Presented at the DLSU Research Congress 2015
De La Salle University, Manila, Philippines
March 2-4, 2015

Local Area Network Analyzer with Multivendor Support without Agents using Switches (LLAMAS)

Kristen Cyril B. Aquino[1], Geanne Ross L. Franco[2], Carlos Javier M. Javier[3], Ria Bianca V. Santiago[4], Dannison Heinrich O. Yao[5]
[1,2,3,4,5] Computer Technology Department, De La Salle University, 2401 Taft Ave., Manila

ABSTRACT - Network analyzers are software that are used to monitor network performance and activity. They make it easier to trace and find problems such as network congestion and inappropriate usage, within the network, or give a general outlook on its health. However, network analyzers are part of a larger scale of applications, known as network managements systems. Aside from network monitoring, they allow centralized configuration of the network devices being handled. Most of the network analyzers and management systems available in the market nowadays either lack multi-vendor support, or they require agents to be installed on the hosts. With these flaws in mind, the study aims to develop a network analyzer that can trace packets in the network without strict restrictions on the vendor of the devices, and without the need for agents to be installed on hosts in the network. As such, the study has successfully researched, implemented, and tested solutions for the present insufficiencies of the technologies addressed in an attempt to eliminate some current limitations to network management.

Keywords - Managed switch; alias; telnet; MAC address

1. INTRODUCTION

In today's information age, a primary concern of many companies, businesses, and industries is how to secure and manage their networks. A well-managed network is a key asset for storage and communication. In terms of storage, what used to occupy papers, files, and cabinets are currently being migrated, if not already fully transferred, to electronic databases. In order for employees to have access to such data efficiently, networks are needed. This need links with communication. Ensuring all aspects of the system have a connection to other assets is essential for any data-centered workflow. Large networks like this will, from time to time, experience problems such as bottlenecks and congestions. For such reasons, solutions have been released in the market, to help monitor and optimize network performance. Network analyzers are made in order to make tracing the sources of these problems easier.

With that said, there is a lack of a switch-based network analyzer that offers important data regarding the network without its core functionalities being dependent on the vendor of the devices in the network, and without the need for additional small programs, called agents, to ensue network monitoring. The study's end goal is a system that monitors a local area network and provides relevant information utilizing captured packets and user-defined aliases for convenient reporting without strict restrictions on the vendors of the devices in use, and without the need for agents to be installed on the host machines.

2. SYSTEM DESIGN

The design of the system divides everything into a set of related tasks, with each division termed a "module". Each module represents a Python script, or a collection of two or more (also known as sub-modules), and only one database with two tables is needed. The entire system is composed of five main modules: MAC Address Manager, Alias Manager, Packet Manager, Report Generator, and User Interface.

A. MAC Address Manager

The MAC Address Manager is responsible for all communication with the switch in the LAN to be monitored. The module consists of two sub-modules:

the Connection Sub-module and the Parser Sub-module.

1. Connection Sub-module

The module connects to a switch via telnet using the telnetlib Python library [3]. In conjunction with the User Interface, it allows the user to connect to each switch in the network, though each switch must be connected to individually. The user needs to input the switch's name (a user-defined string for identification purposes, especially among packets with similar port numbers but coming from different switches; optional), IP address, and the passwords (if there is more than one needed) of the switch in the provided areas in the user interface, as these are the parameters the system will use when establishing the telnet connection. In addition, a text field is provided for the user to input the series of commands needed to bring out the MAC address table in the specified switch. The system saves the latest entered information for future use, specifically for automatic refreshing of the Alias Database with respect to the last connection with a switch. After a successful session, the output is saved as text files, which is passed on to the Parser Sub-module. Should the user wish to connect to and get the data of another switch, the entire process must be repeated.

2. Parser Sub-module

Upon the inputting of the required parameters, the user has the option to select what parser the system will use to parse the current switch's output. This module addresses multi-vendor compatibility by offering user-implemented extensibility. By default, the system provides a parser that parses tables that follow the format similar to the MAC address table shown in Figure 2-1. This is called the "Default Parser", designed for switches that are based on the structure of the Cisco IOS. However, there are cases wherein the MAC address table of the switch does not follow this format (i.e. different branded switch). In this scenario, the user must create a Python script that will parse the table in a manner that would bring out the required data from the switch's telnet output (MAC addresses and switchport interface numbers) like the Default Parser. A directory will be located among the system's files which will store all parsers created for the system, though newly created parsers must be manually imported. The system displays the list of parsers it finds in the repository, and the user selects one, which is basically an indication of how the incoming telnet output from that specific switch should be handled, after which the parsed data is saved in the Alias Database.

```
Switch1#show mac address-table dynamic
          Mac Address Table
-------------------------------------------
Vlan    Mac Address       Type        Ports
----    -----------       --------    -----
   1    000a.b82d.10e0    DYNAMIC     Fa0/1
   1    0012.80b6.4cd8    DYNAMIC     Fa0/2
   1    0012.80b6.4cd9    DYNAMIC     Fa0/3
   2    0014.6915.4100    DYNAMIC     Fa0/10
   3    0018.b921.9200    DYNAMIC     Fa0/20

Switch1#
```

Figure 2-1. Cisco IOS MAC Address Table

It is important to note that the system can automatically retrieve the MAC address tables of the last switch it has successfully been able to connect to using the latest inputted parameters in the Connection Sub-module. This is a feature known as "Auto-refresh", which updates the Alias Database should significant changes be found by re-establishing a telnet connection and going through the data gathering process all over again after a fixed amount of time, replacing or editing data should the need be apparent. The system uses the last inputted variables (switch name, IP address, passwords, and commands) for the automatic reconnection. Parsing the MAC address table will also be done with the most recent parser selected, so the user must change anything (switch commands and parsers) manually should there be a need to do so (i.e. connection to a different switch). In addition, a "Manual Refresh" option will allow the user to prompt the system to forcibly reconnect to the switch if an update is desired. If the user wishes to take the MAC address table of another switch into account and merge it with any existing content in the Alias Database, another telnet connection must be made to the switch, and the entire process repeated all over again. Prior to the assignment of host aliases by the user, the parsed data should be saved in the SQLite database as the indicated data types as shown in Table 2-1.

Table 2-1. Data Types in the Alias Database

Data	**Data Type**
HostNumber	INTEGER PRIMARY KEY
SwitchName (user-defined)	TEXT
MACAddress	TEXT
SwitchportNumber	TEXT
Alias (user-defined)	TEXT

B. Alias Manager

The Alias Manager bridges the user and the Alias Database. Upon request, the system displays the MAC addresses saved in the database via the graphical user interface (GUI), or the User Interface Module. From there, the user may select each address and assign an alias of choice. The correlation of each MAC address to the physical computers is up to the user to determine, but all MAC address are to be listed, and can have their aliases assigned, edited, or revoked at any time. Aliases are strings of a set amount of characters, and unique from other entries in the database, although not all MAC addresses are required to have aliases. They are also in no means necessary for the functionality of the system, and their assignment is entirely up to the user's discretion.

C. Packet Manager

The Packet Manager is responsible for the capturing and parsing of IPv4 packets that pass through the managed switch. The core of the code is based on Prashant Pugalia's Python packet sniffer [2], which uses raw sockets in Linux to capture packets, and breaks them down, afterwards saving them into the database. The data the module retrieves is indicated in Table 2-2. However, only TCP, UDP, and ICMP packets are captured because they represent all IPv4 end-to-end traffic. Other protocols under IPv4 do not deal with network sessions, and thus are not considered, or dropped. [1]

Table 2-2. Data Types in the Packet Database

Data	Data Type
Packet Number	INTEGER
Destination MAC Address	TEXT
Source MAC Address	TEXT
Destination IP Address	TEXT
Source IP Address	TEXT
Destination Port Number	TEXT
Source Port Number	TEXT
Protocol	INTEGER
PacketSize	INTEGER
CaptureTime	TEXT

D. Report Generator

The Report Generator pools data from both the Packet and Alias Databases and prepares them to be drawn on the GUI via the User Interface. Reports are divided into two – Packet Reports and Bandwidth Reports.

Packet Reports are tables of data formed by joining information from both the Alias Database and the Packet Database. The data expected to be previewed in Packet Reports are as follows: Packet Number, Destination Alias, Source Alias, Destination IP Address, Source IP Address, Destination Port Number, Source Port Number, Protocol, Destination Switch Name, Destination Switchport Interface, Source Switch Name, and Source Switchport Interface. If an alias is not present for a specific host, the corresponding MAC address is used instead. Likewise, if a switch is not identified, all conforming fields in the report will remain blank.

This joined table may be filtered by host to clearly see network traffic corresponding to a single computer. For Bandwidth Reports, the user chooses a particular host, and whether to view the upload bandwidth usage or the download bandwidth usage. For upload bandwidth usage, the Report Generator takes all packets in the Packet Database with a source alias (or MAC address) of the host selected by the user. Then, it takes all the packets in ascending order (by packet number), and translates the CaptureTime and PacketSize values as x and y coordinates, respectively. The y values are then added and averaged together within a given interval of x (30 seconds) for an estimated average. If download bandwidth usage is in question, the Report Generator will take into account all packets that have the selected host as the destination. After processing, these values are sent to the User Interface for graphing in the GUI.

E. User Interface

The User Interface is responsible for instantiating communication between the user and the system. It provides forms, tables and drop down lists for data input, a text field for commands to be used with the managed switch, and tables and charts for reports.

For reports, the User Interface takes the data handed to it by the Report Generator and displays it in the GUI. Likewise, this module handles the Bandwidth Reports by simply plotting the calculated x and y values received from the Report Generator on a 2D line graph.

3. EXPERIMENTS

For testing purposes, the basic system set-up consists of four computers – one to run the system, and the other three to simulate hosts running in a typical network setting, as well as a managed switch connecting everything together. The host running the system is connected to the switch's fully configured SPAN port (or equivalent). Packets passing through the managed switch are mirrored, sent out the SPAN

port, and captured and parsed by the system. A diagram depicting the system setup is shown Figure 3-1.

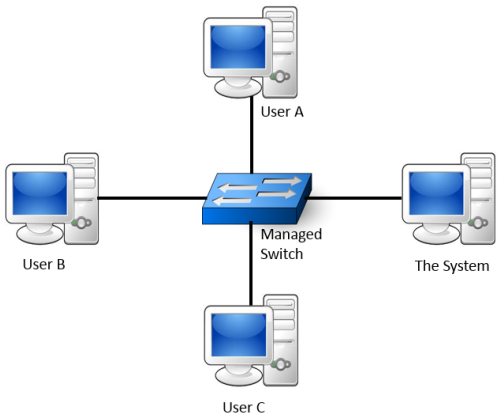

Figure 3-1. System Setup

The system is developed in Python 2.7, with SQLite 3 as the database of choice, on Ubuntu 14.

Based on the nature of system, only two modules were subject to experiments and preliminary testing – the Packet Manager and the MAC Address Manager. All other modules merely rely on the manipulation of data retrieved by these two modules, or simple string inputs provided by the user. Should all necessary data be successfully retrieved, the system's core is effectively complete.

The first module to be completed and tested was the Packet Manager. Tested with heavy internet traffic on an Ubuntu virtual machine, the module was put to the test to see if it can capture thousands of IPv4 packets, possibly even more, even in bursts, and capture and parse each one. Though performance is not a concern of the system, the module held up well, and was able to retrieve the needed data from the numerous packets. Figure 3-2 shows the code snippet that parses both the Ethernet and IP headers of the packets. Figure 3-3 displays the data retrieved from each of the packets captured before being saved in the database.

```
def parse_packet(packet) :

    #gets packet capture time
    captureTime = time.asctime(time.localtime(time.time()))
    print 'Capture Time : ' + str(captureTime)

    #parse ethernet header
    eth_length = 14

    eth_header = packet[:eth_length]
    eth = unpack('!6s6sH' , eth_header)
    eth_protocol = socket.ntohs(eth[2])
    destinationMACAddress = eth_addr(packet[0:6])
    print 'Destination MAC : ' + destinationMACAddress
    sourceMACAddress = eth_addr(packet[6:12])
    print 'Source MAC : ' + sourceMACAddress


    #Parse IP packets, IP Protocol number = 8
    if eth_protocol == 8 :
        #Parse IP header
        #take first 20 characters for the ip header
        ip_header = packet[eth_length:20+eth_length]

        iph = unpack('!BBHHHBBH4s4s' , ip_header)

        version_ihl = iph[0]
        version = version_ihl >> 4
        ihl = version_ihl & 0xF

        iph_length = ihl * 4

        ttl = iph[5]
        protocol = iph[6]
        s_addr = socket.inet_ntoa(iph[8]);
        d_addr = socket.inet_ntoa(iph[9]);

        print 'Protocol : ' + str(protocol)
        destinationIPAddress = str(d_addr)
        print 'Destination IP Address : ' + destinationIPAddress
        sourceIPAddress = str(s_addr)
        print 'Source IP Address : ' + sourceIPAddress
```

Figure 3-2. Code Snippet from Packet Manager

```
Protocol : 17
Destination IP Address : 192.168.95.254
Source IP Address : 192.168.95.131
Dest Port : 67
Source Port : 68
Packet Size : 342
--------------------------------------------------
3325
Capture Time : Tue Nov 11 13:21:44 2014
Destination MAC : 00:0c:29:71:c6:74
Source MAC : 00:50:56:ff:c8:69
Protocol : 17
Destination IP Address : 192.168.95.131
Source IP Address : 192.168.95.254
Dest Port : 68
Source Port : 67
Packet Size : 342
--------------------------------------------------
3326
Capture Time : Tue Nov 11 13:21:44 2014
Destination MAC : 00:50:56:ec:f2:54
Source MAC : 00:0c:29:71:c6:74
Protocol : 6
Destination IP Address : 37.230.96.108
Source IP Address : 192.168.95.131
Dest Port : 80
Source Port : 36820
Packet Size : 54
--------------------------------------------------
3327
Capture Time : Tue Nov 11 13:21:44 2014
Destination MAC : 00:0c:29:71:c6:74
Source MAC : 00:50:56:ec:f2:54
Protocol : 6
Destination IP Address : 192.168.95.131
Source IP Address : 37.230.96.108
Dest Port : 36820
Source Port : 80
Packet Size : 60
--------------------------------------------------
3328
```

Figure 3-3. Parsed Data from Captured Packets

The other module tested was the MAC Address Manager. The computer running the module was connected to a Cisco Catalyst 2960 switch, and used the default "Cisco parser". For testing purposes, the switch connection parameters (IP address and switch commands) were hardcoded into the module, though in normal circumstances they are to be provided by the user. Figure 3-4 shows the Python script used to establish the telnet connection to the switch and parse the MAC addresses and the switchport interface numbers from the IOS's MAC address table.

```
import getpass
import sys
import telnetlib

HOST = raw_input("Enter HOST address: ")
password = raw_input("Enter your password: ")
tn = telnetlib.Telnet(HOST)
tn.read_until("Password: ")
tn.write(password + "\n")
tn.write("en\n")
tn.read_until("Password: ")
tn.write("class\n")
tn.write("show mac address-table"+ "\r\n")
tn.write(" \n")
tn.write("ls \n")
tn.write('exit' + '\r')
with open("out.txt", "w") as f:
        f.write(tn.read_all())
file = open("out.txt" , "r");
word = file.read()
a = []
a = word.split('\n')
z = []
for x in range(7, len(a)-7):
        if a[x] == "":
                print "blank"
        elif "--More--" in a[x]:
                l,m = a[x].split(" ■■■■■■■■■        ■■■■■■■■■ ")
                z,x,c,y = m.split("    ")
                print x,y
        else:
                z,x,c,y = a[x].split("    ")
                print x,y
```

Figure 3-4. MAC Address Manager (Telnet and Parsing) Code

Testing proved successful as the switch was successfully connected to, the MAC address table retrieved, shown in Figure 3-5, and the MAC address and switchport interfaces parsed and ready to be saved in the database, as seen in Figure 3-6.

```
Switch#show mac
Switch#show mac ad
Switch#show mac address-tablecharacter-bits  Size of the e

----    -----------       --------    -----
 All    0100.0ccc.cccc    STATIC      CPU
 All    0100.0ccc.cccd    STATIC      CPU
 All    0180.c200.0000    STATIC      CPU
 All    0180.c200.0001    STATIC      CPU
 All    0180.c200.0002    STATIC      CPU
 All    0180.c200.0003    STATIC      CPU
 All    0180.c200.0004    STATIC      CPU
 All    0180.c200.0005    STATIC      CPU
 All    0180.c200.0006    STATIC      CPU
 All    0180.c200.0007    STATIC      CPU
 All    0180.c200.0008    STATIC      CPU
 All    0180.c200.0009    STATIC      CPU
 All    0180.c200.000a    STATIC      CPU
 All    0180.c200.000b    STATIC      CPU
 All    0180.c200.000c    STATIC      CPU
 All    0180.c200.000d    STATIC      CPU
 All    0180.c200.000e    STATIC      CPU
 All    0180.c200.000f    STATIC      CPU
 All    0180.c200.0010    STATIC      CPU
 All    ffff.ffff.ffff    STATIC      CPU
   1    0021.972a.fb0e    DYNAMIC     Fa0/3
Total Mac Addresses for this criterion: 21
Switch#
```

Figure 3-5. Cisco IOS MAC Address Table

```
Enter HOST address: 192.168.1.100
Enter your password: passsword
-------------- -----
0100.0ccc.cccc    CPU
0100.0ccc.cccd    CPU
0180.c200.0000    CPU
0180.c200.0001    CPU
0180.c200.0002    CPU
0180.c200.0003    CPU
0180.c200.0004    CPU
0180.c200.0005    CPU
0180.c200.0006    CPU
0180.c200.0007    CPU
0180.c200.0008    CPU
0180.c200.0009    CPU
0180.c200.000a    CPU
0180.c200.000b    CPU
0180.c200.000c    CPU
0180.c200.000d    CPU
0180.c200.000e    CPU
0180.c200.000f    CPU
0180.c200.0010    CPU
ffff.ffff.ffff    CPU
0021.972a.fb0e   Fa0/3
```

Figure 3-6. Parsed MAC Addresses and Switchport Interface Numbers

Due to physical hardware limitations (only Cisco switches were available for use), testing of this module was only performed on Cisco devices. However, since this module, and the system as a whole, aims to address multi-vendor support with its expandable framework via changeable connection parameters and swappable Python scripts, the researchers sought professional technical advice on the theory and the design of the system as support for its claim to work across multiple devices via user expansion. This design was proposed and shown to experts in the field during the course of its conception and development, and was deemed a possible, working solution – capable of expansion without affecting the rest of the system. Thus, experiments and recommendations have shown that the system, along with its set objectives, is not only logical, but purposeful as well.

4. CONCLUSION

Network analyzers are vital for large local networks that require low fault tolerances. In the event of undesirable incidents, such as malicious or inappropriate usage, the source must be traced as soon as possible, so that proper mitigation and action may take place. Constant monitoring of the overall performance of the system by ensuring the total absence of bottlenecks is also vital, and a task where in network analyzers and management systems are almost essential to complete.

Developing a network analyzer with a user-implemented expansion framework to accommodate networks with varying vendors of devices alleviates the concern of native support, and allows the user to vary between device brands and still hope for a working network analyzer. In addition, the complete elimination of host agents ensures that the management is "centralized", in a sense that the setup only includes the computer to run the system, and the switch or switches to be monitored. Deployment, in this context, is much faster, and much less of a hassle.

The idea of the system is to give the users freedom to tweak the way it handles communication with switches to their liking. Providing such a platform also removes the burden of patching for support from the developers. It is reasonable to conclude that should the right parser be developed for the right switch, the system is potentially compatible with all kinds of managed switches, which, in itself, is a highly-valuable asset.

5. RECOMMENDATIONS

Despite its solutions to vendor limitations and agent requirements, LLAMAS still has areas where improvement is needed, which the researchers highly encourage anyone interested to work on.

Possibly the biggest drawback of the system is the potential size of the Packet Database. Each packet captured takes approximately 20 bytes of space in the disk. Testing has shown that five minutes worth of network traffic generated by typical web browsing by a single host amounts to approximately 12,000 captured packets. At this rate, if the number of work hours in an office with 24 hosts goes for 8 per day, this will result to, more or less, 27,648,000 captured packets, leading to a total database size of around 527 megabytes a day. Ideally, this should pose no problem if the system is being run on a dedicated server, and the user may always purge the database should it occupy too much space. However, the researchers agree that a built in system to decrease the disk space occupancy of the database would greatly reduce the hardware requirements needed to run the system. The first recommendation, then, would be to lessen the size of the Packet Database on the disk without compromising the much needed data it carries, whether that be through compression, archiving, or any other method.

Another area of probable improvement would be the handling of multiple switches. The system was originally designed for and tested on one switch, but realistically speaking, corporate networks will never run on a single switch. Even though the system supports multiple switches, the user needs to telnet to each one individually. The researchers recommend future work in expanding the platform such that multiple telnet connections may be performed simultaneously, and the data required of each switch may be parsed and saved into the database at the same time. Not to mention, multiple instances of

parameters used to connect to separate switches should also be saved and capable of being called whenever. This would include taking into account the automatic refreshing of the Alias Database after set intervals, or as demanded by the user.

6. ACKNOWLEDGEMENTS

The authors would like to thank Sir Isaac Herculano Sabas for his constant guidance, recommendations, and insights on the technical aspects of the system, and for his comments during the authoring of this document.

7. REFERENCE

[1] Microsoft. IPv4 Protocols. (2009). Retrieved from http://technet.microsoft.com/en-us/library/dd392264(v=ws.10).aspx

[2] Pugalia, Prashant. (2011). Code a Network Packet Sniffer in Python for Linux. Retrieved from http://www.binarytides.com/python-packet-sniffer-code-linux/

[3] Telnetlib – Telnet Client. (n.d.). Retrieved from https://docs.python.org/2/library/telnetlib.html