Assignment 2 – Network Monitoring

CSCI 6706 Network Design

Christian Liu – B00415613
Computer Science Department
Dalhousie University
Halifax, NS
Chris.liu@dal.ca

Table of Contents

ABSTRACT

The assignment 2 is asking to collect, monitor and analyze the network data by utilizing several popular tools. It looks like a mini project. We need to follow the process, such as:

- Setup Work Environment: Cacti or PRTG.
- Discover Network Devices by configurations of Cacti or PRTG.
- Collect traffic / packets through Wireshark.
- Alternatively, we can collect data by Argus Server and Client
- Analyze the data from Wireshark with Argus client.
- Visualize the data with Ragraph, Python or Tableau.
- Advanced classification with ML.

KEYWORDS

Network Monitoring, Cacti, Wireshark, Argus, PRTG, Tableau, Jupyter, ML, Classification

1 Introduction

Assignment 2 is about network monitoring and analytics. In this assignment 2, I followed the instructions from Classes and Labs 3, 4, 5. Specifically I scheduled the workflow as below:

- Utilized Cacti Automation Devices Detection to find possible devices over my network.
- Utilized PRTG which is friendly monitor tool of Windows to generate the most suitable statistics graph for each device and application over my network.
- Utilized WireShark to capture the traffic / packets for one-hour duration on several continuous days, and obtain PCAP files ready for Argus; Plus, I utilized "tshark" to process raw PCAP files and prepared hosts files for preliminary analysis.
- Utilize Argus to devote the PCAP files generated by WireShark, then generated ARGUS files and CSV files. During this process, I followed several subtle steps suggested by Lab 5:
    - Generated ARGUS file by several tunable parameters.
    - Configured filters by Ralabel to generate labels for existing ARGUS file.
    - Read labelled ARGUS files by RA command, then converted those files into CSV which can be used for visualization and analytics
- Utilize Ragraph to generate static simple analyzed charts, then utilize Tableau to generate more complex interactive charts.
- Utilize Python Matplotlib and ML to visualize and analyze the data. Then classify the data by certain classification algorithm.

2 Environment Configuration

Environment setup and configuration are the big part of this assignment. It involved many software tools and it is time consuming. My host PC is Windows 10 platform. Based on Windows 10, VMWare workstation 15 is the virtual machine installed on my host machine. The following software or tools were used for this assignment:

- Cacti: an extremely popular Network Monitoring, Visualization Tool can be installed on both Windows and Linux liked system. To make it work, we need Apache, MySQL, and PHP interpreter.
- PRTG: an alternative software tool of Cacti. Especially it can work more friendly then Cacti.
- WireShark: an immensely popular and commonly used for network traffic monitoring, capturing, filtering, analyzing, and visualizing. Its installation becomes straightforward nowadays.
- Argus (OpenArgus): it comes with two different packages: Argus-sever and Argus-client. Both can be easily installed on Linux liked system, it contains many popular network data analytics tools, such as: RA, Racluster, Rasort, Ragraph, Rafilteraddr and Rahisto etc. Usually it works with WireShark together, it can capture network traffic with Argus-sever though.
- Tableau: an especially useful and straightforward data science tool. The good thing of it is we can complete our tasks mostly by GUI only. And effects are usually aligned with industrial standard and impressive.
- Jupyter notebook: a tool runs with Python language usually. It can allow Data Scientist to interactive with their data sets and analyze data in real time as long as session is alive.

2.1 SNMP Server and Agent Configuration

Firstly, enabled it on my local:

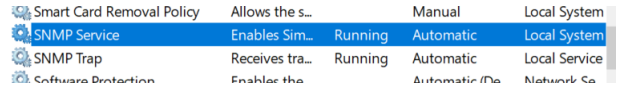


Figure 1 - Enable SNMP Service

Configure the parameters:

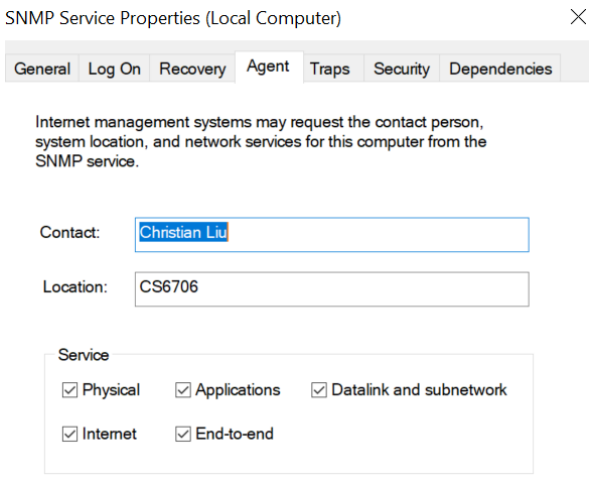


Figure 2 - SNMP Service Configuration

Configure the security:

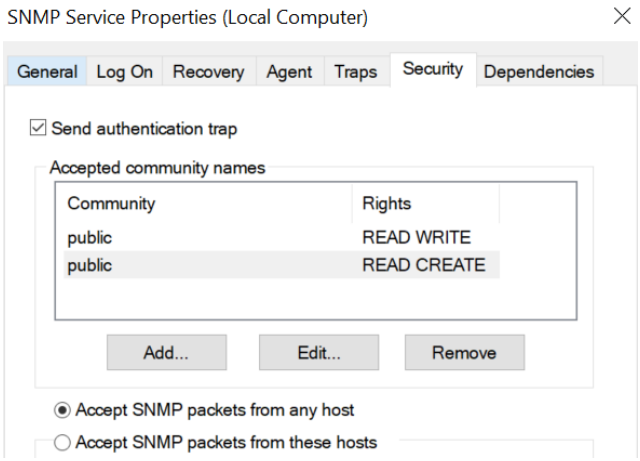


Figure 3 - Configure the network security

For other SNMP enabled devices, which are under my control, I need to do these configurations:

/etc/snmp/snmpd.conf

```
#  Listen for connections from the local system only
#agentAddress  udp:127.0.0.1:161
#  Listen for connections on all interfaces (both IPv4 *and* IPv6)
agentAddress udp:161,udp6:[::1]:161
```

To setup interval polling for Cacti, we can do this as below:

```
chris@DESKTOP-I7H08DT:~$ sudo cat /etc/cron.d/cactipoller
*/2 * * * * php /var/www/html/poller.php > /dev/null 2>&1
```

Figure 4 - Configure Poller

After configuration of local, we can see it is up for running on Cacti:

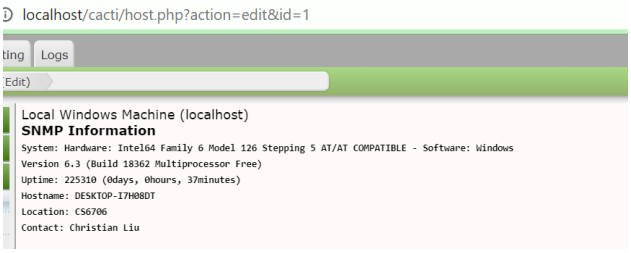


Figure 5 - Local host SNMP results

2.2 SNMP Devices Automation Detection

The next I need to discover more actual SNMP enabled device on my network, so **that I setup SNMP scanner to automatically scan my network for any possible SNMP enabled device**. There is a feature of the Cacti located on "Automation -> Network" page. I followed Lab 4 suggestion to extend discovering range of IP (192.168.*.* and 192.0.*.*), like screenshot below:

Figure 6 - Cacti SNMP Automation

Screenshot tells I am trying to discover reachable devices over my network within the IP range (192.168.*.* and 192.0.*.*)

Interesting things happened. I discovered Four SNMP device including one unknown device which is SNMP enabled over my personal network by utilizing the Cacti Automated Discovery:

Figure 7 - Discovered SNMP devices

Two IP-P2 Remote Power Control Devices (one's IP is 192.0.15.98 and another is 192.0.11.125). As I researched why it appeared in my network, I found usages in https://www.controlbyweb.com/webswitch/#:~:text=Two%2C%20Independent%20IP%20Controlled%20Power,or%20monthly%20subscriptions%20are%20required.)

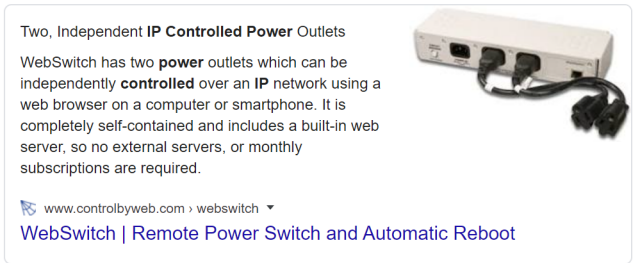


Figure 8 - Power Control Definition: https://www.controlbyweb.com

I owned one NAS family cloud drive (Synology)\, I enabled SNMP service on it. And manually added it in research groups.

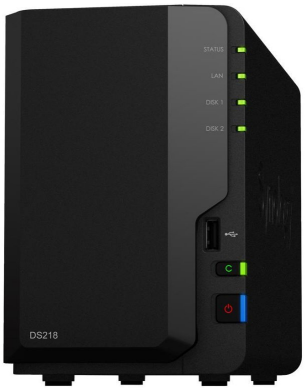


Figure 9 - Synology NAS Drive

Another automatic discovered device is called "LANSide-RobertsonFarms", the name is automatically prompted. I am not sure what this device is for.

The last device I can discovered is "unknown" device which is enabled with SNMP, but I am not sure how and why it appeared in my network.

In a summery, the devices I am going to investigate are:

Device Description	Hostname	ID	Graphs	Data Sources	Status
IP-P2 Remote Power Control	192.0.15.98	11	13	20	Up
IP-P2 Remote Power Control (Another)	192.0.11.125	12	13	20	Up
LANSide-RobertsonFarms	192.0.12.42	7	12	19	Up
Local Windows Machine	localhost	1	10	10	Up
NAS-Synology	192.168.0.12	6	22	16	Up
Unknown-Discovered-Device	192.0.145.27	8	12	19	Up
					All 6 Devices

Figure 10 – Network Devices Summary

3 Traffic generation and Data Collections

This section is the preperations of three questions asked in assignment 3. Here I am using the recommended tools in our class, such as WireShark and Cacti.

About Cacti, I talked about some configurations of device discovery in previous section. Here I want to mention a little bit Cacti configuration about data collection. Cacti collects data through the Poller, which is physical poller.php exectuable script within Cacti application.

Poller.php is executed as a scheduled task in Windows platform, or Cron job scheduled in Linux liked platform. It periodically ping desinations by ICMP and execute some predefined SNMP operations over the desinations, like GET, GETBULK, GETNEXT etc. It can be configured in host platform, for example in Windows which I used by now:

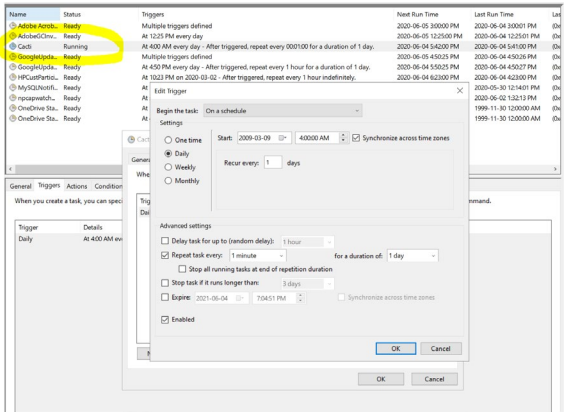


Figure 11 - Cacti / PRTG poller configuration

This screenshot tells we execute Poller.php every 1 minutes (ideally, in reality my laptop shutdown multiple times because of overheated). Therefore, it can generate some ICMP, TCP and SNMP traffic etc.

During the data collection period, I kept running applications over my NAS drive and my Docker image which is running with with Argus. Argus was used for Question 3 in order to analyze and visualize the classification. Remote IP controllers seem to be up for all the time, even though I am not certain how and why they appear in my network as SNMP agents.

Alternatively, PRTG network monitoring tools suite also can generate traffic, discover devices and sensors, analyze and visualize the data we need:

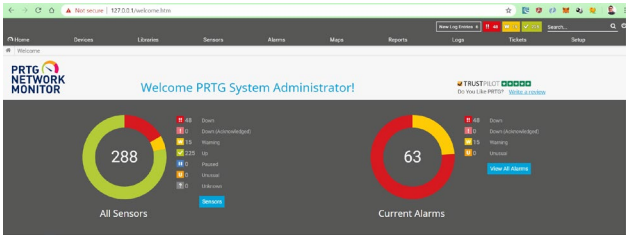


Figure 12 - PRTG Dashboard

After installation of PRTG on Windows, it automatically generates one small Daemon process on the background. The "PRTG Probe" is trying to generate the traffic to the possible destinations with different protocols.

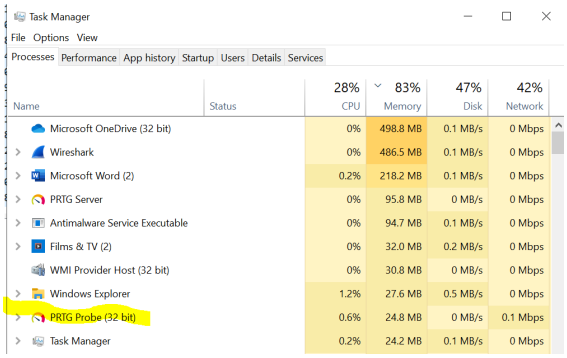


Figure 13 - Poller running as process for PRTG

3.1 Data Collection Periods

I collected data three times on the separated dates and each time collection cost one and half hours. During the data collection, I

run the PRTG and Wireshark at the same time. The data collection includes two aggregations:

- **Data for SNMP enabled devices over my network**
- **Data for Applications on the local Prober**

The three periods are:

- **2020-06-07 from 20:05 PM to 21:05 PM**
- **2020-06-08 from 9:04 AM to 10:04 AM**
- **2020-06-09 from 4:02PM to 5:02 PM**

4 Question 1: PRTG Monitoring

Because I am working on Windows platform, PRTG seems to be more friendly to me, compared with Cacti. Therefore, in order to answer the Question 1, I was mainly using PRTG tools to constantly monitor the devices over my network during several days' investigation.

4.1 SNMP Devices Discovering

I set it as scanning all targeted devices every 1 minutes, so that the traffic can be consumed by wireshark used for question 2.

Regarding about question1, PRTG is so powerful tool, which is able to generate visualized reports automatically, just by its default setting like below:

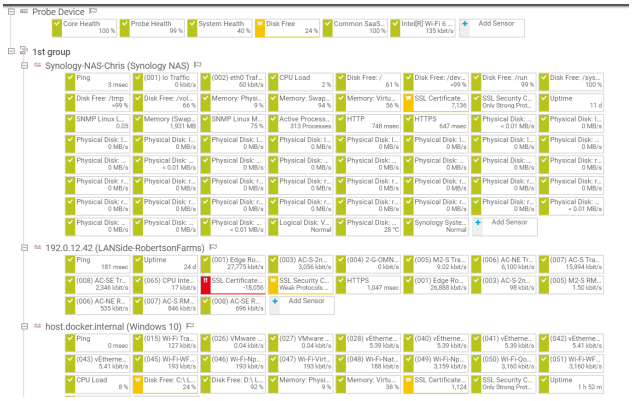
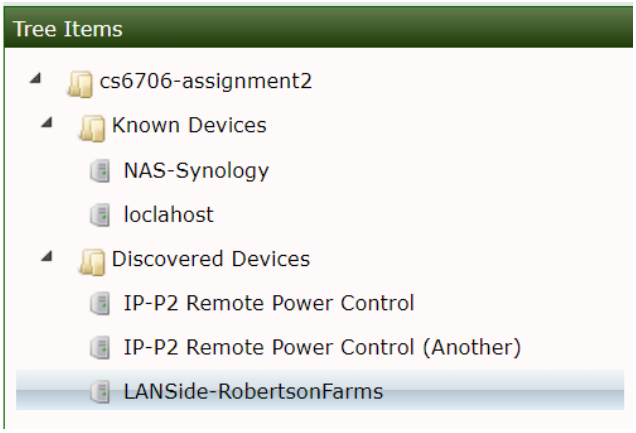


Figure 14 - My network devices overview on PRTG

However we should focus on several devices / applications which are SNMP enabled. Therefore I chosed following devices to investiate:



It organized 5 devices which I am trying to investigate into a tree topology.

Next we need to add suitable graphs to each of devices, because default "Cacti status" generated many graphs which are not working with assgined devices. Moreover, we can add customized graph especially to the devices, based on their own SNMP MIB structure. I will discuss about it in the last improvement section.

Here I want to explain a little bit more of PRTG configuration of features selections for each of devices:

4.2 SNMP trap sensor setup

The interesting thing on PRTG is that we can easily setup SNMP trap sensor on each of devices to track the information we need for certain devices.

For example: we are adding SNMP trap sensor to NAS device:

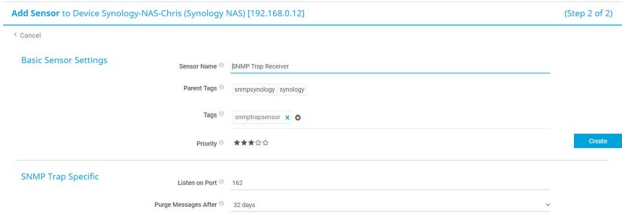


Figure 15 - SNMP trap setup on PRTG

The screenshot above tells us about the SNMP trap sensor basic parameters, such as port number, how long messages can be kept, priority over other sensor etx.

Then we can customize the filters with this SNMP trap further:

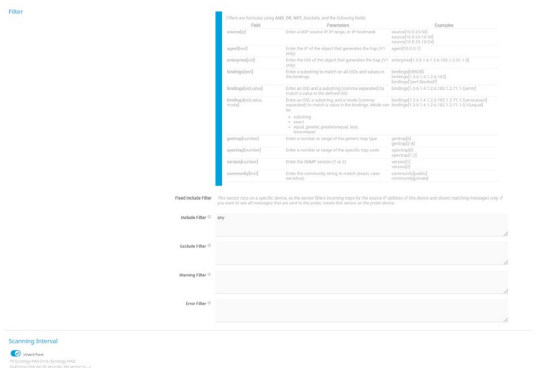


Figure 16 - SNMP trap filter configures on PRTG

The screenshot above tells us about different filters we can customize, including logical expression between different filters. At last, we can indicate Scanning Interval of this sensor.

4.3 Application Discovering – Customized Sensors

Regarding about the application usages during the data collection period, PRTG provides very useful feature to allow user to collect customized application during a certain time windows.

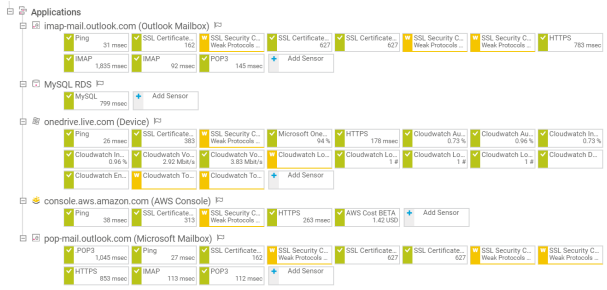


Figure 17 - Applications as sensors on PRTG

- **Email Application**

Adding an customized senesor (Application), we need to follow the process of adding Device -> adding Sensor to that Device on PRTG. For example, I added "Microsoft Mail Sever" as investiate targets. Firstly I needed to add MS email POP/SMTP server (pop-mail.outlook.com/SMTP-Out.outllook.com), then add a sensor to this device.

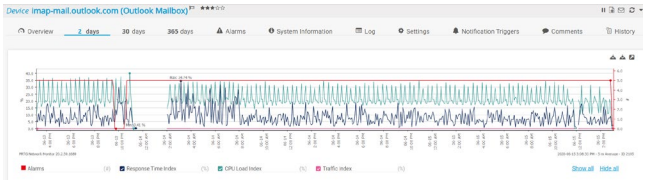


Figure 18 - Outlook Email Traffic Overview

- **OneDrive**

OneDrive is another application I was monitoring, because it is one of the most used cloud drives, I used so far. We can see the red line indicates traffic during the recent several days from the screenshot below:

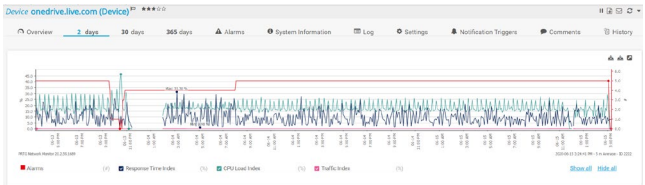


Figure 19 - Microsoft OneDrive Overview

- **MySQL Remote Database**

The remote MySQL database is the AWS RDS I created for another course TA responsibility. I used it very often recently. Therefore, it is worth to add this application to my monitoring targets list.

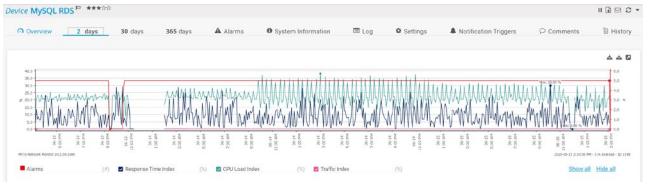


Figure 20 - MySQL RDS Traffic Overview

4.4 Overall Devices and Sensors

Overall, I configured whole Devices and Sensors including identified SNMP enabled devices and applications. Like the screenshot below:

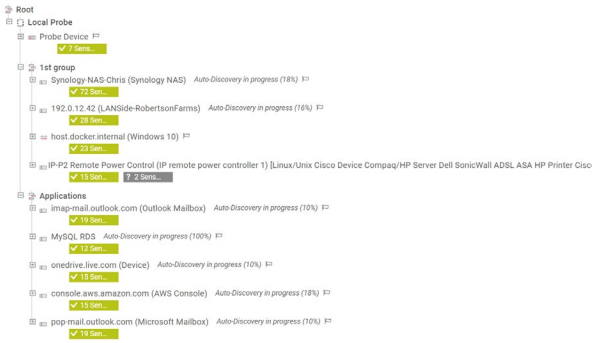


Figure 21 - Monitoring Devices Treeview

PRTG is able to visualize the discovered devices and customized applications as a map:

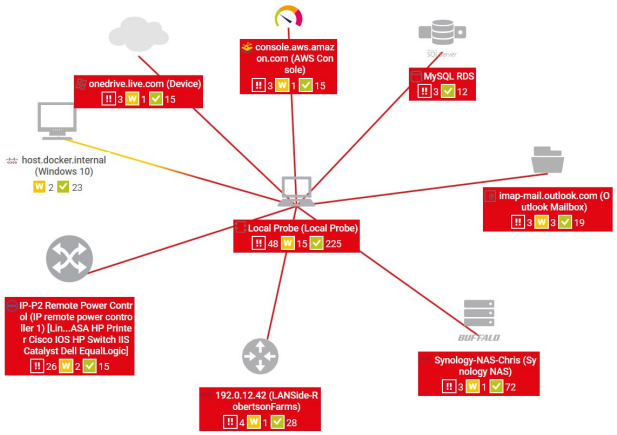


Figure 22 - Star Schema of network connection with host

4.5 Visualization for each Device and Application

It is worth to mention: PRTG server and prober can run as background process on Windows platform. We do not have to bother to schedule a time to run it. What we need to do is specifying the customized time slot from historical data.

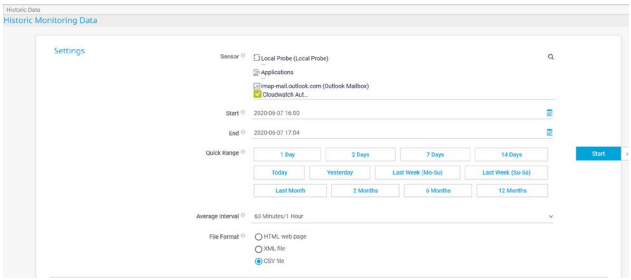


Figure 23 - PRTG Prober configuration

From the overall network devices list, we will show each output of devices (Please note: ping can evaluate RTT performance very well and SNMP traffic measurment can evaluate throughputs effiecently. Therefore, these two are the major methodes by using PRTG):

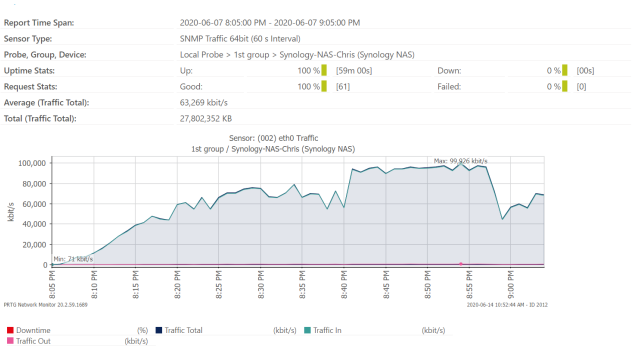
.5.1 Synology-NAS-Chris (Synology NAS)

Synology NAS is my home cloud device, which is enabled SNMP. And it has many possible statistic charts detected by PRTG. Below is the overall dashboard of this device.

PRTG gives very friendly reports of monitoring with several popular indicators, such as Response Time Index, Traffic Index and CPU load index.

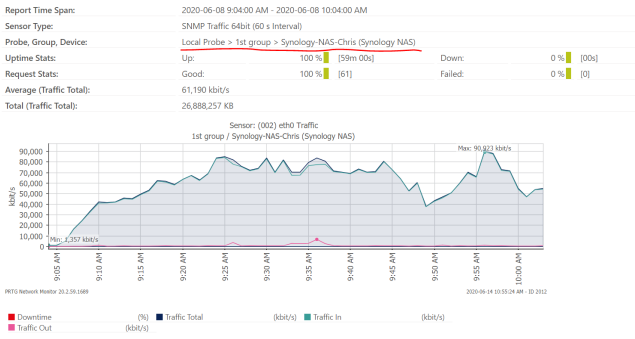
Next I setup a senor to monitor the bandwidth and traffic on the main interface which communicates with host machine vis SNMP protocol.

- Day 1 (2020-06-07 from 20:05 PM to 21:05 PM)



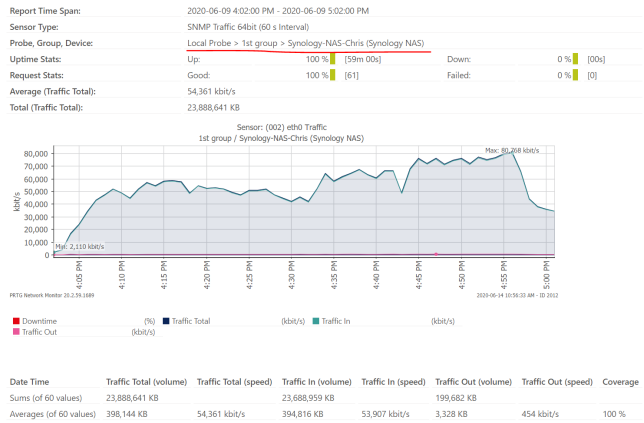
Date Time	Traffic Total (volume)	Traffic Total (speed)	Traffic In (volume)	Traffic In (speed)	Traffic Out (volume)	Traffic Out (speed)	Coverage
Sums (of 60 values)	27,802,352 KB		27,623,686 KB		178,666 KB		
Averages (of 60 values)	463,373 KB	63,269 kbit/s	460,395 KB	62,862 kbit/s	2,978 KB	407 kbit/s	100 %

- Day 2 (2020-06-08 from 9:04 AM to 10:04 AM)



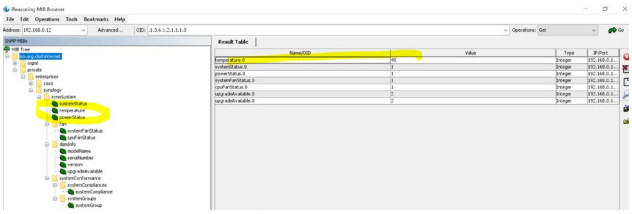
Date Time	Traffic Total (volume)	Traffic Total (speed)	Traffic In (volume)	Traffic In (speed)	Traffic Out (volume)	Traffic Out (speed)	Coverage
Sums (of 60 values)	26,888,257 KB		26,511,820 KB		376,437 KB		
Averages (of 60 values)	448,138 KB	61,190 kbit/s	441,864 KB	60,333 kbit/s	6,274 KB	857 kbit/s	100 %

- Day 3 (2020-06-09-4-02PM to 5-02 PM)

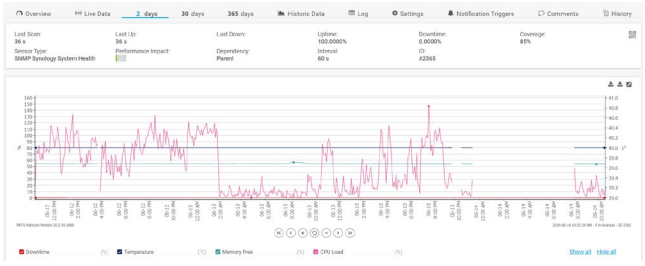


Date Time	Traffic Total (volume)	Traffic Total (speed)	Traffic In (volume)	Traffic In (speed)	Traffic Out (volume)	Traffic Out (speed)	Coverage
Sums (of 60 values)	23,888,641 KB		23,688,959 KB		199,682 KB		
Averages (of 60 values)	398,144 KB	54,361 kbit/s	394,816 KB	53,907 kbit/s	3,328 KB	454 kbit/s	100 %

Moreover, because each device type has their own SNMP MIB structure, I download Synology System MIB from their official website (http://www.circitor.fr/Mibs/Mib/S/SYNOLOGY-SYSTEM-MIB.mib) to check out which is the most valuable variables needed to be focus on for my Synology Device. I found the "temperature - **.1.3.6.1.4.1.6574.1.2.0**" might be interesting variable I need to focus on:

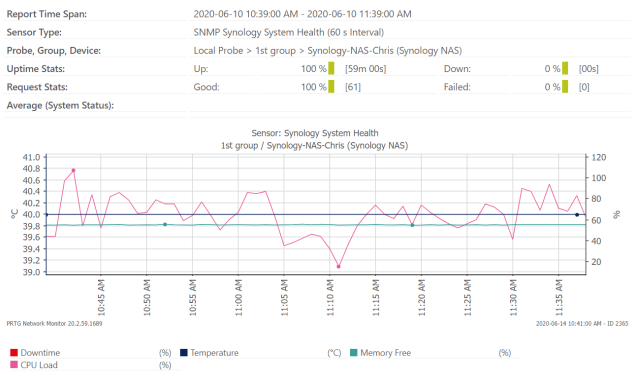


Then, I was able to set SNMP trap on tool to let it monitor that value for checking system health:

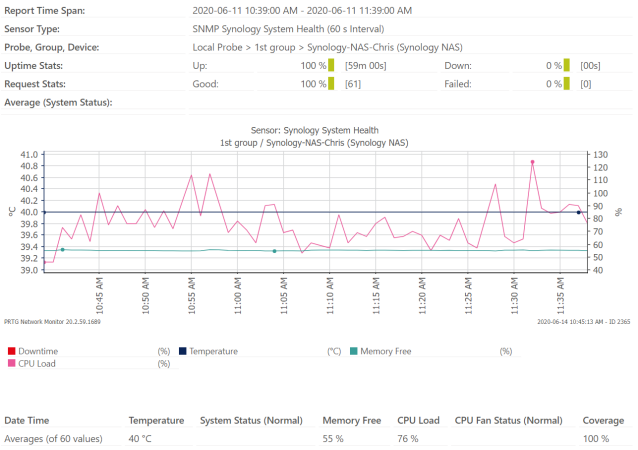


From screenshot above, by the power of SNMP trap setting, I was able to aggregate "Templatature", "CPU Load" and "Memory free" values to decide systeam health over time. Next we will do comparsion of all three days:

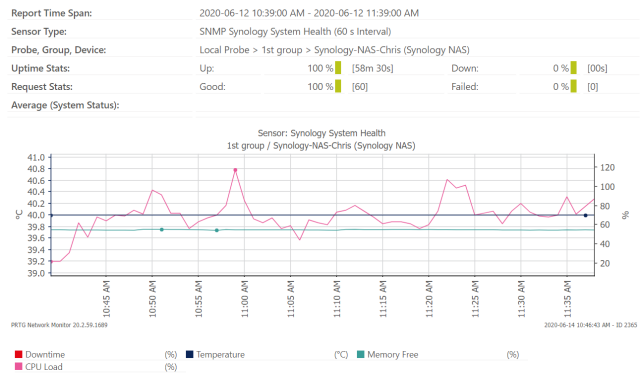
- Day 1 (2020-06-010 from 10:39 AM to 11:39 AM)



- Day 2 (2020-06-11 from 10:39 AM to 11: AM)



- Day 3 (2020-06-12-10:39PM to 12-39 PM)



From the system health check for my NAS drive, we can find the system temperature varied around 40 degree. Overall performance is stable.

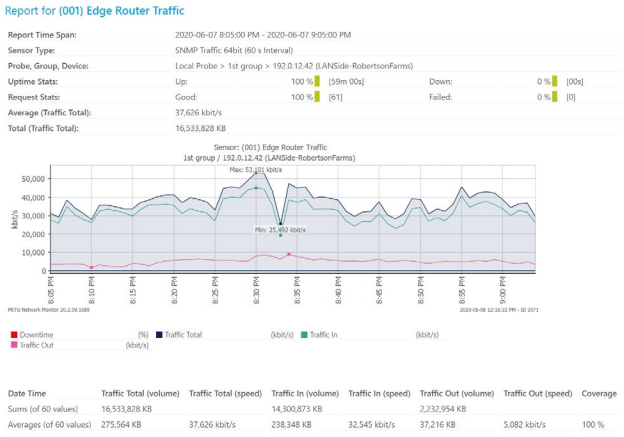
.5.2 192.0.12.42 (LANSide-RobertsonFarms)

Let us look at overall monitoring over 2 days for this device:

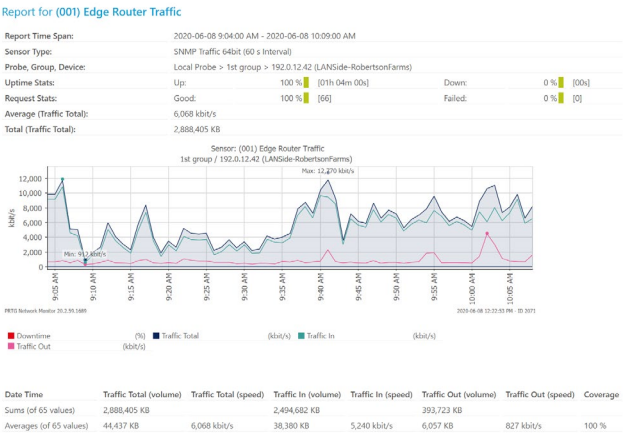
PRTG gives very friendly reports of monitoring with several popular indicators, such as Response Time Index, Traffic Index and CPU load index.

This device was automatically discovered by Cacti automation. I setup SNMP trap for this device to monitor the bandwidth and traffic on each available interfaces.

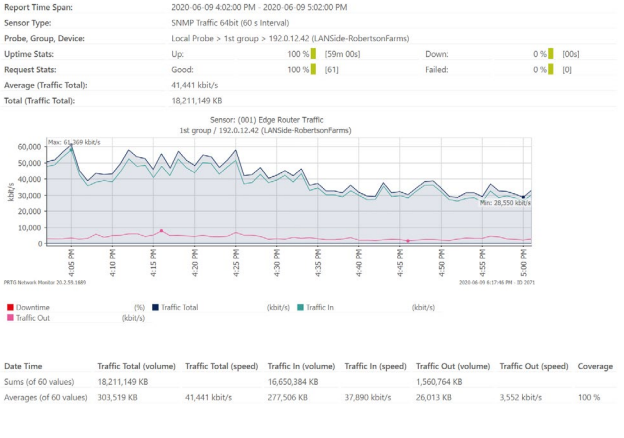
- Day 1 (2020-06-07 from 20:05 PM to 21:05 PM)



- Day 2 (2020-06-08 from 9:04 AM to 10:04 AM)

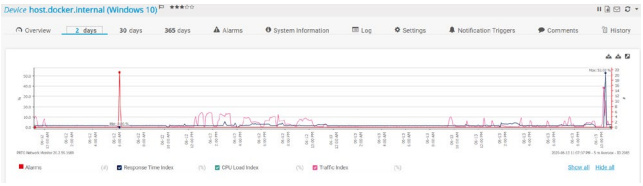


- Day 3 (2020-06-09-4-02PM to 5-02 PM)



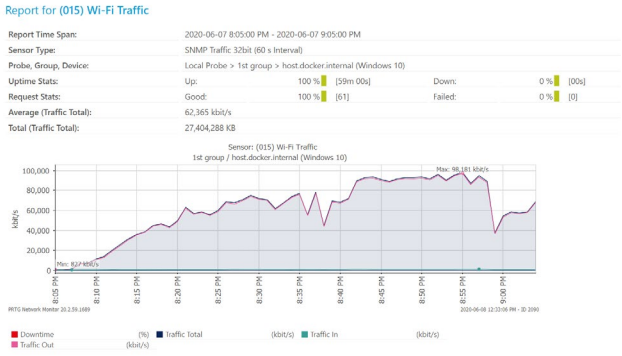
.5.3 host.docker.internal (Windows 10)

Let us look at overall monitoring over 2 days for this platform:



PRTG gives very friendly reports of monitoring with several popular indicators, such as Response Time Index, Traffic Index and CPU load index.

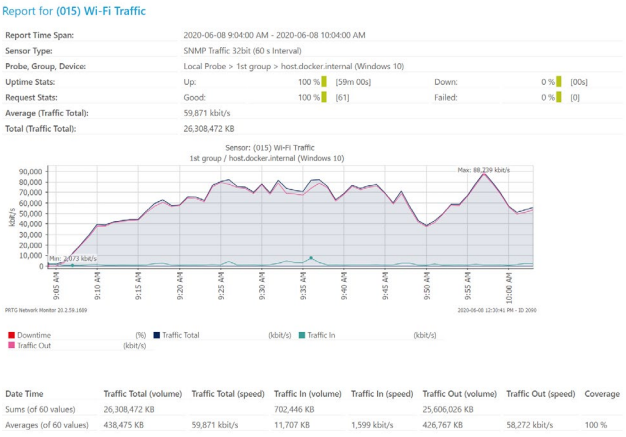
- Day 1 (2020-06-07 from 20:05 PM to 21:05 PM)



Report Time Span:	2020-06-07 8:05:00 PM - 2020-06-07 9:05:00 PM
Sensor Type:	SNMP Traffic 32bit (60 s Interval)
Probe, Group, Device:	Local Probe > 1st group > host.docker.internal (Windows 10)
Uptime Stats:	Up: 100 % [59m 00s] Down: 0 % [00s]
Request Stats:	Good: 100 % [61] Failed: 0 % [0]
Average (Traffic Total):	62,365 kbit/s
Total (Traffic Total):	27,404,288 KB

Date Time	Traffic Total (volume)	Traffic Total (speed)	Traffic In (volume)	Traffic In (speed)	Traffic Out (volume)	Traffic Out (speed)	Coverage
Sums (of 60 values)	27,404,288 KB		401,400 KB		27,002,888 KB		
Averages (of 60 values)	456,738 KB	62,365 kbit/s	6,690 KB	913 kbit/s	450,048 KB	61,452 kbit/s	100 %

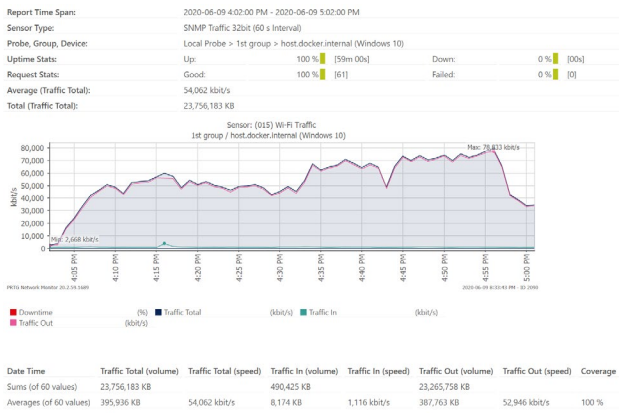
- Day 2 (2020-06-08 from 9:04 AM to 10:04 AM)



Report Time Span:	2020-06-08 9:04:00 AM - 2020-06-08 10:04:00 AM
Sensor Type:	SNMP Traffic 32bit (60 s Interval)
Probe, Group, Device:	Local Probe > 1st group > host.docker.internal (Windows 10)
Uptime Stats:	Up: 100 % [59m 00s] Down: 0 % [00s]
Request Stats:	Good: 100 % [61] Failed: 0 % [0]
Average (Traffic Total):	59,871 kbit/s
Total (Traffic Total):	26,308,472 KB

Date Time	Traffic Total (volume)	Traffic Total (speed)	Traffic In (volume)	Traffic In (speed)	Traffic Out (volume)	Traffic Out (speed)	Coverage
Sums (of 60 values)	26,308,472 KB		702,446 KB		25,606,026 KB		
Averages (of 60 values)	438,475 KB	59,871 kbit/s	11,707 KB	1,599 kbit/s	426,767 KB	58,272 kbit/s	100 %

- Day 3 (2020-06-09-4-02PM to 5-02 PM)

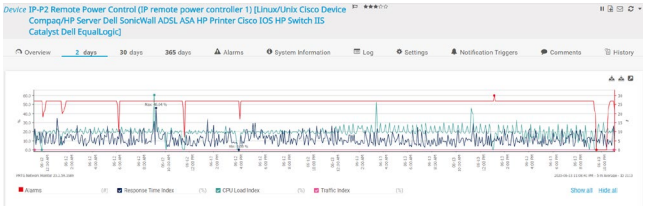


Report Time Span:	2020-06-09 4:02:00 PM - 2020-06-09 5:02:00 PM
Sensor Type:	SNMP Traffic 32bit (60 s Interval)
Probe, Group, Device:	Local Probe > 1st group > host.docker.internal (Windows 10)
Uptime Stats:	Up: 100 % [59m 00s] Down: 0 % [00s]
Request Stats:	Good: 100 % [61] Failed: 0 % [0]
Average (Traffic Total):	54,062 kbit/s
Total (Traffic Total):	23,756,183 KB

Date Time	Traffic Total (volume)	Traffic Total (speed)	Traffic In (volume)	Traffic In (speed)	Traffic Out (volume)	Traffic Out (speed)	Coverage
Sums (of 60 values)	23,756,183 KB		490,425 KB		23,265,758 KB		
Averages (of 60 values)	395,936 KB	54,062 kbit/s	8,174 KB	1,116 kbit/s	387,763 KB	52,946 kbit/s	100 %

.5.4 IP-P2 Remote Power Control

Let us look at overall monitoring over 2 days for this platform:



PRTG gives very friendly reports of monitoring with several popular indicators, such as Response Time Index, Traffic Index and CPU load index.

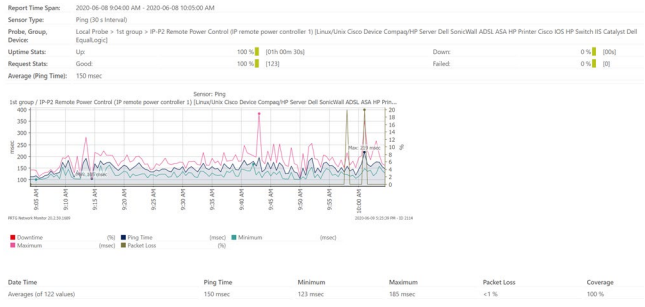
This device seems to be a power controller as suggested by Wiki, which has self contained web services. It is not quite stable during the testing period day by day. So, we can see the RTT varied a lot from time to time.

- Day 1 (2020-06-07 from 20:05 PM to 21:05 PM)

Report Time Span:	2020-06-07 8:05:00 PM - 2020-06-07 9:05:00 PM
Sensor Type:	Ping (30 s Interval)
Probe, Group, Device:	Local Probe > 1st group > IP-P2 Remote Power Control (IP remote power controller 1) [Linux/Unix Cisco Device Compaq/HP Server Dell SonicWall ADSL ASA HP Printer Cisco IOS HP Switch IIS Catalyst Dell EqualLogic]
Uptime Stats:	Up: 100 % [59m 30s] Down: 0 % [00s]
Request Stats:	Good: 100 % [121] Failed: 0 % [0]
Average (Ping Time):	157 msec

Date Time	Ping Time	Minimum	Maximum	Packet Loss	Coverage
Averages (of 120 values)	157 msec	128 msec	194 msec	<1 %	100 %

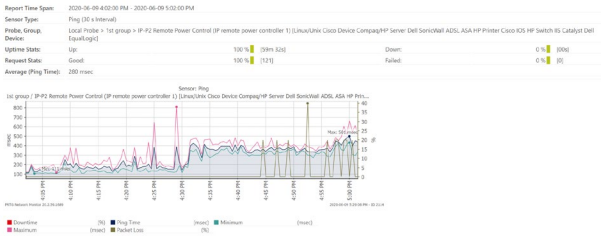
- Day 2 (2020-06-08 from 9:04 AM to 10:04 AM)



Report Time Span:	2020-06-08 9:04:00 AM - 2020-06-08 10:05:00 AM
Sensor Type:	Ping (30 s Interval)
Probe, Group, Device:	Local Probe > 1st group > IP-P2 Remote Power Control (IP remote power controller 1) [Linux/Unix Cisco Device Compaq/HP Server Dell SonicWall ADSL ASA HP Printer Cisco IOS HP Switch IIS Catalyst Dell EqualLogic]
Uptime Stats:	Up: 100 % [01h 00m 30s] Down: 0 % [00s]
Request Stats:	Good: 100 % [123] Failed: 0 % [0]
Average (Ping Time):	150 msec

Date Time	Ping Time	Minimum	Maximum	Packet Loss	Coverage
Averages (of 122 values)	150 msec	123 msec	185 msec	<1 %	100 %

- Day 3 (2020-06-09-4-02PM to 5-02 PM)



.5.5 imap-mail.outlook.com (Outlook Mailbox)

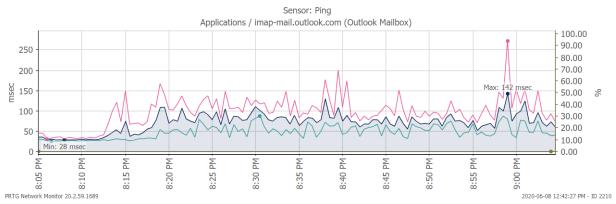
For the application Outlook, PRTG provides Response time and Mailbox size measurement upon I provided credentials of my email box. The one interesting thing I found from chart is that email box size changed all the time.

Let us take a quick look at the overall traffic of several days on the Outlook application:

- Day 1 (2020-06-07 from 20:05 PM to 21:05 PM)

Report for Ping

Report Time Span:	2020-06-07 8:05:00 PM - 2020-06-07 9:05:00 PM				
Sensor Type:	Ping (30 s Interval)				
Probe, Group, Device:	Local Probe > Applications > imap-mail.outlook.com (Outlook Mailbox)				
Uptime Stats:	Up:	100 %	[59m 30s]	Down:	0 % [0s]
Request Stats:	Good:	100 %	[121]	Failed:	0 % [0]
Average (Ping Time):	71 msec				



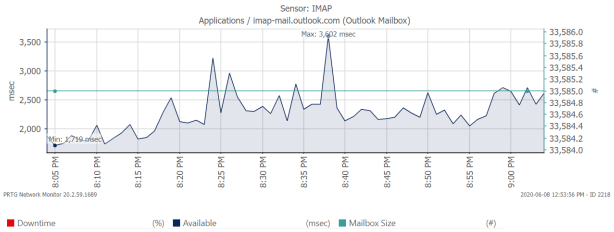
Downtime (%) Ping Time (msec) Minimum (msec)
Maximum (msec) Packet Loss (%)

Date Time	Ping Time	Minimum	Maximum	Packet Loss	Coverage
Averages (of 120 values)	71 msec	49 msec	99 msec	0 %	100 %

Interesting thing is PRTG can automatically decide what suitable graph can work with that protocol. In this case, PRTG can track emailbox size in the real time through SMTP protocol

Report for IMAP

Report Time Span:	2020-06-07 8:04:00 PM - 2020-06-07 9:05:00 PM				
Sensor Type:	IMAP (60 s Interval)				
Probe, Group, Device:	Local Probe > Applications > imap-mail.outlook.com (Outlook Mailbox)				
Uptime Stats:	Up:	100 %	[01h 00m 01s]	Down:	0 % [0s]
Request Stats:	Good:	100 %	[62]	Failed:	0 % [0]
Average (Available):	2,291 msec				



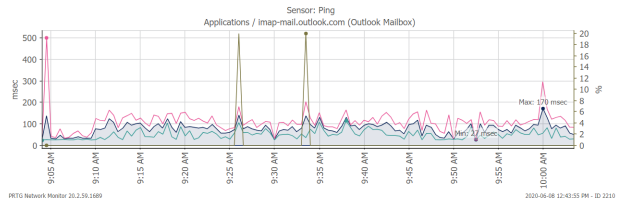
Downtime (%) Available (msec) Mailbox Size (#)

Date Time	Available	Mailbox Size	Coverage
Averages (of 61 values)	2,291 msec	33,585 #	100 %

- Day 2 (2020-06-08 from 9:04 AM to 10:04 AM)

Report for Ping

Report Time Span:	2020-06-08 9:04:00 AM - 2020-06-08 10:04:00 AM				
Sensor Type:	Ping (30 s Interval)				
Probe, Group, Device:	Local Probe > Applications > imap-mail.outlook.com (Outlook Mailbox)				
Uptime Stats:	Up:	100 %	[59m 30s]	Down:	0 % [0s]
Request Stats:	Good:	100 %	[121]	Failed:	0 % [0]
Average (Ping Time):	79 msec				



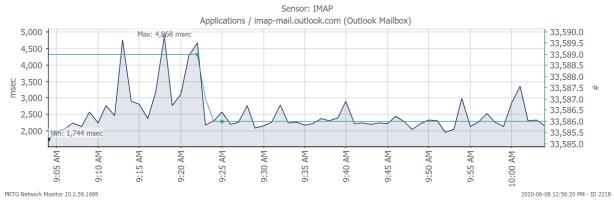
Downtime (%) Ping Time (msec) Minimum (msec)
Maximum (msec) Packet Loss (%)

Date Time	Ping Time	Minimum	Maximum	Packet Loss	Coverage
Averages (of 120 values)	79 msec	50 msec	111 msec	<1 %	100 %

PRTG can track emailbox size in the real time through SMTP protocol

Report for IMAP

Report Time Span:	2020-06-08 9:04:00 AM - 2020-06-08 10:05:00 AM				
Sensor Type:	IMAP (60 s Interval)				
Probe, Group, Device:	Local Probe > Applications > imap-mail.outlook.com (Outlook Mailbox)				
Uptime Stats:	Up:	100 %	[01h 00m 00s]	Down:	0 % [0s]
Request Stats:	Good:	100 %	[62]	Failed:	0 % [0]
Average (Available):	2,534 msec				

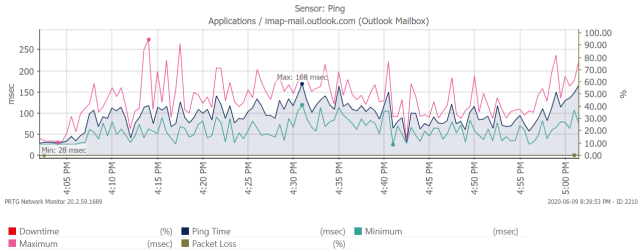


Downtime (%) Available (msec) Mailbox Size (#)

Date Time	Available	Mailbox Size	Coverage
Averages (of 61 values)	2,534 msec	33,587 #	100 %

- Day 3 (2020-06-09-4-02PM to 5-02 PM)

Report Time Span:	2020-06-09 4:02:00 PM - 2020-06-09 5:02:00 PM
Sensor Type:	Ping (30 s Interval)
Probe, Group, Device:	Local Probe > Applications > imap-mail.outlook.com (Outlook Mailbox)
Uptime Stats:	Up: 100 % [59m 31s] Down: 0 % [00s]
Request Stats:	Good: 100 % [121] Failed: 0 % [0]
Average (Ping Time):	95 msec

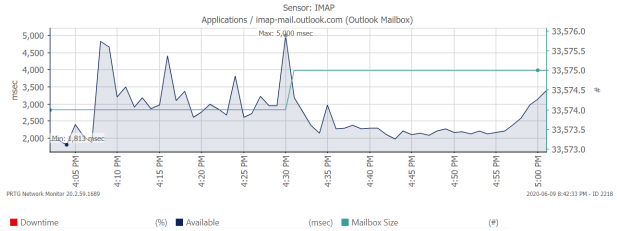


Downtime (%) Ping Time (msec) Minimum (msec)
Maximum (msec) Packet Loss (%)

Date Time	Ping Time	Minimum	Maximum	Packet Loss	Coverage
Averages (of 120 values)	95 msec	58 msec	135 msec	0 %	100 %

PRTG can track emailbox size in the real time through SMTP protocol

Report Time Span:	2020-06-09 4:02:00 PM - 2020-06-09 5:02:00 PM
Sensor Type:	IMAP (60 s Interval)
Probe, Group, Device:	Local Probe > Applications > imap-mail.outlook.com (Outlook Mailbox)
Uptime Stats:	Up: 100 % [59m 02s] Down: 0 % [00s]
Request Stats:	Good: 100 % [61] Failed: 0 % [0]
Average (Available):	2,729 msec



Downtime (%) Available (msec) Mailbox Size (#)

Date Time	Available	Mailbox Size	Coverage
Averages (of 60 values)	2,729 msec	33,575 #	100 %

.5.6 Remote MySQL DB

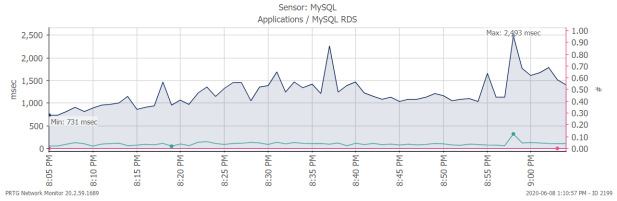
Regarding about database application, I issued pseudo SQL code against my Remote MySQL DB constantly with certain interval of time. in order to evaluate execution performance.

Let us give the first glance at the overall traffic during recent days of this MySQL DB application:

- Day 1 (2020-06-07 from 20:05 PM to 21:05 PM)

Report for MySQL

Report Time Span:	2020-06-07 8:05:00 PM - 2020-06-07 9:05:00 PM
Sensor Type:	MySQL v2 (60 s Interval)
Probe, Group, Device:	Local Probe > Applications > MySQL RDS
Uptime Stats:	Up: 100 % [59m 01s] Down: 0 % [00s]
Request Stats:	Good: 100 % [61] Failed: 0 % [0]
Average (Execution Time):	1,255 msec



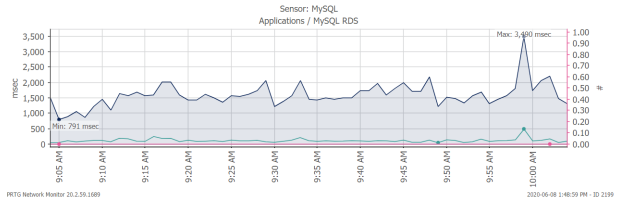
Downtime (%) Execution Time (msec) Query Execution Time (msec)
Affected Rows (#)

Date Time	Execution Time	Query Execution Time	Affected Rows	Coverage
Averages (of 60 values)	1,255 msec	101 msec	0 #	100 %

- Day 2 (2020-06-08 from 9:04 AM to 10:04 AM)

Report for MySQL

Report Time Span:	2020-06-08 9:04:00 AM - 2020-06-08 10:05:00 AM
Sensor Type:	MySQL v2 (60 s Interval)
Probe, Group, Device:	Local Probe > Applications > MySQL RDS
Uptime Stats:	Up: 100 % [01h 00m 00s] Down: 0 % [00s]
Request Stats:	Good: 100 % [62] Failed: 0 % [0]
Average (Execution Time):	1,583 msec

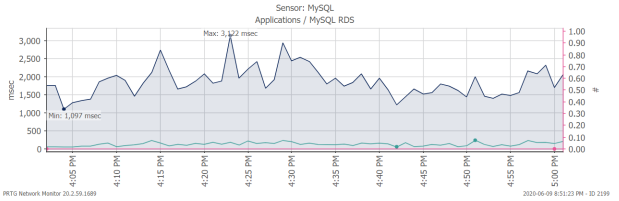


Downtime (%) Execution Time (msec) Query Execution Time (msec)
Affected Rows (#)

Date Time	Execution Time	Query Execution Time	Affected Rows	Coverage
Averages (of 61 values)	1,583 msec	114 msec	0 #	100 %

- Day 3 (2020-06-09-4-02PM to 5-02 PM)

Report Time Span:	2020-06-09 4:02:00 PM - 2020-06-09 5:02:00 PM
Sensor Type:	MySQL v2 (60 s Interval)
Probe, Group, Device:	Local Probe > Applications > MySQL RDS
Uptime Stats:	Up: 100 % [58m 00s] Down: 0 % [00s]
Request Stats:	Good: 100 % [60] Failed: 0 % [0]
Average (Execution Time):	1,862 msec

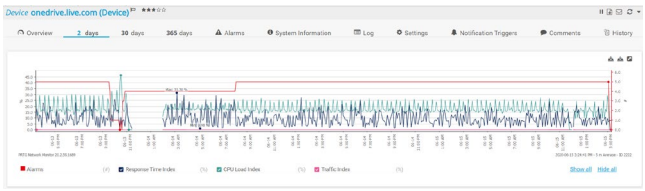


Downtime (%) Execution Time (msec) Query Execution Time (msec)
Affected Rows (#)

Date Time	Execution Time	Query Execution Time	Affected Rows	Coverage
Averages (of 59 values)	1,862 msec	131 msec	0 #	100 %

.5.7 Microsoft OneDrive

OneDrive is another application I was monitoring because it is one of the most used cloud drives, I used so far. We can see the red line indicates traffic during the recent several days from the screenshot below:

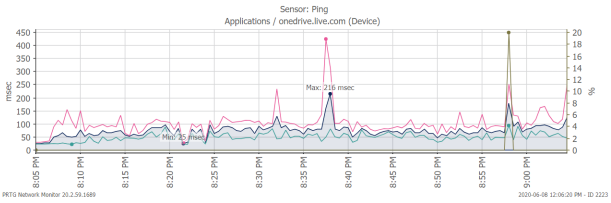


- Day 1 (2020-06-07 from 20:05 PM to 21:05 PM)

For the application cloud drive, PRTG provides Ping and cloud drive size measurement

Report for Ping

Report Time Span:	2020-06-07 8:05:00 PM - 2020-06-07 9:05:00 PM				
Sensor Type:	Ping (30 s Interval)				
Probe, Group, Device:	Local Probe > Applications > onedrive.live.com (Device)				
Uptime Stats:	Up:	100 %	[59m 30s]	Down:	0 % [00s]
Request Stats:	Good:	100 %	[121]	Failed:	0 % [0]
Average (Ping Time):	75 msec				



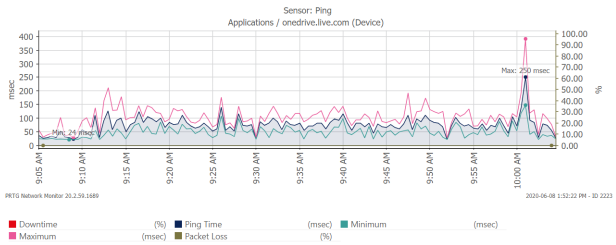
Date Time	Ping Time	Minimum	Maximum	Packet Loss	Coverage
Averages (of 120 values)	75 msec	50 msec	105 msec	<1 %	100 %

Interesting thing is cloud size can also be measured overtime. However the size changes is quite tiny within one hour. So it is not good idea to report here for comparsion.

- Day 2 (2020-06-08 from 9:04 AM to 10:04 AM)

Report for Ping

Report Time Span:	2020-06-08 9:05:00 AM - 2020-06-08 10:05:00 AM				
Sensor Type:	Ping (30 s Interval)				
Probe, Group, Device:	Local Probe > Applications > onedrive.live.com (Device)				
Uptime Stats:	Up:	100 %	[59m 30s]	Down:	0 % [00s]
Request Stats:	Good:	100 %	[121]	Failed:	0 % [0]
Average (Ping Time):	75 msec				

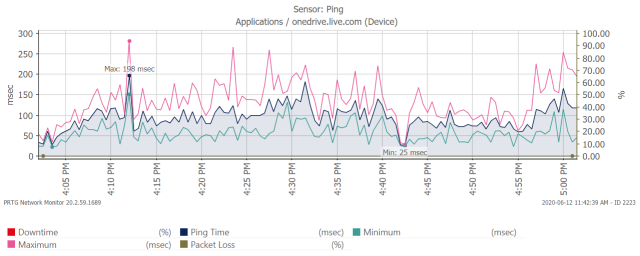


Date Time	Ping Time	Minimum	Maximum	Packet Loss	Coverage
Averages (of 120 values)	75 msec	50 msec	105 msec	0 %	100 %

- Day 3 (2020-06-09-4-02PM to 5-02 PM)

Report for Ping

Report Time Span:	2020-06-09 4:02:00 PM - 2020-06-09 5:02:00 PM				
Sensor Type:	Ping (30 s Interval)				
Probe, Group, Device:	Local Probe > Applications > onedrive.live.com (Device)				
Uptime Stats:	Up:	100 %	[59m 31s]	Down:	0 % [00s]
Request Stats:	Good:	100 %	[121]	Failed:	0 % [0]
Average (Ping Time):	94 msec				



Date Time	Ping Time	Minimum	Maximum	Packet Loss	Coverage
Averages (of 120 values)	94 msec	57 msec	134 msec	0 %	100 %

From above traffic, ping, and application dedicated measurement charts, we can easily identify their own pattern. Like NAS drive traffic shows almost the same curve during one hour of different days. And MySQL DB performs very constantly day by day from the curve of graph. LANSide-RobertsonFarms represents different curve from Host.Docker.Windows in term of traffic of their major network interface**.** IP-P2 Remote Power Control has the most variant status compared with other devices, because it was automatically discovered by Cacti / PRTG Poller**. Therefor we can distinguish one device/ application from another based on the PRTG monitoring.**

5 Question 2: Data Capture Wireshark / Argus Server

The second question can be answered with question 1 together, because Wireshark worked in parallel with the Cacti, as the packets filter, monitor, collector and analyzer tool.

5.1 Wireshark UI / Columns customization

- Wireshark can provide out-of-box default columns for capturing the data:

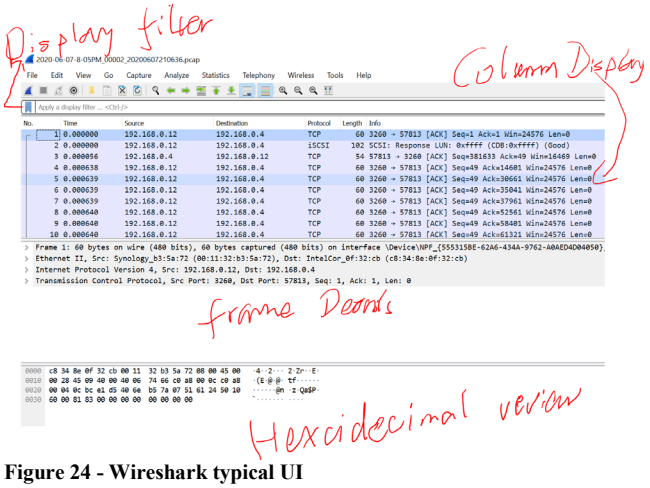


Figure 24 - Wireshark typical UI

Wireshark's default columns are [9]:

1. No. -Frame number from the beginning of the pcap. The first frame is always 1.

2. Time – Seconds broken down to the nanosecond from the first frame of the pcap. The first frame is always 0.000000.
3. Source – Source address, commonly an IPv4, IPv6, or Ethernet address.
4. Destination – Destination address, commonly an IPv4, IPv6, or Ethernet address.
5. Protocol – Protocol used in the Ethernet frame, IP packet, or TCP segment (ARP, DNS, TCP, HTTP, etc.).
6. Length – Length of the frame in bytes.

- What I needs extra for wireshark:

To be better to analyze the data, we might need extra columns for our convienent. Such as:

1. Source port
2. Desintation port
3. Timestamp
4. HTTP / HTTPS host
5. SNMP OID and Values

Remove unneccessary columns:

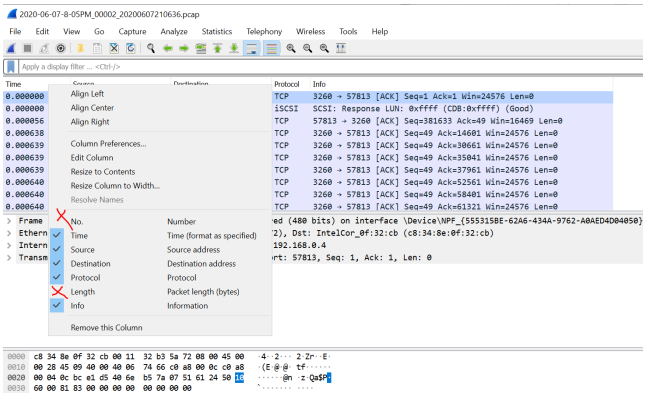


Figure 25 -Wireshark UI Customization

Adding wanted columns:

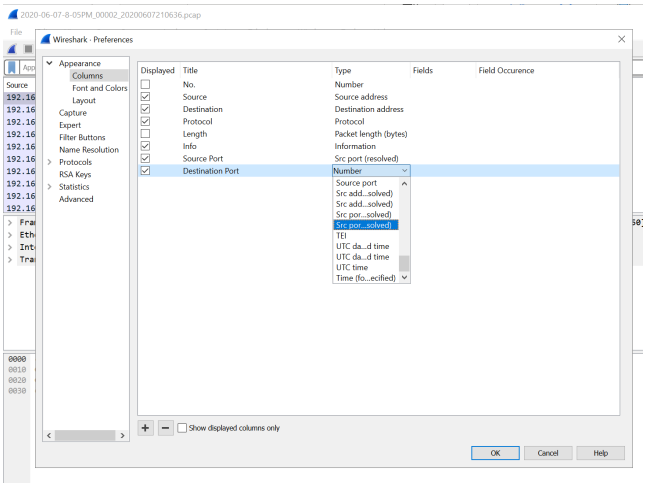


Figure 26 - Wireshark UI Customization

Adding custom columns (SNMP OID and values) as we need:

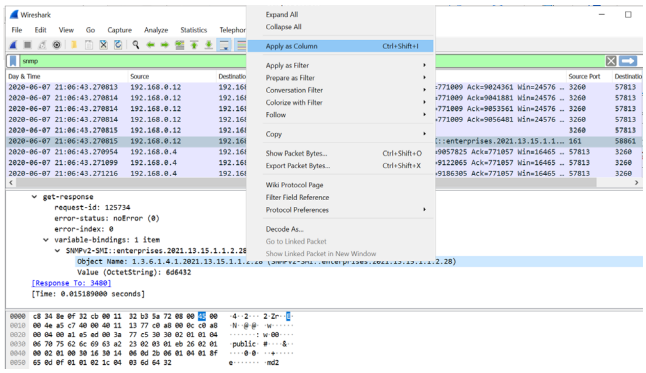


Figure 27 - Wireshark UI Customization

Customization result:

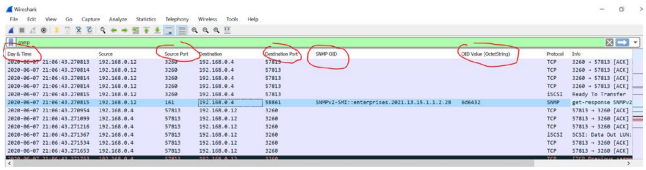


Figure 28 - Wireshark UI Customization Results

The screenshot above shows I successfully removed unnessaary columns, add my needed cusomized column SNMP OID and values. They can be prepared for data analytics in the later section.

5.2 Wireshark Monitoring

Once we have customized UI setup, we can start up a little bit simple data analytics (We will talk about it specifically in the later section).

For example, if we want to identify the suspicious devices which joined in our network during the monitoring period. We can track DHCP activitity.

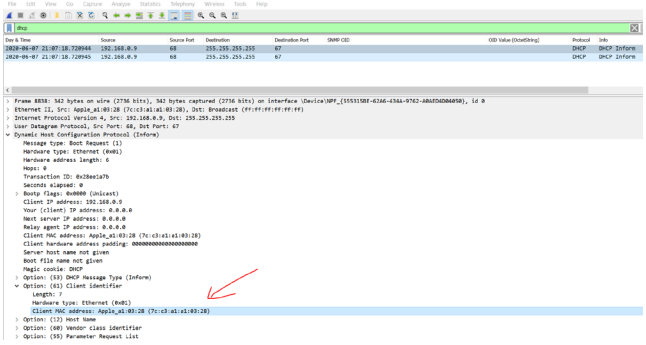


Figure 29 - Wireshark Monitoring

From the screenshot above, we firstly filter the keyword “dhcp” or “bootp” to find out any DHCP activity. Then we navigated to detail section to find “client identifier”. In the result, we can find out new joined device MAC address and its name.

5.3 Wireshark Data Capturing

I was utilizing the Wireshark command line instead of GUI, because its notorious long term capture memorey freezing issue. I was executing the follow command to capture the packets:

```
D:\wireshark>dumpcap -i 10 -b duration:3600 -b files:1 -w D:\wireshark\2020-06-07-8-05PM\2020-06-07-8-05PM.pcap
Capturing on 'Wi-Fi'
File: D:\wireshark\2020-06-07-8-05PM\2020-06-07-8-05PM_00001_20200607200635.pcap
Packets: 4034779 File: D:\wireshark\2020-06-07-8-05PM\2020-06-07-8-05PM_00002_20200607210636.pcap
Packets captured: 4052561
Packets received/dropped on interface 'Wi-Fi': 4052561/32372 (pcap:32372/dumpcap:0/flushed:0/ps_ifdrop:0) (99.2%)
```

Figure 30 - Wireshark CLI execution

D:\wireshark>dumpcap -i 3 -q -b duration:3600 -b files:1 -f "host 192.168.0.4" -w d:\wireshark\2020-06-07-6:50pm.pcap

I use the *duration* keyword to specify a length of time (in seconds) to spend filling each file (for example, one hour, or 3600 seconds). And I specify file number as 1. Moreover, I specify the filter that I tell WireShark to capture the traffic to or from the host "192.168.0.4" which is my Major Internet Connection interface. The example above shows how I can instruct **dumpcap** to maintain a rotating record of the last 24 hours worth of traffic for the major internet connection interface.

Alternatively, we can utilize the Argus Server and Client to collect traffic. For example:

On Server Side, we execute "sudo argus -P 561 -d":

```
mininet@mininet-vm:/$ sudo argus -P 561 -d
mininet@mininet-vm:/$ argus[5430]: started
argus[5431]: connect from 127.0.0.1
argus[5480]: client done.
argus[5431]: connect from 127.0.0.1
argus[5498]: client done.
```

Figure 31 - Argus Sever Setup

On Client Side, we can execute "sudo ratop -S localhost:561 > argus.data":

```
mininet@mininet-vm:~$ sudo ratop -S localhost:561
   14 06-11-20 23:35:28.691377  s      tcp 192.168.157.128.42628      <-> 192.168.247.130.6633      2      0
   15 06-11-20 23:35:28.691525  s      tcp 192.168.157.128.42630      <-> 192.168.247.130.6633      2      0
   16 06-11-20 23:35:28.691724  s      tcp 192.168.157.128.42632      <-> 192.168.247.130.6633      2      0
   17 06-11-20 23:35:30.690674  s      tcp 192.168.157.128.42634      <-> 192.168.247.130.6633      2      0
   18 06-11-20 23:35:30.690905  s      tcp 192.168.157.128.42636      <-> 192.168.247.130.6633      2      0
   19 06-11-20 23:35:30.691086  s      tcp 192.168.157.128.42638      <-> 192.168.247.130.6633      2      0
   20 06-11-20 23:35:30.691263  s      tcp 192.168.157.128.42640      <-> 192.168.247.130.6633      2      0
   21 06-11-20 23:35:23.693488          tcp 192.168.247.130.6633       ?> 192.168.157.128.42522      1      0
   22 06-11-20 23:35:23.694108          tcp 192.168.247.130.6633       ?> 192.168.157.128.42528      1      0
   23 06-11-20 23:35:23.694146          tcp 192.168.247.130.6633       ?> 192.168.157.128.42526      1      0
   24 06-11-20 23:35:23.694150          tcp 192.168.247.130.6633       ?> 192.168.157.128.42524      1      0
   25 06-11-20 23:35:25.692853          tcp 192.168.247.130.6633       ?> 192.168.157.128.42532      1      0
   26 06-11-20 23:35:25.692899          tcp 192.168.247.130.6633       ?> 192.168.157.128.42530      1      0
   27 06-11-20 23:35:25.692904          tcp 192.168.247.130.6633       ?> 192.168.157.128.42536      1      0
   28 06-11-20 23:35:25.692909          tcp 192.168.247.130.6633       ?> 192.168.157.128.42534      1      0
   29 06-11-20 23:35:27.693525          tcp 192.168.247.130.6633       ?> 192.168.157.128.42542      1      0
   30 06-11-20 23:35:27.693581          tcp 192.168.247.130.6633       ?> 192.168.157.128.42544      1      0
   31 06-11-20 23:35:27.694262          tcp 192.168.247.130.6633       ?> 192.168.157.128.42538      1      0
   32 06-11-20 23:35:27.694316          tcp 192.168.247.130.6633       ?> 192.168.157.128.42540      1      0
   33 06-11-20 23:35:29.692846          tcp 192.168.247.130.6633       ?> 192.168.157.128.42546      1      0
   34 06-11-20 23:35:29.692893          tcp 192.168.247.130.6633       ?> 192.168.157.128.42548      1      0
   35 06-11-20 23:35:29.693894          tcp 192.168.247.130.6633       ?> 192.168.157.128.42552      1      0
   36 06-11-20 23:35:29.693940          tcp 192.168.247.130.6633       ?> 192.168.157.128.42550      1      0
   37 06-11-20 23:35:31.694808          tcp 192.168.247.130.6633       ?> 192.168.157.128.42560      1      0
   38 06-11-20 23:35:31.692897          tcp 192.168.247.130.6633       ?> 192.168.157.128.42554      1      0
   39 06-11-20 23:35:31.693417          tcp 192.168.247.130.6633       ?> 192.168.157.128.42558      1      0
```

Figure 32 - Argus Client Call

I uploaded three times .pcap files into OneDrive, because the file size is quite big. Who reviews this document can double check those. https://1drv.ms/u/s!AidJC3PD0CHZndd--8yHxoD-LqRbCw?e=mQF9PJ [15]

- 2020-06-07-8-05PM_00002_20200607210636.pcap
- 2020-06-08-9-04AM_00002_20200608100431.pcap
- 2020-06-09-4-02PM_00002_20200609170231.pcap

With the PCAP data sets, we can go further to do a lot of configure, filtering, and analyzing. I will go through it in the next section.

6 Question 3: Data Analytics and Visualization

6.1 Data preliminary analysis

First, we can utilize "**tshark**" and "**tcpdump**" to check PCAP file a little to discover some hidden information.

Check SNMP port 161 traffic NOT contain "192.168.0.4" which is our main poller host, so that we can find any suspicious activities over my network:

```
tcpdump -r 2020-06-07-8-05PM_00002_20200607210636.pcap -n src port 161 | grep -v 192.168.0.4 | more
```

```
chris@DESKTOP-I7H08DT:/mnt/d/wireshark/2020-06-07-8-05PM$ tcpdump -r 2020-06-07-8-05PM_00002_20200607210636.pcap -n src
port 161 | grep -v 192.168.0.4 | more
reading from file 2020-06-07-8-05PM_00002_20200607210636.pcap, link-type EN10MB (Ethernet)
```

Figure 33 - Port and Host reverse filtering

No suspicious activities detected. Then Check any traffic NOT includes "192.168.0.4":

```
tcpdump -r 2020-06-07-8-05PM_00002_20200607210636.pcap | grep  -v 192.168.0.4 | more
```

```
chris@DESKTOP-I7H08DT:/mnt/d/wireshark/2020-06-07-8-05PM$ tcpdump -r 2020-06-07-8-05PM_00002_20200607210636.pcap  | grep
 -v 192.168.0.4 | more
reading from file 2020-06-07-8-05PM_00002_20200607210636.pcap, link-type EN10MB (Ethernet)
21:06:35.725095 IP 192.168.0.12.iscsi-target > host.docker.internal.57813: Flags [.], ack 122773796, win 24576, length 0
21:06:35.725095 IP 192.168.0.12.iscsi-target > host.docker.internal.57813: Flags [P.], seq 0:48, ack 1, win 24576, lengt
h 48
21:06:35.725151 IP host.docker.internal.57813 > 192.168.0.12.iscsi-target: Flags [.], ack 48, win 16469, length 0
21:06:35.725733 IP 192.168.0.12.iscsi-target > host.docker.internal.57813: Flags [.], ack 14601, win 24576, length 0
21:06:35.725734 IP 192.168.0.12.iscsi-target > host.docker.internal.57813: Flags [.], ack 30661, win 24576, length 0
21:06:35.725734 IP 192.168.0.12.iscsi-target > host.docker.internal.57813: Flags [.], ack 35041, win 24576, length 0
21:06:35.725734 IP 192.168.0.12.iscsi-target > host.docker.internal.57813: Flags [.], ack 37961, win 24576, length 0
```

Figure 34 - Host reverse filtering

The screenshot shows that most of activities NOT involve "192.168.0.4" are ISCSI disk exchange activity.

We also can run the command "tshark -r 2020-06-07-8-05PM_00002_20200607210636.pcap -q -z hosts" to get a host list for the security check later on in the later section.

```
chris@DESKTOP-I7H08DT:/mnt/d/wireshark/2020-06-07-8-05PM$ tshark -r 2020-06-07-8-05PM_00002_20200607210636.pcap -q -z hosts
# TShark hosts output

# Host data gathered from 2020-06-07-8-05PM_00002_20200607210636.pcap

34.240.253.134  autoscaling.us-west-2.amazonaws.com
34.239.55.242   autoscaling.eu-central-1.amazonaws.com
34.240.225.112  autoscaling.ap-northeast-1.amazonaws.com
34.239.30.25    us-east-1-console.aws.amazon.com
52.96.88.178    YQB-efz.ms-acdc.office.com
54.239.96.170   ec2.ap-northeast-1.amazonaws.com
18.189.78.164   ec2-18-189-78-164.us-east-2.compute.amazonaws.com
40.126.7.100    www.tm.a.prd.aadg.akadns.net
40.100.162.178  YQB-efz.ms-acdc.office.com
34.240.206.120  autoscaling.ap-southeast-2.amazonaws.com
54.239.29.8     ec2.amazonaws.com
192.168.0.12    chris.local
13.82.185.0     ec2.ap-southeast-2.amazonaws.com
176.32.118.53   ec2.us-west-1.amazonaws.com
52.46.146.93    ec2.amazonaws.com
54.239.31.83    us-east-1.console.aws.amazon.com
52.95.84.21     ec2.ap-south-1.amazonaws.com
52.96.88.146    YQB-efz.ms-acdc.office.com
34.240.206.131  autoscaling.ap-southeast-2.amazonaws.com
40.100.162.18   YQB-efz.ms-acdc.office.com
54.239.55.167   ec2.eu-central-1.amazonaws.com
52.95.121.170   autoscaling.eu-west-1.amazonaws.com
40.126.7.98     www.tm.a.prd.aadg.akadns.net
54.239.31.45    autoscaling.us-east-1.amazonaws.com
13.107.42.13    1-0004.1-msedge.net
52.95.34.25     autoscaling.ap-northeast-1.amazonaws.com
52.95.35.29     ec2.ap-southeast-1.amazonaws.com
52.95.193.162   autoscaling.ap-northeast-2.amazonaws.com
40.100.162.194  YQB-efz.ms-acdc.office.com
176.32.118.39   ec2.us-west-1.amazonaws.com
54.239.55.239   autoscaling.eu-central-1.amazonaws.com
54.239.39.181   autoscaling.eu-west-1.amazonaws.com
40.126.7.97     www.tm.a.prd.aadg.akadns.net
34.240.249.157  ec2.us-west-2.amazonaws.com
177.72.244.236  autoscaling.sa-east-1.amazonaws.com
40.126.7.103    www.tm.a.prd.aadg.akadns.net
52.96.51.98     YQB-efz.ms-acdc.office.com
176.32.118.221  autoscaling.us-west-1.amazonaws.com
34.240.195.243  ec2.ap-southeast-2.amazonaws.com
52.95.84.98     autoscaling.ap-south-1.amazonaws.com
52.95.18.87     autoscaling.us-east-2.amazonaws.com
40.126.7.66     www.tm.a.prd.aadg.akadns.net
52.96.88.162    YQB-efz.ms-acdc.office.com
52.95.35.192    autoscaling.ap-southeast-1.amazonaws.com
40.126.7.102    www.tm.a.prd.aadg.akadns.net
177.72.245.178  ec2.sa-east-1.amazonaws.com
52.167.249.196  settingsfd-geo.trafficmanager.net
40.100.163.178  YQB-efz.ms-acdc.office.com
177.72.245.184  ec2.sa-east-1.amazonaws.com
176.32.118.214  autoscaling.us-west-1.amazonaws.com
54.239.39.138   eu-west-1.ec2.amazonaws.com
52.95.35.161    autoscaling.ap-southeast-1.amazonaws.com
40.126.7.65     www.tm.a.prd.aadg.akadns.net
52.95.35.57     ec2.ap-southeast-1.amazonaws.com
54.239.96.159   ec2.ap-northeast-1.amazonaws.com
52.95.193.80    ec2.ap-northeast-2.amazonaws.com
52.95.20.2      ec2.us-east-2.amazonaws.com
40.126.7.101    www.tm.a.prd.aadg.akadns.net
52.96.88.138    YQB-efz.ms-acdc.office.com
40.100.162.2    YQB-efz.ms-acdc.office.com
172.217.11.174  play.google.com
40.126.7.64     www.tm.a.prd.aadg.akadns.net
```

Figure 35 - capture IP list for whitelist later on

From the sceenshot above, we can see "amazon.com" and "office.com" are actively work as bots during the investiation.

Racluster is one of Argus client tools to be used for aggregate, filter and analyze raw data. It can also filter out Argus data by specifying certain conditions.

```
sudo racluster -R 2020-06-07-805PM-00002-20200607210636.argus -nu -- net ! 192.168.0.4/24
```

```
user@securityonion:~/Document$ sudo racluster -R 2020-06-07-8-05PM_00002_20200607210636.argus -nu -- net ! 192.168.0.4/24
        StartTime  Proto           SrcAddr  Sport  Dir           DstAddr  Dport  TotPkts     SrcBytes     DstBytes
 1591574833.033539    udp     192.168.100.3.52152    ->    239.255.255.250.1900        16         5708            0
 1591574840.024287    udp  fe80::b7:5e1e:196*.50145    ->           ff02::1:3.5355         4          420            0
 1591574844.590117    udp  fe80::b7:5e1e:196*.5353     ->           ff02::fb.5353          2          182            0
 1591574844.591295    udp  fe80::b7:5e1e:196*.59155    ->           ff02::1:3.5355         2          170            0
 1591574795.727711     ip  00:11:32:b3:5a:72           ->  c8:34:8e:0f:32:cb          2128    117694816            0
```

Figure 36 - Racluster finds suspicious IP

The screenshot and command show I was trying to find any net address NOT equal to my host IP. It can help me to find out suspicious network connections.

6.2 Data Pre-processing

From the generated data (especially by WireShark), I utilized Argus command line to convert PCAP files into ARGUS files firstly, so that we can prepare for the data analytics. For example:

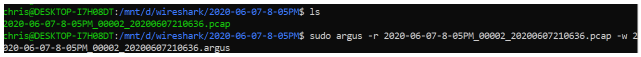


Figure 37 - Argus read PCAP from WireShark

chris@DESKTOP-I7H08DT:/mnt/d/wireshark/2020-06-07-8-05PM$ sudo argus -r 2020-06-07-8-05PM_00002_20200607210636.pcap -w 2020-06-07-8-05PM_00002_20200607210636.argus

Furthermore, we can consider about utilizing the RA command filter and fields parameters to generate customized statistical results:

Specify the fields to print. Ra uses a default printing field list, by specifying a field you can replace this list completely, or you can modify the existing default print list, using the optional '-' and '+[#]' form of the command. The available fields to print are [17]:

startime, lasttime, count, dur, avgdur,
saddr, daddr, proto, sport, dport, ipid,
stos, dtos, sttl, dttl, bytes, sbytes, dbytes,
pkts, spkts, dpkts, load, loss, rate,
srcid, ind, mac, dir, jitter, status, user,
win, trans, seq, vlan, mpls

I was doing this process for all the records:

- **Convert Wireshark PCAP data into ARGUS**

chris@DESKTOP-I7H08DT:/mnt/d/wireshark/2020-06-07-8-05PM$ sudo argus -r 2020-06-07-8-05PM_00002_20200607210636.pcap -w 2020-06-07-8-05PM_00002_20200607210636.argus

- **Specify the Label for the ARGUS data**

Rlabel_assignment2.conf:

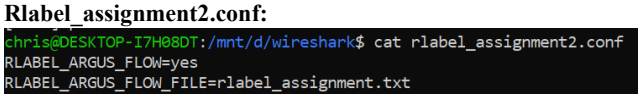


Figure 38 - Specify Rlabel configuration file

Rlabel_assignment2.txt

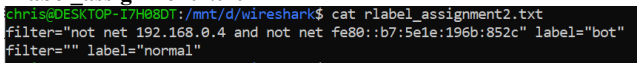


Figure 39 - Ralabel configure content

Using CONF and Filter TXT

ralabel -f rlabel_assignment2.conf -r 2020-06-07-8-05PM/2020-06-07-8-05PM_00002_2020 -w 2020-06-07-8-05PM/2020-06-07-8-05PM_00002_20200607210636_labeled.argus

- **Customize the format of output and save into CSV file**

ra -c , -r 2020-06-07-8-05PM/2020-06-07-8-05PM_00002_20200607210636_labeled.argus -s +label:20 +load +count + dur +mac +win + trans + mpls +rate +stos + dtos +sttl +dttl > 2020-06-07-8-05PM/2020-06-07-8-05PM_00002_20200607210636_labed.csv

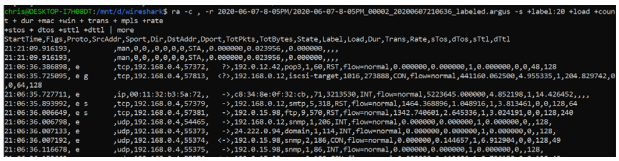


Figure 40 - generate CSV with customized filter and columns

Then I was planning to visualize the results based on the data analyzing of Argus. The processed Argus files were converted into CSV formatted files, and they were uploaded to OneDrive:

https://1drv.ms/u/s!AidJC3PD0CHZndd--8yHxoD-LqRbCw?e=mQF9PJ [15]

- 2020-06-07-8-05PM_00002_20200607210636.csv
- 2020-06-08-9-04AM_00002_20200608100431.csv
- 2020-06-09-4-02PM_00002_20200609170231.csv

After collecting all three days data, and processed by Argus, we can go to the next step to visualize the data to make the data more understandable and eventually transform the data into information to the user.

Here I want to exploit the power of Python preprocess the data and feed preprocessed data into Tableau, which is one of the most popular data visualization tools on the market, chart to compare three days traffic. And furthermore, we even can break it down into more subtle comparison among the applications and devices. which is attached in this assignment. We notice the data combined with several major features:

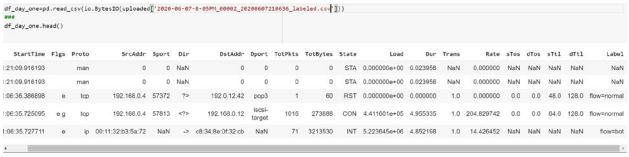


Figure 41 - Python Pandas Load Raw CSV data

The screenshot above shows the datasets finally includes following fields:

Index(['**StartTime**', '**Flgs**', '**Proto**', '**SrcAddr**', '**Sport**', '**Dir**', '**DstAddr**', '**Dport**', '**TotPkts**', '**TotBytes**', '**State**' , '**Load**', '**Dur**', '**Trans**', '**Rate**', '**sTos**', '**dTos**', '**sTtl**', '**dTtl**', '**Label**'],
dtype='object')

We noticed there is “label” field. It can be used for classification. There are NA values in the dataframe, we need to cleanup the data firstly, I did it via Python Pandas Lib. There are several data cleanup processes, like dropna(), drop unnecessary columns etc.

```
# remove NA values
df_day_one.dropna()
# remove unnecessary columns, or the columns have too many NA vlues
df_day_one = df_day_one.drop(['Flgs', 'Sport', 'Dir', 'State', 'StartTime', 'Load', 'Dur', 'Trans', 'Rate',
       'sTos', 'dTos', 'sTtl', 'dTtl'], axis=1)
# remove abnormal address field
df_day_one = df_day_one[df_day_one.SrcAddr != '0']
df_day_one
```

	Proto	SrcAddr	DstAddr	Dport	TotPkts	TotBytes	Label
2	tcp	192.168.0.4	192.0.12.42	pop3	1	60	flow=normal
3	tcp	192.168.0.4	192.168.0.12	iscsi-target	1016	273888	flow=normal
4	ip	00:11:32:b3:5a:72	c8:34:8e:0f:32:cb	NaN	71	3213530	flow=bot
5	tcp	192.168.0.4	192.168.0.12	smtp	5	318	flow=normal
6	tcp	192.168.0.4	192.0.15.98	ftp	9	570	flow=normal
...	...	...	...	...	...	...	...
2199	tcp	192.168.0.4	177.72.244.236	https	19	7034	flow=normal
2200	tcp	192.168.0.4	23.2.5.87	https	2	123	flow=normal
2201	icmp	192.168.0.4	24.222.0.95	0x04f1	1	139	flow=normal
2202	udp	192.168.0.4	192.168.0.12	snmp	2	431	flow=normal
2203	tcp	192.168.0.4	13.107.42.13	https	9	5225	flow=normal

2200 rows × 7 columns

Figure 42 - Pandas Preprocess Data

6.3 Data Visualization

6.3.1 Ragraph

It is worth to mention at the beginning, Ragraph provides the visualization features out of box of Argus. However, there are many other more visualization can perform better and easier. So in this assignment I will facilitate Python + Tableau to deal with the Argus data source. I applied several common usages of "Ragraph" as below:

- **Day 1: 2020-06-7**

```
user@securityonion:~/Documents$ sudo ragraph pkts dport -M 10s -r 2020-06-07-8-0
5PM_00002_20200607210636.argus -title "2020-06-07-8-05PM Total Load on ports" -w
 2020-06-07-8-05PM-total-load-port.png
user@securityonion:~/Documents$ sudo ragraph pkts dport -M 10s -r 2020-06-07-8-0
5PM_00002_20200607210636.argus -srcid 192.168.0.4 -title "2020-06-07-8-05PM on t
he host" -w 2020-06-07-8-05PM-host.png
user@securityonion:~/Documents$ sudo ragraph pkts dport -M 10s -r 2020-06-07-8-0
5PM_00002_20200607210636.argus -srcid ! 192.168.0.4 -title "2020-06-07-8-05PM on
 the host" -w 2020-06-07-8-05PM-bots.png
user@securityonion:~/Documents$ ls
2020-06-07-8-05PM_00002_20200607210636.argus  2020-06-07-8-05PM-total-load.png       ragraph.png
2020-06-07-8-05PM-bots.png                    2020-06-07-8-05PM-total-load-port.png
2020-06-07-8-05PM-host.png                    ragraph
user@securityonion:~/Documents$
```

Figure 43 - Ragraph data collection

Overall load traffic by stacked ports:

sudo ragraph pkts dport -M 10s -r 2020-06-07-8-05PM_00002_20200607210636.argus -title "2020-06-07-8-05PM Total Load on ports" -w 2020-06-07-8-05PM-total-load-port.png

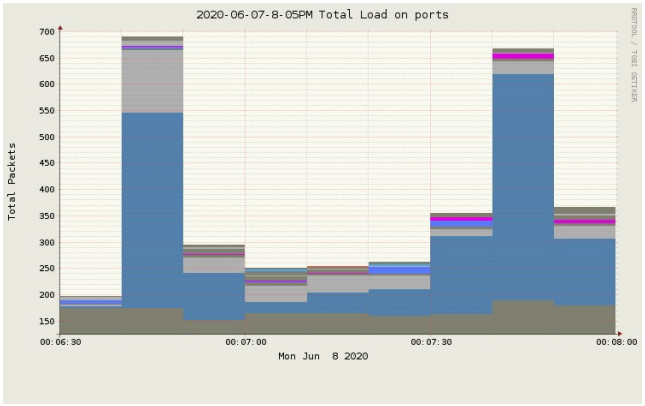


Figure 44 - Traffic visualization by Ragraph Day 1

load traffic by SNMP port (161) including udp and tcp on "192.168.0.4" only. It provides two filters to let us identify how much SNMP traffic inside:

- **net 192.168.0.4**
- **port 161**

sudo ragraph pkts dport -M 10s -r 2020-06-09-4-02PM_00002_20200609170231.argus -title "2020-06-09-4-02PM on the host with SNMP" -w 2020-06-09-4-02PM-host.png - net 192.168.0.4 and port 161

Filter with host 192.168.0.4 and SNMP traffic only

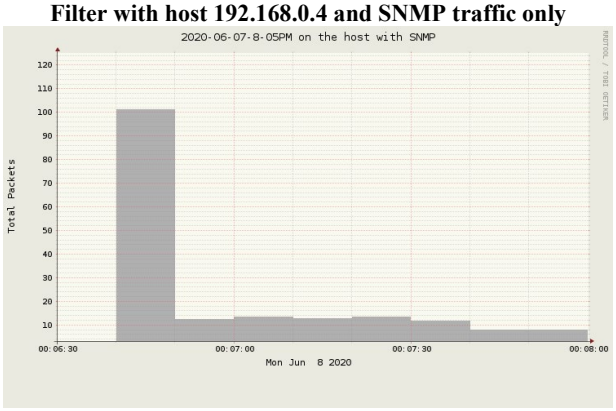


Figure 45 – Day 1 Ragraph with RA filter

We can see the two pictures are almost the same, because host 192.168.0.4 traffic dominate overall traffic.

- Day 2 (2020-06-08 from 9:04 AM to 10:04 AM)

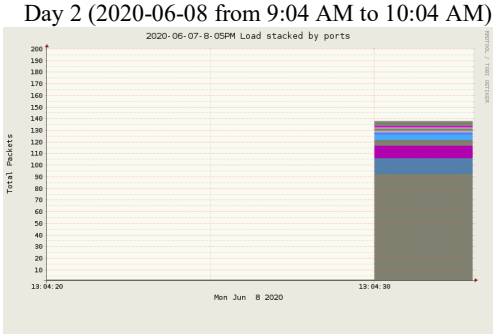


Figure 46 - Traffic visualization by Ragraph Day 2

Filter with host 192.168.0.4 and SNMP traffic only

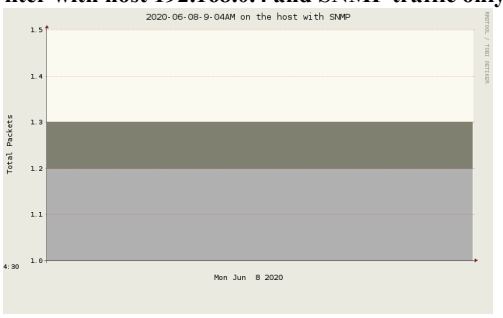
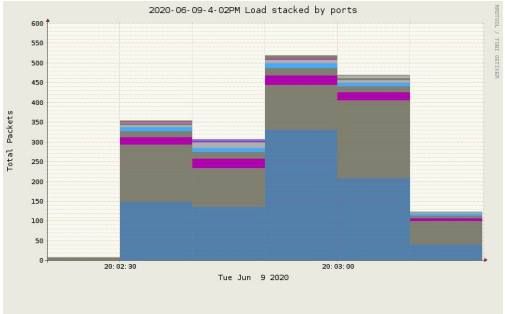


Figure 47 - Day 2 Ragraph with RA filter

- Day 3 (2020-06-09-4-02PM to 5-02 PM)



Filter with host 192.168.0.4 and SNMP traffic only

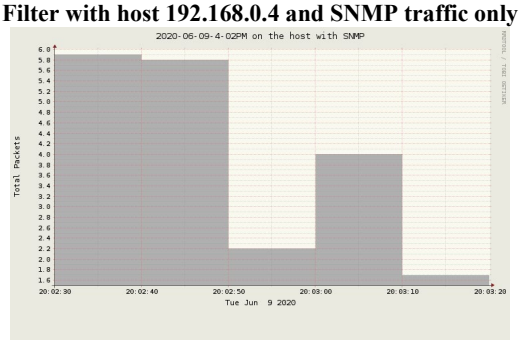


Figure 48 - Day 3 Ragraph with RA filter

From above, we can see SNMP traffic does not weight too much within overall traffic. It is because most of traffic is caused by iSCSI protocol transcations between host and NAS drive. Next let us analyze data with Python Matplotlib with more subtle customizations.

6.3.2 Python Matplotlib

Here we also want to practise a little bit Python Matplotlib Visualization following Lab 5 instructions:

- **Day 1: 2020-06-7**

Labels stacked bar chart:

```
# Plot a histogram of flows
# Similar to rahisto

plt.hist([df_day_one[df_day_one['Label']=='flow=Normal']['TotPkts'], df_day_one[df_day_one['Label']=='flow=bot']['TotPkts']],log=True ,
         stacked=True, color=['blue','red'],bins = np.arange(0,150,10), label=['Normal','Bot'])
plt.ylabel('Number of flows')
plt.xlabel('Number of packets')
plt.legend()
plt.show()
```

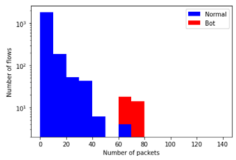


Figure 49 - Stacked bar by labelled Day 1

Top 10 IP address by traffic:

```
# source: https://meyob.github.io/blog/2017/network-trace-analysis/
# similar to rasort

f, ax = plt.subplots(1, 1,figsize=(6, 4))

per_ip = df_day_one.groupby("DstAddr", as_index=False).agg({"TotBytes":[np.sum]})

p = per_ip.sort_values(by=("TotBytes","sum"),ascending=False)[:10]
p["TotBytes","sum"] = p["TotBytes","sum"]/1000000
sns.barplot(p["DstAddr"], p["TotBytes","sum"],palette="Spectral_r", ax=ax)

ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
ax.set_ylabel("Traffic volume (MB)")
ax.set_title("Top 10 IP addresses by traffic volume")
```

```
Text(0.5, 1.0, 'Top 10 IP addresses by traffic volume')
```

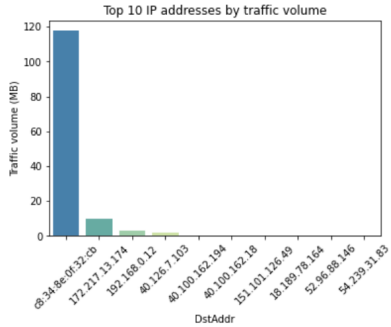


Figure 50 - Top 10 traffic ranking by IP Day 1

Ranking by protocol:

```
df1 = df_day_one.sample(frac=0.1, random_state=10)
ax = box_plot_df_setup(df1['Dport'], df1['SrcAddr']).T.plot(kind='bar', stacked=True)
plt.ylabel('Total # of connections')
plt.xlabel('Source IP')
patches, labels = ax.get_legend_handles_labels()
ax.legend(patches, labels, bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0., title="Port")
```

```
<matplotlib.legend.Legend at 0x7efe15eb99b0>
```

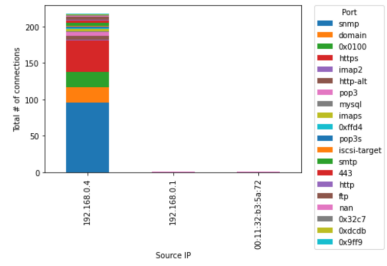


Figure 51 - Stacked Bar with Ports Day 1

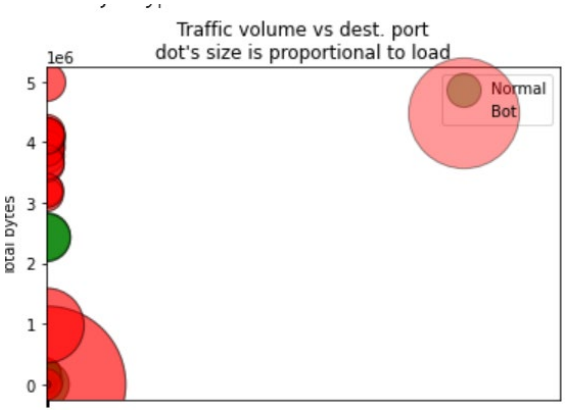


Figure 52 - Labelled Traffic by Ports Day 1

- **Day 2: 2020-06-8**

We will do the same visualization for day 2 and day 3:

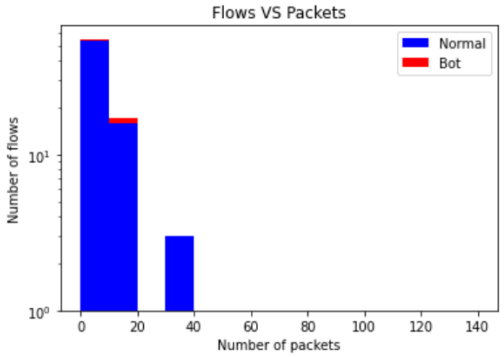


Figure 53 -Stacked bar by labelled Day 2

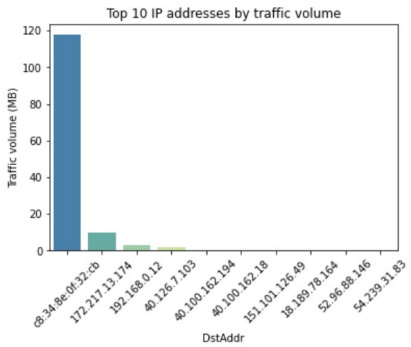


Figure 54 - Top 10 traffic ranking by IP Day 2

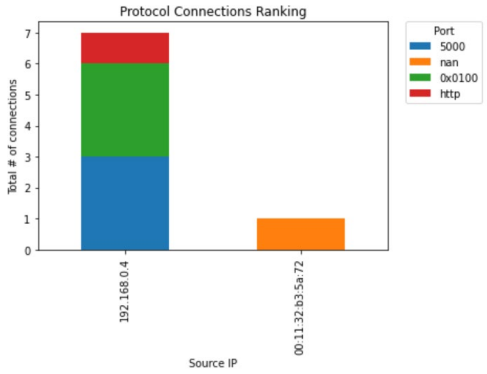


Figure 55 - Stacked Bar with Ports Day 2

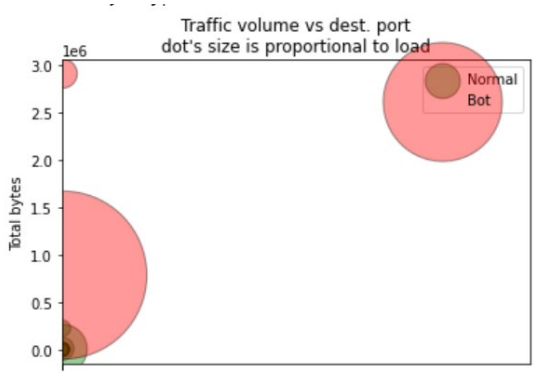


Figure 56 - Labelled Traffic by Ports Day 2

- **Day 3: 2020-06-9**

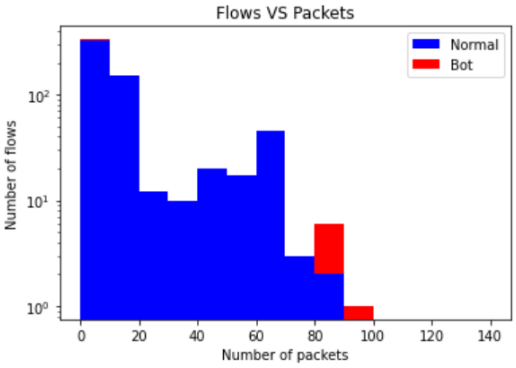


Figure 57 - Stacked bar by labelled Day 3

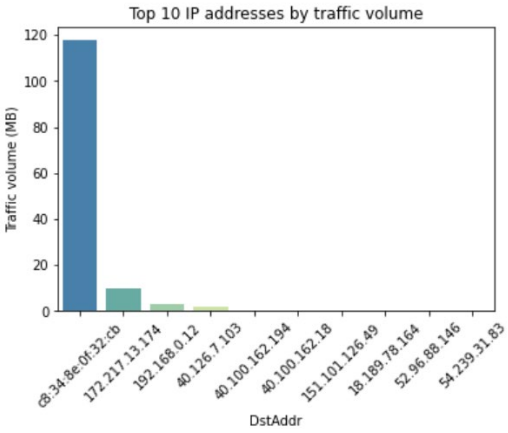


Figure 58 - Top 10 traffic ranking by IP Day 3

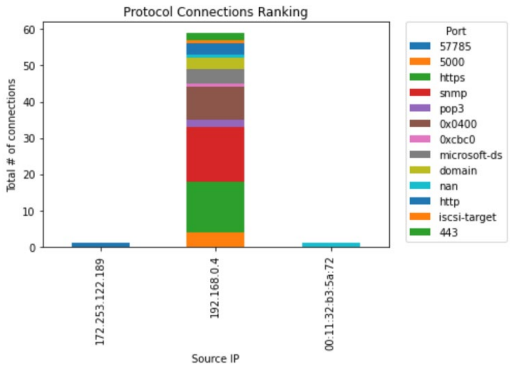


Figure 59 - Stacked Bar with Ports Day 3

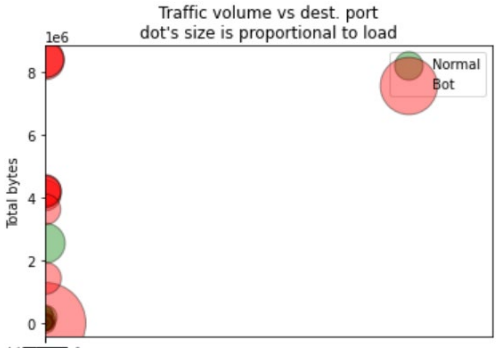


Figure 60 - Labelled Traffic by Ports Day 3

The result we can draw from the Python Charts comparsion of three days:

- Similarity: they all have variant port types for traffic; c8:34:8e:0f:32:cb interface receives and sends most of traffic; There always some unkown traffic “bot” appearing traffics each day;
- Dissimilarity: Day 2 has the least traffic, and interfaces activities were very low compared to two others days. Day 1 SNMP port has the most traffic and has the most connections for each protocol. Day 2 traffic over different ports has biggest variation, while Day 3 traffic over different ports looks more balanced.

We want to see overall staticstic viualization. In this case, I merged three days Data together.

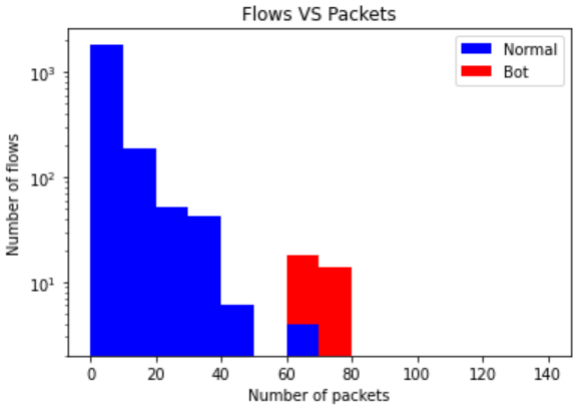


Figure 61 - Stacked bar by labelled Overall

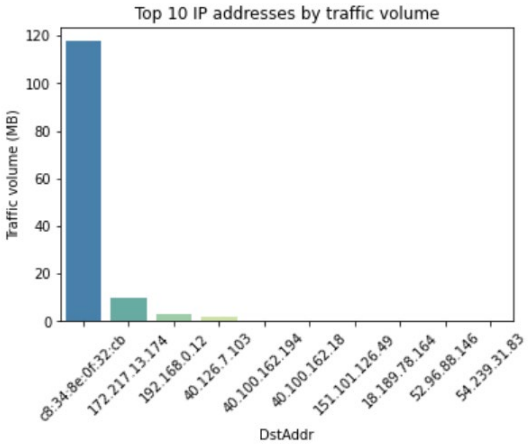


Figure 62 - Top 10 traffic ranking by IP Overall

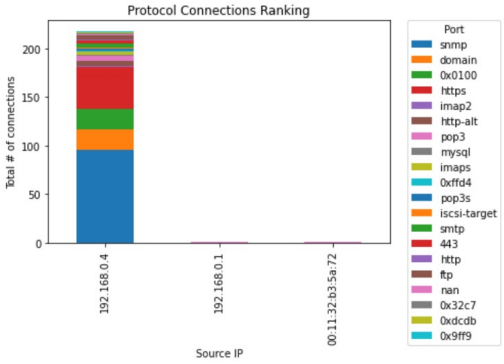


Figure 63 - Stacked Bar with Ports Overall

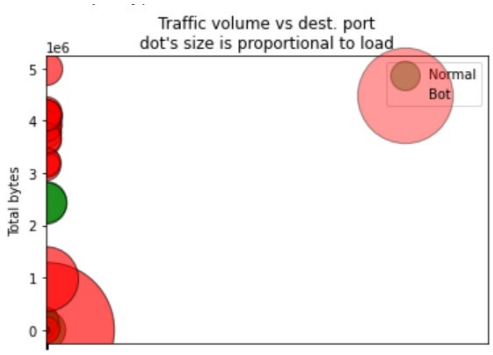


Figure 64 - Labelled Traffic by Ports Overall

Next I want to go steps further to make visualization more richful and interactive. Therefore, I conducted visualization charts based on the same network datasets with Tableau, which is popular data science visualizaiton tools.

6.3.3 Tableau

Because Python visualization is overkilling for this assignment, I will exploit the power of Tableau to visualize the results with an interative way. Therefore, we are going to feed those cleansed data to Tableau [13]. Here I only posted a brief list of graphs to give a quick glance of my data analytics based on Argus. For more rich interactive visualized data analtyics, I published it on Tableau public portal: https://public.tableau.com/profile/chris.gang.liu#!/vizhome/Assignment2-DataAnalytics/Dashboard1. [14]
Please feel free to check it.

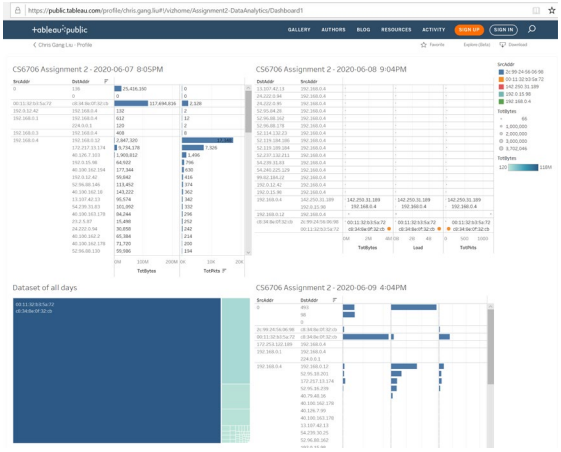


Figure 65 - My Network Data dashboard created upon Tableau

We can conclude several things:

- From c8:34:8e:0f:32:cb to 00:11:32:b3:5a:72 dominated traffic flows, because Synology NAS works as SCSI drive to my PC.
- 192.168.0.4 communicating with 192.168.0.12 is second largest traffic
- Other traffics from 192.168.0.4 can be seen in the summarized chart
- Few other traffics which does not involve the 192.168.0.4 become neglected compared with 3 types of traffic mentioned above

6.4 Data Analytics and Classification

Regarding about data analytics from our Wireshark and Argus Datasets, we can have many things to do depending on our initial purposes. Such as: statistics, network security detection etc.

In our assignment, we are focusing on the network monitoring, Moreover, I will bring some network security concerns by our datasets.

6.4.1 Argus Client Analytics – Racluster, Racounter, Rasort, rafilteraddr and Rahisto

Racluster is able to get session data from flow records. I applied it for each day's record, then uploaded shared folder in case reviewer wants to check it. The result likes screenshot below:

```
chris@DESKTOP-I7H08DT:/mnt/d/wireshark$ racluster -r 2020-06-07-8-05PM/2020-06-07-8-05PM_00002_20200607210636.argus -w 2020-06-07-8-05PM/2020-06-07-8-05PM_00002_20200607210636_clustered.argus
```

Figure 66 - Racluster usages on Datasets

Racounter provides very convientent way to generate statistic analytics results through command line. Here I want to compared three days results by reading the Argus:

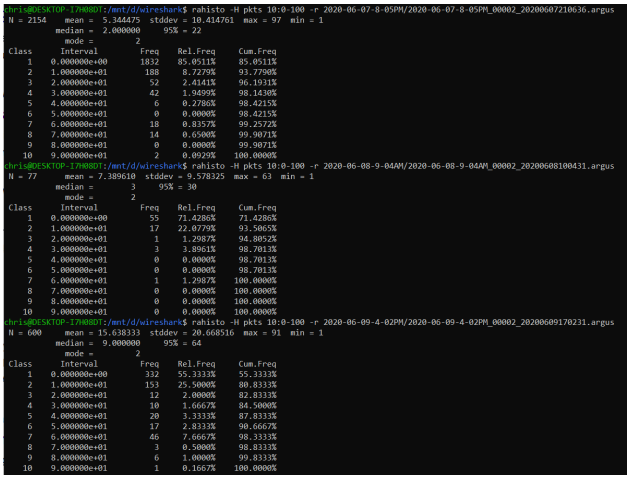
```
chris@DESKTOP-I7H08DT:/mnt/d/wireshark$ racount -r 2020-06-07-8-05PM/2020-06-07-8-05PM_00002_20200607210636.argus - tcp and retrans
racount   records     total_pkts     src_pkts      dst_pkts      total_bytes       src_bytes         dst_bytes
    sum   289         2288           1166          1122          544104            109392            434712
chris@DESKTOP-I7H08DT:/mnt/d/wireshark$ racount -r 2020-06-08-9-04AM/2020-06-08-9-04AM_00002_20200608100431.argus - tcp and retrans
racount   records     total_pkts     src_pkts      dst_pkts      total_bytes       src_bytes         dst_bytes
    sum   9           98             27            43            19774             2522              16252
chris@DESKTOP-I7H08DT:/mnt/d/wireshark$ racount -r 2020-06-09-4-02PM/2020-06-09-4-02PM_00002_20200609170231.argus - tcp and retrans
racount   records     total_pkts     src_pkts      dst_pkts      total_bytes       src_bytes         dst_bytes
    sum   73          6827           1634          5193          1803814           501482            1302332
```

Figure 67 - Racount usages on Datasets

By running "**racount -r 2020-06-09-4-02PM/2020-06-09-4-02PM_00002_20200609170231.argus - tcp and retrans "** , we can see the traffic varys day by by, the second day has the lowest traffic, which the third day raised it up dramatically.

Rahisto can generate frequency distribution table for user-selected metrics. Such as flow duration, src and dst port numbers, byte transfer, packet counts, average duration etc [19].

Same as above I run the command "**rahisto -H pkts 10:0-100 -r 2020-06-09-4-02PM/2020-06-09-4-02PM_00002_20200609170231.argus** " against three days flow data:



```
chris@DESKTOP-I7H08DT:/mnt/d/wireshark$ rahisto -H pkts 10:0-100 -r 2020-06-07-8-05PM/2020-06-07-8-05PM_00002_20200607210636.argus
N = 2154     mean =  5.344475  stddev = 10.414761  max = 97  min = 1
           median =  2.000000      95% = 22
             mode =  2
Class      Interval         Freq    Rel.Freq    Cum.Freq
    1  0.000000e+00         1832      85.0511%    85.0511%
    2  1.000000e+01          188       8.7279%    93.7790%
    3  2.000000e+01           52       2.4141%    96.1931%
    4  3.000000e+01           42       1.9499%    98.1430%
    5  4.000000e+01            6       0.2786%    98.4215%
    6  5.000000e+01            0       0.0000%    98.4215%
    7  6.000000e+01           18       0.8357%    99.2572%
    8  7.000000e+01           14       0.6500%    99.9071%
    9  8.000000e+01            0       0.0000%    99.9071%
   10  9.000000e+01            2       0.0929%   100.0000%
chris@DESKTOP-I7H08DT:/mnt/d/wireshark$ rahisto -H pkts 10:0-100 -r 2020-06-08-9-04AM/2020-06-08-9-04AM_00002_20200608100431.argus
N = 77       mean = 7.389610  stddev = 9.578325  max = 63  min = 1
           median =  3      95% = 30
             mode =  2
Class      Interval         Freq    Rel.Freq    Cum.Freq
    1  0.000000e+00           55      71.4286%    71.4286%
    2  1.000000e+01           17      22.0779%    93.5065%
    3  2.000000e+01            1       1.2987%    94.8052%
    4  3.000000e+01            3       3.8961%    98.7013%
    5  4.000000e+01            0       0.0000%    98.7013%
    6  5.000000e+01            0       0.0000%    98.7013%
    7  6.000000e+01            1       1.2987%   100.0000%
    8  7.000000e+01            0       0.0000%   100.0000%
    9  8.000000e+01            0       0.0000%   100.0000%
   10  9.000000e+01            0       0.0000%   100.0000%
chris@DESKTOP-I7H08DT:/mnt/d/wireshark$ rahisto -H pkts 10:0-100 -r 2020-06-09-4-02PM/2020-06-09-4-02PM_00002_20200609170231.argus
N = 600      mean = 15.638333  stddev = 20.668516  max = 91  min = 1
           median =  9.000000      95% = 64
             mode =  2
Class      Interval         Freq    Rel.Freq    Cum.Freq
    1  0.000000e+00          332      55.3333%    55.3333%
    2  1.000000e+01          153      25.5000%    80.8333%
    3  2.000000e+01           12       2.0000%    82.8333%
    4  3.000000e+01           10       1.6667%    84.5000%
    5  4.000000e+01           20       3.3333%    87.8333%
    6  5.000000e+01           17       2.8333%    90.6667%
    7  6.000000e+01           46       7.6667%    98.3333%
    8  7.000000e+01            3       0.5000%    98.8333%
    9  8.000000e+01            6       1.0000%    99.8333%
   10  9.000000e+01            1       0.1667%   100.0000%
```

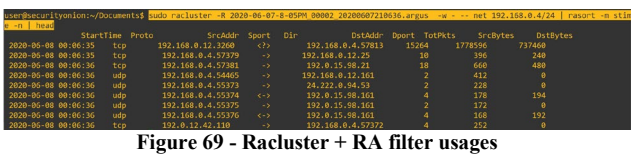
Figure 68 - Rahisto Usages over the flow data

We can see the result above: 2020-06-08 has the lowest standard deviation next to 2020-06-07, while its mean and median values are a little bit higher than 2020-06-07. On the other hand, 2020-06-09 has much higher standard deviation, as well as the midian and mean values. Therefore, we can image 2020-06-08 and 2020-06-07 traffic curve looks much smoother that 2020-06-09.

Now I want to define whitelist to identify unknown network traffic by the usages of combinaton of **rafilteraddr, Racluster, Racounter. Therefore, we can address potential network issues.**

Firest of all, we can utilize Racluster and Rasort start time to give a quick glance of traffic through main network interface along with time elapse:

```
sudo racluster -R 2020-06-07-8-05PM_00002_20200607210636.argus  -w - -- net 192.168.0.4/24 | rasort -m stime -n | head
```



```
user@securityonion:~/Documents$ sudo racluster -R 2020-06-07-8-05PM_00002_20200607210636.argus  -w - -- net 192.168.0.4/24 | rasort -m stime -n | head
                StartTime  Proto            SrcAddr  Sport  Dir          DstAddr  Dport  TotPkts     SrcBytes     DstBytes
2020-06-08 00:06:35    tcp       192.168.0.12.3260     <?>       192.168.0.4.57813     15264      1778596       737460
2020-06-08 00:06:35    tcp        192.168.0.4.57379     ->        192.168.0.12.25         10          396          240
2020-06-08 00:06:36    tcp        192.168.0.4.57381     ->         192.0.15.98.21         10          660          480
2020-06-08 00:06:36    udp        192.168.0.4.54465     ->       192.168.0.12.161          2          412            0
2020-06-08 00:06:36    udp        192.168.0.4.55373     ->         24.222.0.94.53          2          228            0
2020-06-08 00:06:36    udp        192.168.0.4.55374    <->         192.0.15.98.161          4          178          194
2020-06-08 00:06:36    udp        192.168.0.4.55375     ->         192.0.15.98.161          2          172            0
2020-06-08 00:06:36    udp        192.168.0.4.55376    <->         192.0.15.98.161          4          168          192
2020-06-08 00:06:36    tcp       192.0.12.42.110        ->       192.168.0.4.57372          4          252            0
```

Figure 69 - Racluster + RA filter usages

We can see from screenshot above, 192.168.0.4 was trying to send SNMP request to predefined SNMP client agents, which are normal. However the last "**192.0.12.42.110 -> 192.168.0.4.57372**" looks suspicious. Not only the port number is random, but also 192.0.12.42 is unknown IP over my network.

Next I want to have an approved whitelist of domain name and IP, then I can use it for identify unknown suscipious domain name and IP over my network.

The dataset well-known public websites I got is from https://www.kaggle.com/cheedcheed/top1m , which includes top 1 million famous domain recorded by Alexa. I converted it to text formatted whitelist file.

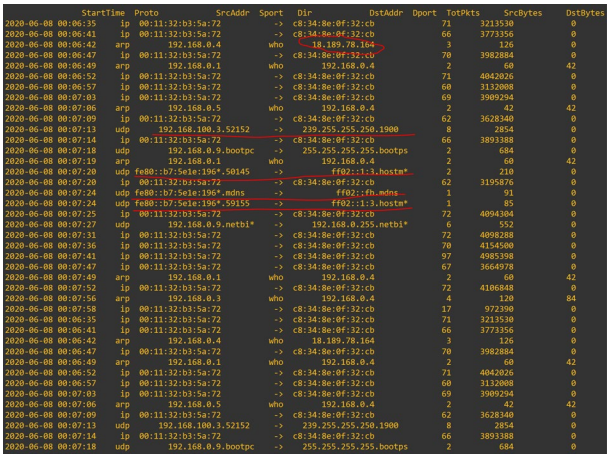
Adding host and known SNMP agents to whitelist.txt:

```
user@securityonion:~/Documents$
user@securityonion:~/Documents$ echo 192.168.0.4 >> whitelist.txt
user@securityonion:~/Documents$ echo 192.168.0.12 >> whitelist.txt
user@securityonion:~/Documents$ echo 192.0.15.98 >> whitelist.txt
user@securityonion:~/Documents$ echo 192.0.11.125 >> whitelist.txt
user@securityonion:~/Documents$ echo 192.0.12.42 >> whitelist.txt
user@securityonion:~/Documents$ echo 192.0.145.27 >> whitelist.txt
user@securityonion:~/Documents$
```

Figure 70 - Prepare whitelist for rafilteraddr

Reverse filter whitelist.txt to find unkown domain or IP:

```
sudo rafilteraddr -R 2020-06-07-8-05PM_00002_20200607210636.argus -v -f whitelist.txt
```



```
            StartTime  Proto            SrcAddr  Sport  Dir          DstAddr  Dport  TotPkts     SrcBytes     DstBytes
2020-06-08 00:06:35     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      71      3213530            0
2020-06-08 00:06:41     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      66      3773356            0
2020-06-08 00:06:42    arp        192.168.0.4         who        18.189.78.164       3          126            0
2020-06-08 00:06:47     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      70      3982884            0
2020-06-08 00:06:49    arp        192.168.0.1         who        192.168.0.4        2           60           42
2020-06-08 00:06:52     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      71      4042026            0
2020-06-08 00:06:57     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      60      3132008            0
2020-06-08 00:07:03     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      69      3909294            0
2020-06-08 00:07:06    arp        192.168.0.5         who        192.168.0.4        2           42           42
2020-06-08 00:07:09     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      62      3628340            0
2020-06-08 00:07:13    udp      192.168.100.3.52152     ->     239.255.255.250.1900       8         2854            0
2020-06-08 00:07:14     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      66      3893388            0
2020-06-08 00:07:18    udp        192.168.0.9.bootpc     ->     255.255.255.255.bootps       2          684            0
2020-06-08 00:07:19    arp        192.168.0.1         who        192.168.0.4        2           60           42
2020-06-08 00:07:20    udp  fe80::b7:5e1e:196*.50145     ->              ff02::1:3.hostm*       2          210            0
2020-06-08 00:07:20     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      62      3195876            0
2020-06-08 00:07:24    udp  fe80::b7:5e1e:196*.mdns      ->              ff02::fb.mdns       1           91            0
2020-06-08 00:07:24    udp  fe80::b7:5e1e:196*.59155     ->              ff02::1:3.hostm*       1           85            0
2020-06-08 00:07:25     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      72      4094304            0
2020-06-08 00:07:27    udp        192.168.0.9.netbi*     ->      192.168.0.255.netbi*       6          552            0
2020-06-08 00:07:31     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      72      4098288            0
2020-06-08 00:07:36     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      70      4154500            0
2020-06-08 00:07:41     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      97      4985398            0
2020-06-08 00:07:47     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      67      3664978            0
2020-06-08 00:07:49    arp        192.168.0.1         who        192.168.0.4        2           60           42
2020-06-08 00:07:52     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      72      4106848            0
2020-06-08 00:07:56    arp        192.168.0.3         who        192.168.0.4        4          120           84
2020-06-08 00:07:58     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      17       972390            0
2020-06-08 00:06:35     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      71      3213530            0
2020-06-08 00:06:41     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      66      3773356            0
2020-06-08 00:06:42    arp        192.168.0.4         who        18.189.78.164       3          126            0
2020-06-08 00:06:47     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      70      3982884            0
2020-06-08 00:06:49    arp        192.168.0.1         who        192.168.0.4        2           60           42
2020-06-08 00:06:52     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      71      4042026            0
2020-06-08 00:06:57     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      60      3132008            0
2020-06-08 00:07:03     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      69      3909294            0
2020-06-08 00:07:06    arp        192.168.0.5         who        192.168.0.4        2           42           42
2020-06-08 00:07:09     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      62      3628340            0
2020-06-08 00:07:13    udp      192.168.100.3.52152     ->     239.255.255.250.1900       8         2854            0
2020-06-08 00:07:14     ip  00:11:32:b3:5a:72          ->  c8:34:8e:0f:32:cb      66      3893388            0
2020-06-08 00:07:18    udp        192.168.0.9.bootpc     ->     255.255.255.255.bootps       2          684            0
```

Figure 71 - Rafilteraddr results over whitelist filtering

00:11:32:b3:5a:72 is my host mac address and c8:34:8e:0f:32:cb is NAS drive MAC address. The rest unknown suscipious domain / IP were marked by red lines. They can be recorded for next further investigation.

Next we are going to utilize the Ralabel with Python Classification ML lib to do some advanced analytics.

6.4.2 Relabel

First of all, we need to setup the Label by utilizing RaLabel command. From the visualization conclusion above, we can say the traffic between c8:34:8e:0f:32:cb and 00:11:32:b3:5a:72 is mostly caused by 'bot'. Others we label it as 'normal'. Please check configuration below:

Rlabel_assignment2.conf:

```
chris@DESKTOP-I7H08DT:/mnt/d/wireshark$ cat rlabel_assignment2.conf
RLABEL_ARGUS_FLOW=yes
RLABEL_ARGUS_FLOW_FILE=rlabel_assignment.txt
```

Rlabel_assignment2.txt

```
chris@DESKTOP-I7H08DT:/mnt/d/wireshark$ cat rlabel_assignment2.txt
filter="not net 192.168.0.4 and not net fe80::b7:5e1e:196b:852c" label="bot"
filter="" label="normal"
```

6.4.3 Analytics with Python

Analytics with Python needs a little bit knowledge about ML. Thanks to my other graduate courses and my previous experience,

with Lab 5 guidelines I am able to compare two different ML algorthima. Then compared results from one to another:

I need to decided which can be used for classification features, and which can be labels. Of course, we are going to use "nomal/bot" as classification labels. Regarding about features, we need to consider about overfitting issue, that means we can not use source addr and destination addr as features, and ports can weight too much according to the visualization graphs showing in the last section. So I decision to use traffic [pkts, byts, load] as features.

```
X_features = pd.DataFrame(df1.iloc[:, :label_index])
print("Classification features:", X_features.columns)
y_labels = pd.DataFrame(df1.iloc[:, label_index])
print("Classification targets:", y_labels.columns)
```

```
Classification features: Index(['TotPkts', 'TotBytes', 'Load'], dtype='object')
Classification targets: Index(['Label'], dtype='object')
```

Figure 72 - ML feature selection and labels

- **Decision Tree:**

Train model confusion matrix:

```
pipe_clf = Pipeline([('scl', StandardScaler()), ('clf',DecisionTreeClassifier())])
pipe_clf.fit(X_train, y_train)
pred = pipe_clf.predict(X_train)
cfm=metrics.confusion_matrix(y_train, pred)
print('Train confusion matrix: \n', cfm)
```

```
Train confusion matrix:
 [[  37    2]
 [   0 1721]]
```

Figure 73 – decision tree generates training model

Test (Evaulation):

```
print("test confusion matrix:")
cfm=metrics.confusion_matrix(y_test, pred)
print(cfm)

print("classification report:")
classificationReport = classification_report(y_true=y_test, y_pred=pred)
print (classificationReport)
```

```
accuracy:   0.991
test confusion matrix:
[[  7   4]
 [  0 429]]
classification report:
              precision    recall  f1-score   support

    flow=bot       1.00      0.64      0.78        11
 flow=normal       0.99      1.00      1.00       429

    accuracy                           0.99       440
   macro avg       1.00      0.82      0.89       440
weighted avg       0.99      0.99      0.99       440
```

Figure 74 – decision tree evaluates training model

- **Navie Bay:**

Train model confusion matrix:

```
pipe_clf = Pipeline([('scl', StandardScaler()), ('clf',GaussianNB())])
pipe_clf.fit(X_train, y_train)
pred = pipe_clf.predict(X_train)
cfm=metrics.confusion_matrix(y_train, pred)
print('Train confusion matrix: \n', cfm)
```

```
Train confusion matrix:
 [[  26   13]
 [   4 1717]]
```

Figure 75 - Naive Bay generates training model

Test (Evaulation):

```
print("test confusion matrix:")
cfm=metrics.confusion_matrix(y_test, pred)
print(cfm)

print("classification report:")
classificationReport = classification_report(y_true=y_test, y_pred=pred)
print (classificationReport)
```

```
accuracy:   0.993
test confusion matrix:
[[  8   3]
 [  0 429]]
classification report:
              precision    recall  f1-score   support

    flow=bot       1.00      0.73      0.84        11
 flow=normal       0.99      1.00      1.00       429

    accuracy                           0.99       440
   macro avg       1.00      0.86      0.92       440
weighted avg       0.99      0.99      0.99       440
```

Figure 76 - naive bay training model evaulation

Form the comparsion above, we can see Naïve Bay can has more accuracy over decision tree. I committed my Python work to https://github.com/chrisliu01/network-management/blob/master/assignment2.ipynb [3][18]

6.4.4 Analysis with Tableau

Last but not least, Tableau also provides strightforward method to analyze input datasets. I created worksheet for analyzing label normal / bot through Tableau [17]: https://public.tableau.com/profile/chris.gang.liu#!/vizhome/Assignment2-DataAnalytics-Total/total

Figure 77 - Data Analystics over Tableau

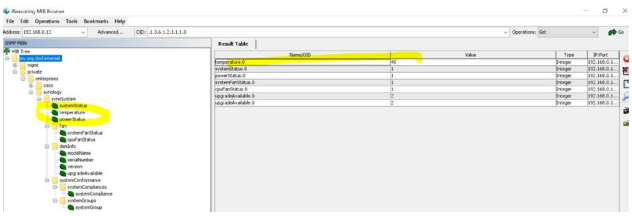
we can classify and identify investigated targets by using several features, like SOURCE, DESINTATION and PROTOCAL. Based on SOURCE and DESINATION, we can decide which device we are looking at. Furthon on, based on PROTOCL, we can decicde what kinds of traffic those two peers talk with each other, such as SNMP, ISCSI, ICMP, TCP ect. For example, I knew NAS drive uses ISCSI protocol to be mounted on host PC. Therefore, if I use ISCSI as a filter, most likely the results belong to the NAS drive.

7 Advanced Usages beyond completion of assignment

During this monitoring and analyzing processes, I was utilizing several advanced usages beyond the assignment requirements. I list here for examples only:

7.1 SNMP Special MIB for certain device

Because each device type has their own SNMP MIB structure, I download Synology System MIB from their official website (http://www.circitor.fr/Mibs/Mib/S/SYNOLOGY-SYSTEM-MIB.mib) to check out which is the most valuable variables needed to be focus on for my Synology Device. I found the "temperature - **.1.3.6.1.4.1.6574.1.2.0**" might be interesting variable I need to focus on:



Then, I was able to set SNMP trap on tool to let it monitor that value for checking system health:

7.2 Argus Filter

In "**Data Pre-processing"** mentioned above, I applied RaLabel to add label, specify output fields and customize output format. All these works can make visualization and data analytics more efficient. I issued several Filter expressions mentioned in official Argus Client Manual [17]

7.3 Ragraph + RA filter

I utilized RA filters with Ragraph to generate customized graph, such as host and port.

7.4 Data analytics with Argus Client tools– Racluster, Racounter, Rasort, rafilteraddr and Rahisto

I facilitated Argus client tools together to identify suspicious behaviors and IP for further investigation.

7.5 Visualization Interactives

I facilitated Tableaus to make the data visualization more interactive and integration.

Interactive data worksheets [16]:
https://public.tableau.com/profile/chris.gang.liu#!/vizhome/Assignment2-DataAnalytics/Dashboard1

Interactive data analytics [17]:
https://public.tableau.com/profile/chris.gang.liu#!/vizhome/Assignment2-DataAnalytics-Total/total

7.6 ML classification comparsion

I exploited the Python ML lib to classify the data more efficiently. By split train and test data with 4:1 rate, I was able to compare the accuracy of classification by Naïve Bay Gaussian and Decision Tree Algorithm. Please check https://colab.research.google.com/drive/1FxK0UI0C47sNfN2CzzRbWfnIq3-irjFb?usp=sharing for my source code.

8 Conclusion and Future work

Assignment 2 gives students a good chance to practice a lot of tools for actual network data collection, monitoring and analyzing in reality for a possible project in future working place. Personally, I appreciated this assignment design and learned a lot.

I can foresee the assignments / projects later on will be based on the techniques and experiences we gained from this assignment. Therefore, I moved on the reading and studying security management and improvement.

REFERENCES

[1] DR. NUR ZINCIR-HEYWOOD, https://web.cs.dal.ca/~zincir/cs6706.html
[2] Quick Reference: Building Cacti Network Monitoring: https://books.google.ca/books?id=A6nUBAAAQBAJ
[3] Chrisgangliu/github: https://github.com/chrisliu01/network-management/blob/master/assignment2.ipynb
[4] Cacti Architecture: https://www.terena.org/activities/tf-noc/meeting6/slides/Cacti_TF-NOC_2012.pdf
[5] Cacti Autom8 Plugin: https://docs.cacti.net/_media/plugin:automate_manual-v0.35.pdf
[6] MIB Depot: http://www.mibdepot.com/cgi-bin/mibvendors.cgi?id=221233
[7] PRTG manual: https://www.paessler.com/manuals/prtg
[8] WireShark Filter: https://wiki.wireshark.org/CaptureFilters
[9] Tableau: https://www.tableau.com/academic/students
[10] Assignment 2 Data Analytics Worksheet : https://public.tableau.com/profile/chris.gang.liu#!/vizhome/Assignment2-DataAnalytics/6-7
[11] Assignment 2 Data source of Argus: https://1drv.ms/u/s!AidJC3PD0CHZndd--8yHxoD-LqRbCw?e=mQF9PJ
[12] Duc Le Lab 5: https://colab.research.google.com/drive/1BG2BPmFGegaMEjLIrKQ-Y589-8n0v8Zz?usp=sharing#scrollTo=W9aNNiTLfsy-
[13] Argus Client Manual : https://manpages.debian.org/jessie/argus-client/ra.1.en.html
[14] Rahisto : http://webpages.sou.edu/~ackler/CF_II.Network_Forensics/Lectures/Ch-5.0.odp
[15] Interactive Workbook for assignment 2: https://public.tableau.com/profile/chris.gang.liu#!/vizhome/Assignment2-DataAnalytics/Dashboard1
[16] Interactive Assignment 2 Label Classification: https://public.tableau.com/profile/chris.gang.liu#!/vizhome/Assignment2-DataAnalytics-Total/total
[17] Colab Assignment 2 Data Analytics: https://colab.research.google.com/drive/1FxK0UI0C47sNfN2CzzRbWfnIq3-irjFb?usp=sharing