafana's capabilities of visualizing different scenarios by first trying random ten minute segments, then segregating our dataset into MORNING, AFTERNOON, EVENING, and NIGHT subsets and plotting those against one another. Next I used Grafana to identify the spikes in our different subsets, and derived a considerable amount of details about the types of anomalies UO allows into their networking services. Lastly, I applied the Slow Loris DoS attack to the MORNING subset, and upon graphing the two subsets side by side, I were able to capture the true potential behind Grafana's ability to visualize the effects of malicious attacks.

5 CONCLUSIONS

In this research paper, we were able to discover the visualization software Grafana and prove that its services were more than sufficient to analyze any aspect of networking that a user wishes to visualize. In this way, it provides the perfect platform for conceptualizing components to be implemented into this innovative idea of a self-driven network.

Throughout this project, memory space and the overall performance of my personal computer was not sufficient enough to handle some of the functionalities of Grafana. This was only due to the fact that I mounted my Linux system onto an older Hard Disk Drive, and the fact that I did not edit Grafana's initialization file to load temporary data to a file other than my root directory. The combination of these two faults lead to a decent amount of irritation and a hefty number of crashes and recurrent "Filesystem Root has #Mb left." warnings. If I were to redo my project, I would have undoubtedly mounted my machine to a solid state drive, set my Grafana's default temporary data storage directory to /tmp, and set up my NVDIA graphics card for dual booting purposes to allow quicker graphing.

As for the potential knowledge acquisition one could derive from Grafana's capabilities of interpreting and visualizing netflow data, I have only touched the tip of the iceberg. Therefore, the foreseeable next steps in the usage of Grafana as a basis for the conceptualization of a self-driven network are as follows:

1. The idea of introducing real-time data fusion into graph production and manipulation thoroughly intrigues me, and I hope to gain access to these abilities in future projects.
2. I wish to try many more of Grafana's graph styles including heat maps and gauges. There are also a handful of contributor created styles that have peaked my interest, such as the phalanx dashboard [15].
3. Though Grafana doesn't contain built in scripts to create influences on data, I wish to better my understanding of MySQL and then use Grafana to implement table manipulations to mimic a multitude of DoS attacks inline.

6 REFERENCES

6.1 Directly Referenced Resources

[1] https://conferences.sigcomm.org/imc/2017/papers/imc17-103.pdf

[2] https://www.juniper.net/us/en/solutions/automation/bots/

[3] https://www.juniper.net/us/en/solutions/automation/platform/

[4] https://newsroom.cisco.com/feature-content?articleId=1816954&type=webcontent

[5] http://ciscodna.cycloneinteractive.net/interactive/#readiness-model

[6] https://github.com/NolanRudolph/UONetflow/blob/master/PyScripts/isolateArg.py

[7] https://github.com/secdev/scapy

[8] https://github.com/NolanRudolph/UONetflow/blob/master/PyScripts/netflowPackager.py

[9] https://github.com/NolanRudolph/UONetflow/blob/master/PyScripts/createWindows.py

[10] https://github.com/NolanRudolph/UONetflow/blob/master/Bash/iteratoryReplace.sh

[11] https://github.com/NolanRudolph/UONetflow/blob/master/Bash/saturateTables.sh

[12] https://github.com/NolanRudolph/UONetflow/blob/master/PyScripts/findAnomalies.py

[13] https://pypi.org/project/whois/

[14] https://github.com/NolanRudolph/UONetflow/blob/master/PyScripts/slowLorisDoS.py

[15] https://grafana.com/dashboards/4208

6.2 Other Resources

(1) Roshan Lal Neupane, Travis Neely, Nishant Chettri, Mark Vassell, Yuanxun Zhang, Prasad Calyam, Ramakrishnan Durairajan, Dolus: Cyber Defense using Pretense against DDoS Attacks in Cloud Platforms

(2) Philippe Biondi and the Scapy community, Scapy Documentation -- Release 2.4.2-dev, Mar 09, 2019

(3) Secynic, ipwhois Documentation -- Release 1.1.0, Feb 02, 2019

(4) Mattijs Jonkey, Anna Sperotto, Roland van Rijswijk-Deij, Ramin Sadre, and Aiko Pras. Measuring the Adoption of DDoS Protection Services. In*Proceedings of the 2016 ACM Internet Measurement Conference, pages 279-285, 2016*