# “Study of EtherTrace - a layer-2 topology discovery tool”

by

Zohra Jabeen

A project report submitted in partial fulfillment of the requirements for the degree of

Master of Science in Internetworking
(Department of Computer Science)
**University of Alberta**

Supervisor
Dr. Mike MacGregor

November 2011

# ABSTRACT

Network of these days have grown in size and complexity. The dependency on computer networks is increased in order to run a business successfully. To keep the network (devices and services) up and running is the highest priority job for the network administrators. They use various tools and techniques for network monitoring and maintain data logs that help them in various types of analysis and performance monitoring.

Virtual LANs has also gained significant importance in network industry because of their performance and physical topology independent feature.

VLANs use layer-2 switch to communicate with the other network devices on VLAN. Intercommunication of VLANs can be performed with the use of router that is a layer-3 device. IP layer paths that are otherwise disjoint can be trunked at layer-2, thus it introduces sharing between paths of layer-2 and layer-3. This VLAN induced sharing results in hidden dependencies between IP level network and the physical infrastructure. The paper 'Characterizing VLAN-induced sharing in a campus network' motivated me to study about VLANs and dependencies induced between network layers due to VLANs inter-communication.

This project first summarizes the paper 'Characterizing VLAN-induced sharing in a campus network' [1] and then discusses EtherTrace [2] that's a layer-2 topology discovery tool, designed and implemented for topology discovery and measuring VLAN induced sharing in a large campus network. EtherTrace works on three sets of data ARP tables, bridge tables and IP traceroute results. This study focuses on scripts that use traceroute results for determining the path elements. Traceroute results between nodes (end hosts in the most distinct subnets) on a large campus network were used in the actual study. We however didn't have access to the actual files, so files containing dummy data were created for this study.

This project explains the format of the file to store traceroute results, and then it explains the usage of the script that handles traceroute results. EtherTrace uses My SQL as a backend database; this study also discusses data stored in the tables and explains the areas that can be useful in determining layer-2 path between hosts.

# Table of Content

ABSTRACT ......2
ACKNOWLEDGEMENTS ......5
Chapter 1 – Introduction ......6
1.1 Overview of the project ......6
1.2 Network management and monitoring ......6
1.3 VLAN ......7
1.3.1 Inter-VLAN Communication ......7
1.4 Dependencies between layer-2 and IP topologies ......8
1.5 Traceroute ......8
1.5.1 How traceroute works ......9
Chapter 2 – Summarizing the paper "Characterizing VLAN-induced sharing in a campus network". ...... 11
Chapter 3 – How EtherTrace monitors traceroute results ...... 13
3.1 Overview ...... 13
3.2 File format ...... 13
3.2.1 General format ......14
*3.2.2 Sample 1 ......14*
*3.2.3 Sample 2 ......15*
*3.2.4 Sample 3 ......16*
3.3 Database information ...... 17
3.4 Three different ways to run tr.py script ...... 18
3.4.1 Usage 1 - Recursively parses all the files in the root directory. ......18
*Usage 1 - l3_devs table ......20*
*Usage 1 - l3_tr table ......20*
*Usage 1 - l3_tr_sd table ......21*
3.4.2 Usage 2 - Recursively parses a specified file. ......22
*Usage 2 - l3_devs table ......23*
*Usage 2 - l3_tr table ......24*
*Usage 2 - l3_tr_sd table ......25*

*Usage 2 - l3_tr_meta table* ....................25
3.5 Different scenarios .................... 26
3.5.1 Case 1 - Invalid IP address in an input file .................... 26
*Case 1 - l3_devs table* .................... 27
*Case 1 - l3_tr table* .................... 27
*Case 1 - l3_tr_sd table* .................... 28
*Case 1 - l3_tr_meta table* .................... 28
3.5.2 Case 2 - A failed traceroute result .................... 28
*Case 2 - l3_tr_meta table* .................... 29
3.5.3 Case 3 - A loop is created in a path .................... 30
*Case 3 - l3_devs table* .................... 31
*Case 3 - l3_tr table* .................... 32
*Case 3 - l3_tr_sd table* .................... 33
*Case 3 - l3_tr_meta table* .................... 34
3.5.4 Case 4 - Redundant paths in a file .................... 34
*Case 4 - l3_devs table* .................... 36
*Case 4 - l3_tr table* .................... 37
*Case 4 - l3_tr_sd table* .................... 39
*Case 4 - l3_tr_meta table* .................... 39
3.5.5 Case 5 - More than one source-destination pair .................... 40
*Case 5 - l3_devs table* .................... 42
*Case 5 - l3_tr table* .................... 45
*Case 5 - l3_tr_sd table* .................... 46
3.6 Conclusion .................... 47
Chapter 4 – Summary and Future work .................... 48
REFERENCES .................... 49

# ACKNOWLEDGEMENTS

Many people have played an important role during the accomplishment of this project. I would like to thank them all as without their support this project would not have been possible.

First of all I would like to thank the omnipresent God for answering my prayers, for giving me the strength to complete this study.

My foremost and utmost gratitude goes to my supervisor Dr. Mike MacGregor, whose sincerity and encouragement will never be forgotten. He has been my inspiration as I hurdle all the obstacles in the completion of this project. He gave me a lot of positive perspective in life. I thank him for sharing his valuable time and for giving me helpful information to finish this project.

I would like to express my gratitude towards my husband and parent who has always been there for me whenever I needed them, the encouragement they gave to keep me going and been my source of strength and inspiration.

My thanks and appreciations also go to my friends and colleagues for their moral support.

Last but not the least; I am thankful to my lovely kids for their patience throughout my studies.

# Chapter 1 – Introduction

## 1.1 Overview of the project

This project is a study of EtherTrace [2] that's a layer-2 topology discovery tool. EtherTrace is designed by students of Georgia Institute of Technology, Atlanta, as part of their study about VLAN induced sharing in their campus wide network.

This project first presents a formal definition of VLANs, outlines general network monitoring concepts and then discusses EtherTrace, a passive layer-2 topology discovery tool. IP traceroute results used by EtherTrace are discussed in the later section to demonstrate the idea of determining layer-2 path between hosts on different VLANs.

Following section outlines the report.

Chapter 1 briefly discusses the need of network monitoring techniques, an overview of VLANs, inter-VLAN communications. It also presents a brief comparison of layer-2 and layer-3 network monitoring.

Chapter 2 summarizes the paper 'Characterizing VLAN-induced sharing in a campus network'.

Chapter 3 demonstrates how EtherTrace scripts handle traceroute results. This section presents different scenarios that help understanding the usage of these scripts and information they store in the database.

Chapter 4 first summarizes this study and then suggests future work.

## 1.2 Network management and monitoring

With the growing use of computer networks in today's business market, the dependency on computer network has increased to run the business successfully. Network management has become an essential factor in successfully operating a network.

Network Management is a broad functional area, it can be defined in general as a set of activities where a variety of tools, applications, and devices are utilized by network administrator to operate, provision, monitor and maintain different networks in order to keep the network up and running and provide a consistent level of reliability and performance.

Network monitoring is the information collection function of network management. The purpose of network monitoring is to collect useful information from various parts of the network so that the network can be managed and controlled using the collected information. This information helps troubleshooting and resolving issues when they occur so that network services do not slow down or stop functioning for extended periods of time.

Network management systems are used for various reasons e.g. availability of network devices, performance of devices, utilization & throughput of devices/servers, delay or error rate on a network link etc. As networks of these days contain various devices ranging from simple to complex, a variety of monitoring techniques are utilized by these systems.

## 1.3 VLAN

VLAN – virtual LAN is just like an ordinary LAN that is used as a broadcast domain, the only difference is that a VLAN logically groups the network users and resources with a common set of requirements and goals rather than physically separating them on a LAN.

VLANs are useful in situations where the need exists to separate the logical topology of network segments from the physical topology. A VLAN uses layer-2 switches to communicate with in that VLAN. Different ports on a layer-2 switch can be assigned to different VLANs e.g. for a three port switch, ports 1 and 2 can belong to VLAN X and port 3 can belong to VLAN Z; which means port 1 and 2 can communicate to each other but port 1 and 2 can't communicate to port 3 however physically they are on the same network but logically they are on different VLANs.

By default, hosts in a specific VLAN cannot communicate with hosts that are members of another VLAN, if inter-VLAN communication is required then a router is required.

### 1.3.1 Inter-VLAN Communication

When talking about Virtual LANs in a complex network, intercommunication of various VLANs is an important factor to discuss. Because VLANs isolate traffic to a defined broadcast domain and subnet, network devices in different VLANs cannot communicate with each other natively.

As mentioned earlier that Ethernet switches are used for creating a VLAN. A Switch is a layer-2 device, it cannot forward frames between VLANs. To allow communication between VLANs either a Layer-3 switch or a router is required to forward frames between routers.

For example, port 1 on a switch belongs to VLAN X and port 2 belongs to VLAN Z. If all of the switch's ports were part of VLAN X, the hosts connected to these ports could communicate without any issue but as they are part of two different VLANs, a router or a layer-3 switch needs to be involved.

## 1.4 Dependencies between layer-2 and IP topologies

VLANs function at layer-2. As mentioned earlier that since VLAN's purpose is to isolate traffic within the VLAN, in order to bridge from one VLAN to another either a router or a layer-3 switch is required. The router works at the higher layer-3 network protocol, which requires that network layer segments are identified and coordinated with the VLANs. In a LAN employing VLANs, a one-to-one relationship often exists between VLANs and IP subnets, although it is possible to have multiple subnets on one VLAN or have one subnet spread across multiple VLANs.

The interaction between layer-2 and IP topologies in these VLANs introduces hidden dependencies between IP level network and the physical infrastructure.

Although the entire layer-2 infrastructure may be owned and operated by a single network entity, maintaining the mapping between layer-2 devices and paths for IP subnets can be challenging because of size and changes in configuration over time. Unlike IP networks, where ICMP allows for topology discovery, there is no standard method for discovering layer-2 topology. Mechanisms such as Cisco Discovery Protocol (CDP) can perform active traceroutes on layer-2, but they require all switches in the network to run CDP, which is seldom the case, due to the heterogeneity of devices in the network. This necessitates developing new systems that can infer layer-2 topology and its mapping to IP network using interfaces that are widely supported by layer-2 switches. [1]

## 1.5 Traceroute

The tool EtherTrace uses IP Traceroute to collect data between routers (hosts on different VLANs). As we discussed already that that inter-VLAN communication is performed via routers.

Traceroute is a program that is often used as a network diagnostic tool; it measures the sequence of routers an IP packet travels from a source to a destination. Traceroute is intended to use on a network where routers (at least two of them) exist. Traceroute indicates various diagnostic facts such as how hosts are connected to each other, connection failures, connection delays and where routing loops exist etc. Ethertrace will address all these facts.

### 1.5.1 How traceroute works

Every IP packet contains a field in its header called the TTL field. This is a number that can range from 0 to 255. When a packet is sent out from a machine, it starts with a relatively high TTL (usually 255) and each router the packet passes through along the way to its destination decrements the TTL value by one. If in the way of decrementing the TTL value the router finds the new value will be zero, the packet is discarded and an ICMP error message is sent back to the original sender. The idea is that no packet should be able to live on the network forever. Eventually, after being forwarded 255 times, a packet will just disappear from the network.

An example of a traceroute (used an open source network tool [4] to get this traceroute)

Source IP address is 50.99.201.182, destination IP Address is: 111.11.111.11

| Hop | (ms) | (ms) | (ms) | IP Address | Host name |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 206.123.64.46 | - |
| 2 | 0 | 0 | 0 | 8.9.232.73 | xe-5-3-0.edge3.dallas1.level3.net |
| 3 | 31 | 0 | 0 | 4.69.145.210 | ae-4-90.edge10.dallas1.level3.net |
| 4 | 1 | 1 | 1 | 4.68.62.230 | att |
| 5 | 37 | 39 | 39 | 12.123.16.110 | cr2.dlstx.ip.att.net |
| 6 | 41 | 39 | 39 | 12.122.28.178 | cr2.la2ca.ip.att.net |
| 7 | 36 | 36 | 36 | 12.123.30.249 | cr84.la2ca.ip.att.net |
| 8 | 36 | 35 | 35 | 12.122.129.49 | gar2.lsrca.ip.att.net |
| 9 | 44 | 44 | 44 | 12.118.130.18 | - |
| 10 | 228 | 228 | 227 | 221.176.17.89 | - |
| 11 | 228 | 228 | 228 | 221.176.17.246 | - |
| 12 | 228 | 228 | 228 | 221.176.17.61 | - |
| 13 | Timed out | Timed out | Timed out | | - |
| 14 | Timed out | Timed out | Timed out | | - |
| 15 | 266 | 266 | 266 | 221.176.15.46 | - |
| 16 | 266 | 268 | 266 | 221.176.21.178 | - |

| | | | | | |
|---|---|---|---|---|---|
| 17 | 274 | 274 | 274 | 111.11.64.46 | - |
| 18 | 269 | 270 | 273 | 111.11.64.238 | - |
| 19 | 270 | 270 | 270 | 111.11.74.10 | - |
| 20 | 279 | 275 | 274 | 111.11.111.11 | - |

Trace complete

In this traceroute result, it can be noticed that it took 20 hops to go from IP address 50.99.201.182 to 111.11.111.11 and that the round trip time was roughly 274-279 ms (based on the 3 numbers on the last line). RTT's reported are the round trip times from the source host to that router hop.

Places where a hostname is not displayed it simply means that a reverse DNS lookup on the address failed, so the name of the system could not be determined.

If it displays * * * or 'timed out', it means that the target system could not be reached.

Any connection over the Internet actually depends on two routes: the route from source system to the destination and the route from that destination back to source. These routes are often completely different (asymmetric). Ethertrace has made an assumption that these routes are symmetric and uses pair wise traceroute results to and from the routers.

## Chapter 2 – Summarizing the paper "Characterizing VLAN-induced sharing in a campus network".

The paper "Characterizing VLAN-induced sharing in a campus network" [1] discusses dependencies between layer-2 and layer-3 technologies that a VLAN of one layer can possibly introduce when communicates to other layer's VLAN. This paper characterizes these dependencies and outlines their extent and impact in a large and complex campus network. It performs a cross-layer analysis with a focus on how VLANs create sharing between IP layer paths that are otherwise disjoint. It also characterizes the dependencies that exist among IP subnets that run over the VLANs on this large campus network. They also have discussed the purpose of this sharing, and examined how this sharing impacts to accuracy and network fault diagnosis.

In order to achieve data that helped performing this study, these folks has designed a passive layer -2 topology discovery tool called EtherTrace. This tool is implemented in Python that uses MYSQL as a backend database. EtherTrace is made available publicly.

EtherTrace mostly uses ARP tables from the switches, bridge tables from the routers and IP traceroutes between hosts for inferring layer2 topology discovery. The logic it uses to discover layer-2 corresponding to each IP path segment is that since the bridges on LAN form a tree, only one layer-2 path between any pair of hosts exists at any time. The bridges along the path on the same VLAN must always receive frames from two hosts on two separate ports.

For determining the path elements on layer-2, EtherTrace considers two cases:

Case1: When hosts exist on the same VLAN and IP subnet - It uses ARP tables to obtain MAC addresses of the two hosts and determines whether for some VLANs, there exists a set of switches that receives the MAC addresses of the two hosts on separate ports of that VLAN. Obtaining bridge tables from switches and ARP tables from routers require having administrative rights to these devices.

Case 2: When hosts exist on different VLANS - EtherTrace can determine the layer-2 path between these hosts by using IP traceroute tool. For running IP traceroute tool, administrative rights are not required.

The objective of this project is to this study the component of EtherTrace that deals with the case when hosts exist on different VLANs (i.e. Case 2) and comprehend the traceroute results so that they can be used to determine the path between two hosts.

Types of data - They have used three sources of data for this study, all from the Georgia Tech campus network.

- Bridge table entries obtained from the switches.
- ARP tables from the routers.
- IP traceroutes between CPR nodes [2] that are the end hosts deployed in distinct subnets on the campus network. These nodes perform pair wise traceroutes to each other once every five minutes.

They have used pair wise IP trace routes between two nodes (routers) and measured IP topology using these results. They concluded that an IP edge exists between two IP routers if the routers appear as adjacent hops in any of the traceroutes between the nodes. As they had access to all the routers and switches, they were able to de-alias IP addresses of different interfaces of a single router.

To infer layer-2 topology for the nodes (routers) belong to same VLAN, Case 1 was utilized. Case 2 was applied for the nodes that belong to different VLANs. To infer the path between two nodes they obtained the layer-2 path outlined by the IP hops along that path, the overall path was obtained by concatenating those IP addresses.

EtherTrace makes one assumption regarding the IP paths. As we know that the traceroute results do not provide reverse path information and traceroute's ICMP TTL time-exceeded messages use the IP address of the return interface of the router as the source address. Hence if the IP paths are asymmetric, then the adjacent IP addresses that appear in a traceroute path may belong to different subnets and therefore different VLANs, it makes the IP traceroute unsuitable for computing the layer-2 path.

In this study they have made an assumption that the IP paths are symmetric and this problem does not arise. Therefore they used the IP address of the forward interface on the router configuration.

After obtaining different path information they have presented layer-2 topology for the campus network and then performed an analysis for the extent of infrastructure sharing among IP edges and paths. They presented statistics to show the sharing of layer-2 edges with layer-3 edges, they also presented number of layer-3 paths that shared a network element.

Finally, they presented how this sharing and dependencies affected the accuracy and specificity of fault diagnosis.

# Chapter 3 – How EtherTrace monitors traceroute results

## 3.1 Overview

In order to infer the layer-2 topology EtherTrace handles three different sources of data, bridge tables, ARP tables and IP traceroutes. EtherTrace consists of nine python scripts to handle these different sources of data. The scripts that are used to deal with IP traceroute results ate tr.py and tr_db.py.

This chapter illustrates how EtherTrace parses the IP traceroute results obtained from two routers for different date and time, and how does it store this data in MySQL database. IP traceroute results are saved in a text file with the date-time they were taken. It's not the real time traceroute results obtained directly from any router; rather it's the data files these researchers have obtained from their Information Technology department for research purposes, the department keeps these files for audit purposes. Data is saved in .txt files in a specific format. EtherTrace reads and parses these .txt files.

Two EtherTrace scripts named tr.py and tr_db .py are used for this purpose. EtherTrace not only parses a file saved in the work directory (i.e. EtherTrace-v0.1), it also parses a file saved in a directory and a list of files saved in a directory. EtherTrace uses the term 'usage' for referring these different file locations.

Traceroute results saved in the text file may not be guaranteed a sorted list. tr.py script is written to first read these text files and then it sorts the file by time stamp; this sorted file will then be parsed and save useful information in the database.

Although EtherTrace is available publicly but there is no documentation available that outlines a general traceroute file format used by EtherTrace. This initial study of EtherTrace focuses on how to use EtherTrace so that it can be effectively utilized for network analysis.

The later sections in this chapter will describe how EtherTrace saves useful information in the tables and explain the objective of different fields in the tables. All the files (IP traceroute results) used in the following section are dummy. Real data can be used for meaningful data analysis as done by Georgia University students in their research paper for characterization of VLAN-induced dependencies.

## 3.2 File format

In the following section a general file format in which a 'file-to-parse' can be saved is shown. Later I've shown possible variations in the general file format and ran tr.py script to display the output that is a sorted list.

### 3.2.1 General format

EtherTrace reads a file properly if it's saved in the following format:

Date Time stamp IP Address *[tab]*IP Address*[tab]*IP Address

Where:
Date=YYYY-MM-DD or YYYY/MM/DD
Time stamp=hh:mm:ss or hh:mm
IP Address=standard format i.e. values within the range of 0.0.0.0 – 255.255.255.255
IP addresses must be separated by Tab and there is no limit of IP addresses

#### 3.2.2 Sample 1

If a file uses no colon at the end of the time stamp and/or a semicolon at the end of the path, EtherTrace reads this file properly.

**Sample file**

```
2011-08-12 15:00:20  7.7.7.7 9.2.1.9 13.13.3.1;
2011-08-17 10:05:27  11.11.11.21     12.12.12.12     113.13.31.2;
2011-08-14 00:00:20  13.43.225.101 125.125.126.117        110.111.12.13;
2011-09-11 10:05:27  110.111.111.20          123.196.116.200        1.1.1.1;
2011-09-17 10:05:27  11.11.125.200 20.20.02.20     15.26.26.18;
2011-09-13 21:05:27  170.171.111.20          190.192.176.200        15.26.26.18;
```

Running the script using the following usage:

zohrra@innisfree:~/EtherTrace-v0.1$ python tr.py -d fileformat_tr -k

We can make few observations in the output displayed in the following section:

- Initially the program simply reads and prints all the lines in the order and format they appear in the original file.
- Once the entire file is displayed the program prints 'sorting the trs'. (It uses python's built-in sort function to sort this list).
- Once sorting is done it prints 'done' and prints the result in the sorted order by date-time.

- The final sorted list doesn't display the date time, rather it assigns a line number to the original (unsorted) list and the sorted list refers to that number.
- The last IP address in each path is considered as the destination device's IP address and it's not get picked so that the sorted list displays only the IP addresses a source device traversed to reach to the destination device on a specific date and time.

**Output**

```
Processing: fileformat_tr/router1/tr-file.txt

2011-08-12 15:00:20    7.7.7.7 9.2.1.9 13.13.3.1;

2011-08-17 10:05:27    11.11.11.21    12.12.12.12    113.13.31.2;

2011-08-14 00:00:20    13.43.225.101   125.125.126.117 110.111.12.13;

2011-09-11 10:05:27    110.111.111.20  123.196.116.200 1.1.1.1;

2011-09-17 10:05:27    11.11.125.200   20.20.02.20    15.26.26.18;

2011-09-13 21:05:27    170.171.111.20  190.192.176.200 15.26.26.18;

sorting the trs
done
fileformat_tr/router1/tr-file.txt:2 router1 file ['7.7.7.7', '9.2.1.9']
fileformat_tr/router1/tr-file.txt:4 router1 file ['13.43.225.101', '125.125.126.117']
fileformat_tr/router1/tr-file.txt:3 router1 file ['11.11.11.21', '12.12.12.12']
fileformat_tr/router1/tr-file.txt:5 router1 file ['110.111.111.20', '123.196.116.200']
fileformat_tr/router1/tr-file.txt:7 router1 file ['170.171.111.20', '190.192.176.200']
fileformat_tr/router1/tr-file.txt:6 router1 file ['11.11.125.200', '20.20.02.20']
```

### 3.2.3 Sample 2

If a file stores a colon at the end of the timestamp and/or no semicolon at the end of the path, EtherTrace reads this file properly.

**Sample file**

```
2011-08-12 15:00:20: 17.7.7.7        9.2.1.9 13.13.3.100
2011-08-17 10:05:27: 111.11.11.21   12.12.12.12    113.13.31.2
2011-08-14 00:00:20: 113.43.225.101        125.125.126.117       110.111.12.13
2011-09-11 10:05:27: 110.111.111.20        123.196.116.200       1.1.1.1
2011-09-17 10:05:27: 11.11.125.200 20.20.02.20    15.26.26.18
2011-09-13 21:05:27: 170.171.111.20        190.192.176.200       15.26.26.18
```

Running script using the following usage:

zohrra@innisfree:~/EtherTrace-v0.1$ python tr.py -d fileformat_tr -k

In the following section we can see that files runs and generates output successfully, output is a sorted list of IP addresses.

```
Processing: fileformat_tr/router1/tr-file.txt

2011-08-12 15:00:20:    7.7.7.7 9.2.1.9 13.13.3.100

2011-08-17 10:05:27:    11.11.11.21     12.12.12.12     113.13.31.2

2011-08-14 00:00:20:    13.43.225.101   125.125.126.117 110.111.12.13

2011-09-11 10:05:27:    110.111.111.20  123.196.116.200 1.1.1.1

2011-09-17 10:05:27:   11.11.125.200   20.20.02.20     15.26.26.18

2011-09-13 21:05:27:    170.171.111.20  190.192.176.200 15.26.26.18

sorting the trs
done
fileformat_tr/router1/tr-file.txt:2 router1 file ['7.7.7.7', '9.2.1.9']
fileformat_tr/router1/tr-file.txt:4 router1 file ['13.43.225.101', '125.125.126.117']
fileformat_tr/router1/tr-file.txt:3 router1 file ['11.11.11.21', '12.12.12.12']
fileformat_tr/router1/tr-file.txt:5 router1 file ['110.111.111.20', '123.196.116.200']
fileformat_tr/router1/tr-file.txt:7 router1 file ['170.171.111.20', '190.192.176.200']
fileformat_tr/router1/tr-file.txt:6 router1 file ['11.11.125.200', '20.20.02.20']
```

### 3.2.4 Sample 3

If a file saves no colon at the end of the timestamp and/or no semicolon at the end of the path, Ethertrace reads this file successfully. Time stamp can be in hh:mm format and date can be just yyyy/mm/dd.

**Sample file**

```
2011/08/12 15:00      17.7.7.7          9.2.1.9 13.13.3.100
2011/08/17 10:05      111.11.11.21   12.12.12.12     113.13.31.2
2011/08/14 00:00      113.43.225.101          125.125.126.117         110.111.12.13
2011/09/11 10:05      110.111.111.20          123.196.116.200         1.1.1.1
```

```
2011/09/17 10:05      11.11.125.200 20.20.02.20     15.26.26.18
2011/09/13 21:05      170.171.111.20         190.192.176.200       15.26.26.18
```

Running the script using the following usage:

zohrra@innisfree:~/EtherTrace-v0.1$ python tr.py -d fileformat_tr -d

**Output**

In the following section we can observe that the output generates successfully, output is a sorted list of IP addresses.

```
Processing: fileformat_tr/router1/tr-file.txt

2011/08/12 15:00      17.7.7.7      9.2.1.9 13.13.3.100

2011/08/17 10:05      111.11.11.21    12.12.12.12     113.13.31.2

2011/08/14 00:00      113.43.225.101  125.125.126.117 110.111.12.13


2011/09/11 10:05      110.111.111.20  123.196.116.200 1.1.1.1

2011/09/17 10:05      11.11.125.200   20.20.02.20     15.26.26.18

2011/09/13 21:05      170.171.111.20  190.192.176.200 15.26.26.18

sorting the trs
done
fileformat_tr/router1/tr-file.txt:2 router1 file ['17.7.7.7', '9.2.1.9']
fileformat_tr/router1/tr-file.txt:4 router1 file ['113.43.225.101', '125.125.126.117']
fileformat_tr/router1/tr-file.txt:3 router1 file ['111.11.11.21', '12.12.12.12']
fileformat_tr/router1/tr-file.txt:5 router1 file ['110.111.111.20', '123.196.116.200']
fileformat_tr/router1/tr-file.txt:7 router1 file ['170.171.111.20', '190.192.176.200']
fileformat_tr/router1/tr-file.txt:6 router1 file ['11.11.125.200', '20.20.02.20']
```

## 3.3 Database information

When tr.py runs, it creates following four tables in the database that save information in different ways, a brief explanation of every table is presented.

1. l3_devs: This table assigns dev_id to every distinct IP address in the sorted file.

2. l3_tr: This table assigns tr_id (trace id) to every hop associated to a particular dev_id.
3. l3_tr_meta: This tables provides useful information about the trace results in a file, e.g. an invalid IP address or loop in a path etc.
4. l3_tr_sd: This table indicates the number of distinct source-destination devices pair.

## 3.4 Three different ways to run tr.py script

Tr.py script can be run in the following different ways, each way is meant to handle a specific file structure:

1. Usage 1: Recursively parses all the files from the root directory and populates the DB
2. Usage 2: Recursively parses the specified file populates the DB
3. Usage 3: Recursively parses the files listed in the specified file

The following section discusses Usage 1 and 2 in detail and explains how the values are populated in the database.

### 3.4.1 Usage 1 - Recursively parses all the files in the root directory.

This piece of code works when we create a directory e.g. "test_tr" into the code directory (ethertrace-v0.1) and run the script with the following parameters:
"python tr.py -d test_tr -k".

The directory "test_tr" contains another directory named "router1" (name of the source device) which includes a file named "tr-router2.txt" (router 2 is the name of the destination device). This is the file that contains all the traceroutes from node 1 to node 2 (or in other words router1 to router2).

Filename should be in a specific format i.e. 'tr-*filename*.txt' where only filename can be changed.

```
zohrra@innisfree:~/ethertrace-v0.1$ cd test_tr
zohrra@innisfree:~/ethertrace-v0.1/test_tr$ ls
router1
zohrra@innisfree:~/ethertrace-v0.1/test_tr$ cd router1
zohrra@innisfree:~/ethertrace-v0.1/test_tr/router1$ ls
tr-router2.txt
zohrra@innisfree:~/ethertrace-v0.1/test_tr/router1$
```

Figure 3.1

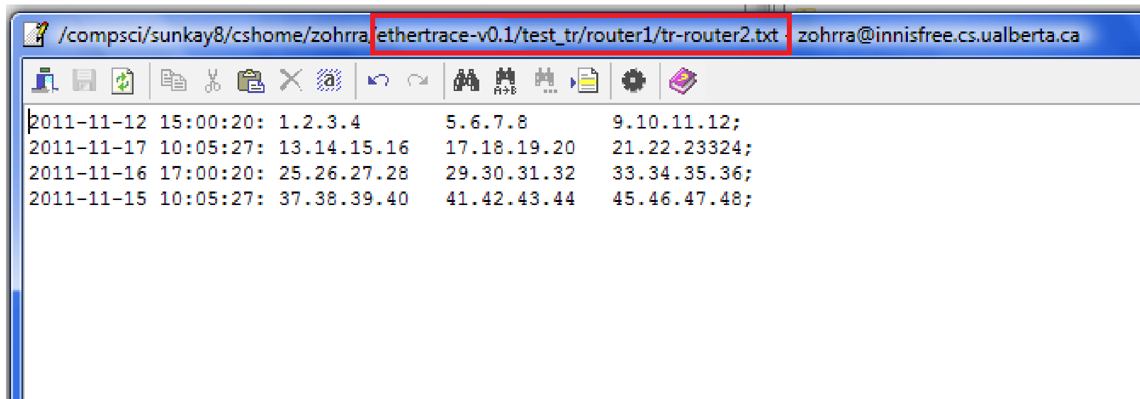



Figure 3.2

**Data file**

```
2011-11-12 15:00:20: 1.2.3.4 5.6.7.8 9.10.11.12;
2011-11-17 10:05:27: 13.14.15.16    17.18.19.20    21.22.23324;
2011-11-16 17:00:20: 25.26.27.28    29.30.31.32    33.34.35.36;
2011-11-15 10:05:27: 37.38.39.40    41.42.43.44    45.46.47.48;
```

Running the script

zohrra@innisfree:~/EtherTrace-v0.1$ python tr.py -d test_tr -k

**Output**

We can observe that when system starts processing this file it initially reads all the IP addresses row by row. Then it prints ‘sorting the trs –done’. In the original text file the rows not sorted by Date/Time, we can observe that the result displayed after ‘sorting the trs –done’ is now sorted by date-time (displays row in the following sequence: 1,4,3,2). Also, original file has three IP addresses in each row, when sorting is performed; it reads Total-1 (i.e. 3-1=2) IP addresses those are displayed in square brackets. All the IP addresses are valid otherwise they would have been replaced by an asterisk (*).

```
Processing: test_tr/router1/tr-router2.txt 2011-11-12 15:00:20: 1.2.3.4 5.6.7.8
9.10.11.12;

2011-11-17 10:05:27:   13.14.15.16    17.18.19.20    21.22.23324;

2011-11-16 17:00:20:   25.26.27.28    29.30.31.32    33.34.35.36;

2011-11-15 10:05:27:   37.38.39.40    41.42.43.44    45.46.47.48;
```

```
sorting the trs
done

test_tr/router1/tr-router2.txt:1 router1 router2 ['1.2.3.4', '5.6.7.8']
test_tr/router1/tr-router2.txt:4 router1 router2 ['37.38.39.40', '41.42.43.44']
test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32']
test_tr/router1/tr-router2.txt:2 router1 router2 ['13.14.15.16', '17.18.19.20']
```

**Database results**

**Usage 1 - l3_devs table**

When tr.py sorts the list, it displays four rows each containing 2 IP addresses, therefore we have 8 distinct IP addresses. In the above table dev_ids 0 and 1 are assigned to source and destination devices respectively, dev_Id 3 to dev_id 9 are assigned to each IP address in the order they appear in the sorted file.

| dev_id | dev_name |
|---|---|
| 0 | Router1 |
| 1 | router2 |
| 2 | 1.2.3.4 |
| 3 | 5.6.7.8 |
| 4 | 37.38.39.40 |
| 5 | 41.42.43.44 |
| 6 | 25.26.27.28 |
| 7 | 29.30.31.32 |
| 8 | 13.14.15.16 |
| 9 | 17.18.19.20 |

**Usage 1 - l3_tr table**

In this example there are four rows, each row has 2 IP addresses. IP addresses indicate the number of hops taken by the source device to reach to the destination device, therefore two IP addresses means two hops were taken to reach to the destination device. Hence the above table assigns hop_num to each distinct IP address in l3_devs table (0 to 1).

| **tr_id** | **hop_num** | **dev_id** |
|---|---|---|
| 0 | 0 | 2 |
| 0 | 1 | 3 |
| 1 | 0 | 4 |
| 1 | 1 | 5 |
| 2 | 0 | 6 |

| 2 | 1 | 7 |
|---|---|---|
| 3 | 0 | 8 |
| 3 | 1 | 9 |

tr_id represents row number in the sorted list, we have four rows 0-3. It means tr_Id=3, hop_num=1 should represent row number 4's second IP address i.e. '17.18.19.20'.

| 3 | 1 | 9 |
|---|---|---|

Row number 4 indicates row 4 after sorting is performed.

```
test_tr/router1/tr-router2.txt:1 router1 router2 ['1.2.3.4', '5.6.7.8']
test_tr/router1/tr-router2.txt:4 router1 router2 ['37.38.39.40', '41.42.43.44']
test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32']
test_tr/router1/tr-router2.txt:2 router1 router2 ['13.14.15.16', '17.18.19.20']
```

If we go back to l3_devs table we can do the cross check the dev_id assigned to this IP address and we can see that dev_id 13 is assigned to this IP address.

| 9 | 17.18.19.20 |
|---|---|

**Usage 1 - l3_tr_sd table**

This table displays the number of pairs of source and destination. In the above example we have only one source i.e. router 1 and only one destination i.e. router 2 therefore source_destination_pair_id is 0.

| sd_pair_id | src | dst |
|---|---|---|
| 0 | 0 | 1 |

**Usage 1 - l3_tr_meta table**

This table saves useful information, it's use will make more sense in the later examples. Here is a short description of it's fields:

- tr_id: It's a foreign key from l3_tr table.
- ts: Date and time for a trace route result.
- sd_pair_id: It's a foreign key from l3_tr_sd table and represents to which source-destination pair a particular trace_id belongs.
- tr_len: Number of IP addresses (or in other words # of hops) taken to reach to the destination IP address, in this example it is 2.
- valid_tr_len: Number of valid IP addresses taken to reach to the destination IP address, in this example it is 2.

- b_failed: Failed path, in this example it's 0.
- b_loopy: Number of loops created in a path, in this example it's 0 in all the rows as there are no loops generated in any paths.
- b_has_star_hop: Number of stars (asterisk) appeared in this traceroute. In this example it's 0 in all the rows as there are no *s generated in any paths.
- description: Description of each row in the file, directory path and the file name prints in reverse order (test_tr/router1/tr-router2.txt is displayed as txt.2retuor-rt/1retuor/ rt_tset).

| tr_id | ts | sd_pair_id | tr_len | valid_tr_len | b_failed | b_loopy | b_has_star_hop | description | hash_id |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011-11-12 15:00:20 | 0 | 2 | 2 | 0 | 0 | 0 | 1:txt.2retuor-rt/1retuor/rt_tset | 39568859 |
| 1 | 2011-11-15 10:05:27 | 0 | 2 | 2 | 0 | 0 | 0 | 4:txt.2retuor-rt/1retuor/rt_tset | 0728f405 |
| 2 | 2011-11-16 17:00:20 | 0 | 2 | 2 | 0 | 0 | 0 | 3:txt.2retuor-rt/1retuor/rt_tset | e7a3a891 |
| 3 | 2011-11-17 10:05:27 | 0 | 2 | 2 | 0 | 0 | 0 | :txt.2retuor-rt/1retuor/rt_tset | 65ca0bb3 |

### 3.4.2 Usage 2 - Recursively parses a specified file.

Filename 'traceulaberta.txt' is stored in ethertrace-v0.1 directory.

**Data**

```
2011-10-01 15:20:56    150.151.152.153  154.155.156.157  158.159.160.161  162.163.164.165;
2011-10-02 10:05:27    166.167.168.169  170.171.172.173  174.175.176.177  178.179.180.181;
2011-10-03 10:05:27    182.183.184.185  186.187.188.189  190.191.192.193  194.195.196.197;
2011-10-04 10:05:27    198.199.200.201  203.204.205.206  207.208.209.210  211.212.213.214;
2011-10-05 10:05:27    215.216.217.218  219.220.221.222  223.224.225.226  227.228.229.230;
```

Following snap shot ensures the file name, location and format:

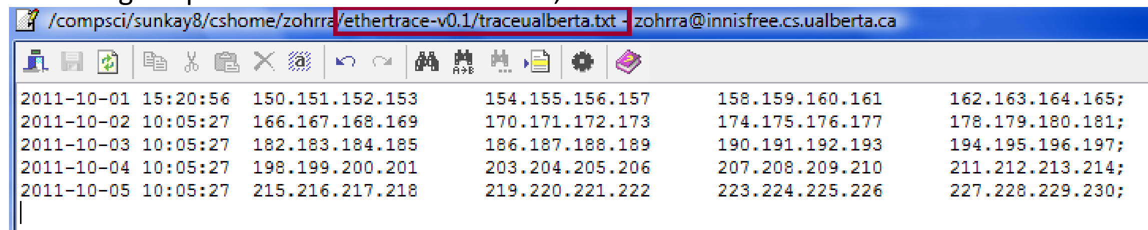



Figure 3.3: traceualberta.txt file

Running the script using following parameters:
tr.py -f traceualberta.txt:router1:router2 -k

**Output**

```
Processing:  traceualberta.txt  2011-10-01  15:20:56        150.151.152.153  154.155.156.157
158.159.160.161 162.163.164.165;
```

```
2011-10-02 10:05:27    166.167.168.169 170.171.172.173 174.175.176.177 178.179.180.181;

2011-10-03 10:05:27    182.183.184.185 186.187.188.189 190.191.192.193 194.195.196.197;

2011-10-04 10:05:27    198.199.200.201 203.204.205.206 207.208.209.210 211.212.213.214;

2011-10-05 10:05:27    215.216.217.218 219.220.221.222 223.224.225.226 227.228.229.230;

sorting the trs
done
traceualberta.txt:1 router1 router2 ['150.151.152.153', '154.155.156.157', '158.159.160.161']
traceualberta.txt:2 router1 router2 ['166.167.168.169', '170.171.172.173', '174.175.176.177']
traceualberta.txt:3 router1 router2 ['182.183.184.185', '186.187.188.189', '190.191.192.193']
traceualberta.txt:4 router1 router2 ['198.199.200.201', '203.204.205.206', '207.208.209.210']
traceualberta.txt:5 router1 router2 ['215.216.217.218', '219.220.221.222', '223.224.225.226']
```

We can observe that when the system starts processing this file it initially reads all the IP addresses line by line. Then it prints 'sorting the trs –done'. In this original text file all the lines are already in the order of date and time, therefore the result displayed after 'sorting the trs – done' is in the same order of actual file. Original file has four IP addresses in each row, when sorting is performed; it reads Total-1 (i.e. 4-1=3 in this example) IP addresses. Last IP address is supposed to be the destination device address.

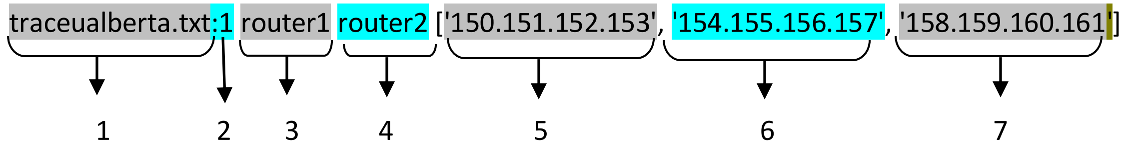



Figure 3.4

1 - file name
2 - line number in the actual text file (i.e. before sorting is performed)
3 - source name
4 - destination name
5 - First IP address in a specific line number
6 - Second IP address in a specific line number
7 - Second last IP address in a specific line number

**Database results**

Following are the records that create in the tables when tr.py script runs with the above file format and specified usage.

**Usage 2 - l3_devs table**

When tr.py sorts the file, it displays five rows each containing three IP addresses; therefore we have 15 distinct IP addresses. In the above table dev_ids 0 and 1 represent source and destination devices respectively. dev_Id 3 to dev_id 16 are assigned to each IP addresses in the order they appear in the sorted file.

| dev_id | dev_name |
|---|---|
| 0 | router1 |
| 1 | router2 |
| 2 | 150.151.152.153 |
| 3 | 154.155.156.157 |
| 4 | 158.159.160.161 |
| 5 | 166.167.168.169 |
| 6 | 170.171.172.173 |
| 7 | 174.175.176.177 |
| 8 | 182.183.184.185 |
| 9 | 186.187.188.189 |
| 10 | 190.191.192.193 |
| 11 | 198.199.200.201 |
| 12 | 203.204.205.206 |
| 13 | 207.208.209.210 |
| 14 | 215.216.217.218 |
| 15 | 219.220.221.222 |
| 16 | 223.224.225.226 |

**Usage 2 - l3_tr table**

In this example there are five rows and each row has three IP addresses. IP addresses indicate the number of hops source device took to reach to the destination device, therefore three IP addresses means three hops. Hence the above table assigns hop_num to each distinct IP address in l3_devs table (0 to 2).

| **tr_id** | **hop_num** | **dev_id** |
|---|---|---|
| 0 | 0 | 2 |
| 0 | 1 | 3 |
| 0 | 2 | 4 |
| 1 | 0 | 5 |
| 1 | 1 | 6 |
| 1 | 2 | 7 |
| 2 | 0 | 8 |
| 2 | 1 | 9 |

| 2 | 2 | 10 |
|---|---|---|
| 3 | 0 | 11 |
| 3 | 1 | 12 |
| 3 | 2 | 13 |
| 4 | 0 | 14 |
| 4 | 1 | 15 |
| 4 | 2 | 16 |

tr_id represents the row number, we have five rows 0-4. It means tr_Id=3, hop_num=2 should represent row number 4's 3rd IP address i.e. 207.208.209.210.

traceualberta.txt:4 router1 router2 ['198.199.200.201', '203.204.205.206', '207.208.209.210']

If we go back to l3_devs table we can check the dev_id assigned to this IP address:

| 13 | 207.208.209.210 |
|---|---|

**Usage 2 - l3_tr_sd table**

This table displays the number of pairs of source and destination. In the above example we have only one source i.e. router 1 and only one destination i.e. router 2 therefore source_destination_pair_id is 0.

| sd_pair_id | src | dst |
|---|---|---|
| 0 | 0 | 1 |

**Usage 2 - l3_tr_meta table**

| tr_id | ts | sd_pair_id | tr_len | valid_tr_len | b_failed | b_loopy | b_has_star_hop | description | hash_id |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011-10-01 15:20:56 | 0 | 3 | 3 | 0 | 0 | 0 | 1:txt.atreblauecart | e678378e |
| 1 | 2011-10-02 10:05:27 | 0 | 3 | 3 | 0 | 0 | 0 | 2:txt.atreblauecart | 6254a557 |
| 2 | 2011-10-03 10:05:27 | 0 | 3 | 3 | 0 | 0 | 0 | 3:txt.atreblauecart | b0c95847 |
| 3 | 2011-10-04 10:05:27 | 0 | 3 | 3 | 0 | 0 | 0 | 4:txt.atreblauecart | 1c06327c |
| 4 | 2011-10-05 10:05:27 | 0 | 3 | 3 | 0 | 0 | 0 | 5:txt.atreblauecart | 3e1847c9 |

## 3.5 Different scenarios

### 3.5.1 Case 1 - Invalid IP address in an input file

If an invalid IP address appears any where in the file, l3_tr_meta table's b_has_star_hop column indicates this error by displaying 1.

**Data**

```
2011-11-12 15:00:20: 1.2.3.4.142     5.6.7.8 9.10.11.12;
2011-11-17 10:05:27: 13.14.15.16     17.18.19.20     21.22.23324;
2011-11-16 17:00:20: 25.26.27.28     29.30.31.32     33.34.35.36;
2011-11-15 10:05:27: 37.38.39        41.42.43.44     45.46.47.48;
```

Following snapshot displays that first boxed IP address has more than four quads and second has less than four quads.

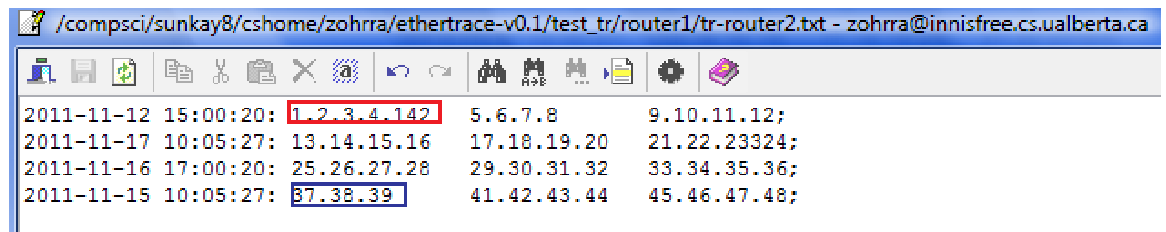



Figure 3.5 – Displays invalid IP addresses

Running the script
zohrra@innisfree:~/ethertrace-v0.1$ python tr.py -d test_tr -k

**Output**

The original file has two invalid IP addresses and they are replaced by an asterisk (*) in the sorted result.

```
Processing:  test_tr/router1/tr-router2.txt  2011-11-12  15:00:20:  1.2.3.4.142      5.6.7.8
9.10.11.12;

2011-11-17 10:05:27:   13.14.15.16     17.18.19.20     21.22.23324;

2011-11-16 17:00:20:   25.26.27.28     29.30.31.32     33.34.35.36;

2011-11-15 10:05:27:   37.38.39        41.42.43.44     45.46.47.48;

sorting the trs
done
```

```
test_tr/router1/tr-router2.txt:1 router1 router2 ['*', '5.6.7.8']
test_tr/router1/tr-router2.txt:4 router1 router2 ['*', '41.42.43.44']
test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32']
test_tr/router1/tr-router2.txt:2 router1 router2 ['13.14.15.16', '17.18.19.20']
```

```
zohrra@innisfree:~/ethertrace-v0.1$ python tr.py -d test_tr -k
tr.py:7: DeprecationWarning: the sets module is deprecated
  import sets
/compsci/sunkay8/cshome/zohrra/ethertrace-v0.1/tr_db.py:11: DeprecationWarning: the md5 module is deprecated; use hashlib instead
  import md5
Processing: test_tr/router1/tr-router2.txt 2011-11-12 15:00:20: 1.2.3.4.142    5.6.7.8 9.10.11.12;

2011-11-17 10:05:27:    13.14.15.16     17.18.19.20     21.22.23324;

2011-11-16 17:00:20:    25.26.27.28     29.30.31.32     33.34.35.36;

2011-11-15 10:05:27:    37.38.39        41.42.43.44     45.46.47.48;


sorting the trs
done
test_tr/router1/tr-router2.txt:1 router1 router2 ['*', '5.6.7.8']
test_tr/router1/tr-router2.txt:4 router1 router2 ['*', '41.42.43.44']
test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32']
test_tr/router1/tr-router2.txt:2 router1 router2 ['13.14.15.16', '17.18.19.20']
```

Figure 3.6 - Invalid IP addresses are replaced by asterisks

## Database result

### Case 1 - l3_devs table

When tr.py sorts the file, it displays four rows each containing two IP addresses. At two places asterisks appear in the result. We have 6 distinct IP addresses. In the table, dev_id 0 and 1 are assigned to source and destination devices respectively, dev_Id 2 to 7 are assigned to six IP addresses in the order they appear in the sorted file. We can observe that no dev_id is assigned to invalid IP addresses (*s).

| dev_id | dev_name |
|---|---|
| 0 | Router1 |
| 1 | router2 |
| 2 | 5.6.7.8 |
| 3 | 41.42.43.44 |
| 4 | 25.26.27.28 |
| 5 | 29.30.31.32 |
| 6 | 13.14.15.16 |
| 7 | 17.18.19.20 |

### Case 1 - l3_tr table

In this example we have four rows and each row has two IP addresses. There are two invalid IP addresses that appear in row 1 and 2. IP addresses indicate the number of hops taken to reach to the destination device, therefore two IP addresses means two hops were taken to reach to the destination device. This table assigns hop_num to each distinct IP address in l3_devs table (0 to 1). tr_id represents the row number, we can

notice that tr_ids 0 and 1 have only one hop that has value 1, it means the first hop is missing; all other tr_ids have two hops 0 and 1.

| tr_id | hop_num | dev_id |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 1 | 3 |
| 2 | 0 | 4 |
| 2 | 1 | 5 |
| 3 | 0 | 6 |
| 3 | 1 | 7 |

**Case 1 - l3_tr_sd table**

This table displays the number of pairs of source and destination. In the above example we have only one source i.e. router 1 and only one destination i.e. router 2 therefore source_destination_pair_id is 0.

| sd_pair_id | src | dst |
|---|---|---|
| 0 | 0 | 1 |

**Case 1 - l3_tr_meta table**

b_has_star_hop displays number of stars (asterisk) appeared in this traceroute. In this example the first two rows has 1 in this field.

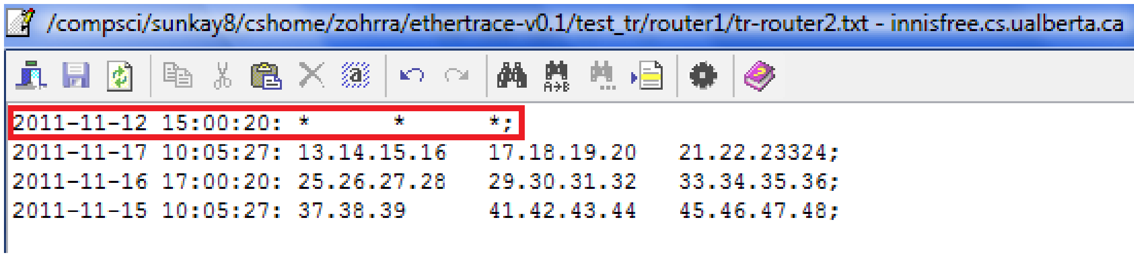
| tr_ id | ts | sd_pair_id | tr_len | valid_ tr_len | b_failed | b_loop y | b_has_ star_hop | description | hash_id |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011-11-12 15:00:20 | 0 | 2 | 2 | 0 | 0 | 1 | 1:txt.2retuor-rt/1retuor/rt_tset | 1906239 1 |
| 1 | 2011-11-15 10:05:27 | 0 | 2 | 2 | 0 | 0 | 1 | 4:txt.2retuor-rt/1retuor/rt_tset | bc84841 6 |
| 2 | 2011-11-16 17:00:20 | 0 | 2 | 2 | 0 | 0 | 0 | 3:txt.2retuor-rt/1retuor/rt_tset | e7a3a89 1 |
| 3 | 2011-11-17 10:05:27 | 0 | 2 | 2 | 0 | 0 | 0 | 2:txt.2retuorrt/1retuor/rt _tset | 65ca0bb 3 |

### 3.5.2 Case 2 - A failed traceroute result

If a failed path exists somewhere in the file, l3_tr_meta table's b_failed column indicates this error by displaying 1.

**Data**

```
2011-11-12 15:00:20: *       *       *;
2011-11-17 10:05:27: 13.14.15.16    17.18.19.20    21.22.23324;
2011-11-16 17:00:20: 25.26.27.28    29.30.31.32    33.34.35.36;
2011-11-15 10:05:27: 37.38.39       41.42.43.44    45.46.47.48;
```


/compsci/sunkay8/cshome/zohrra/ethertrace-v0.1/test_tr/router1/tr-router2.txt - innisfree.cs.ualberta.ca

```
2011-11-12 15:00:20: *          *          *;
2011-11-17 10:05:27: 13.14.15.16    17.18.19.20    21.22.23324;
2011-11-16 17:00:20: 25.26.27.28    29.30.31.32    33.34.35.36;
2011-11-15 10:05:27: 37.38.39       41.42.43.44    45.46.47.48;
```

Figure 3.7 – Failed path

**Output**

```
Processing: test_tr/router1/tr-router2.txt 2011-11-12 15:00:20: *      *      *
;

2011-11-17 10:05:27:   13.14.15.16    17.18.19.20    21.22.23324;

2011-11-16 17:00:20:   25.26.27.28    29.30.31.32    33.34.35.36;

2011-11-15 10:05:27:   37.38.39       41.42.43.44    45.46.47.48;


sorting the trs
done
test_tr/router1/tr-router2.txt:1 router1 router2 ['*', '*']
test_tr/router1/tr-router2.txt:4 router1 router2 ['*', '41.42.43.44']
test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32']
test_tr/router1/tr-router2.txt:2 router1 router2 ['13.14.15.16', '17.18.19.20']
```

### Case 2 - l3_tr_meta table

It can be observed that the b_failed column displays 1 in the corresponding path. Also, b_has_star_hop also displays 1 in first two columns as expected.

| tr_ id | ts | sd_pair_id | tr_len | valid_ tr_len | b_failed | b_loop y | b_has_ star_hop | description | hash_id |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011-11-12 15:00:20 | | 2 | 2 | 1 | 0 | 1 | 1:txt.2retuor-rt/1retuor/rt_tset | 2950f9ec |
| 1 | 2011-11-15 10:05:27 | | 2 | 2 | 0 | 0 | 1 | 4:txt.2retuor-rt/1retuor/rt_tset | bc848416 |
| 2 | 2011-11-16 17:00:20 | | 2 | 2 | 0 | 0 | 0 | 3:txt.2retuor-rt/1retuor/rt_tset | e7a3a891 |
| 3 | 2011-11-17 10:05:27 | | 2 | 2 | 0 | 0 | 0 | 2:txt.2retuor-rt/1retuor/rt_tset | 65ca0bb3 |

```
| tr_id | ts                  | sd_pair_id | tr_len | valid_tr_len | b_failed | b_loopy | b_has_star_hop | description                      | hash_id
|     0 | 2011-11-12 15:00:20 |          0 |      2 |            2 |        1 |       0 |              1 | 1:txt.2retuor-rt/1retuor/rt_tset | 2950f9ec
|     1 | 2011-11-15 10:05:27 |          0 |      2 |            2 |        0 |       0 |              1 | 4:txt.2retuor-rt/1retuor/rt_tset | bc848416
|     2 | 2011-11-16 17:00:20 |          0 |      2 |            2 |        0 |       0 |              0 | 3:txt.2retuor-rt/1retuor/rt_tset | e7a3a891
|     3 | 2011-11-17 10:05:27 |          0 |      2 |            2 |        0 |       0 |              0 | 2:txt.2retuor-rt/1retuor/rt_tset | 65ca0bb3
4 rows in set (0.00 sec)
```

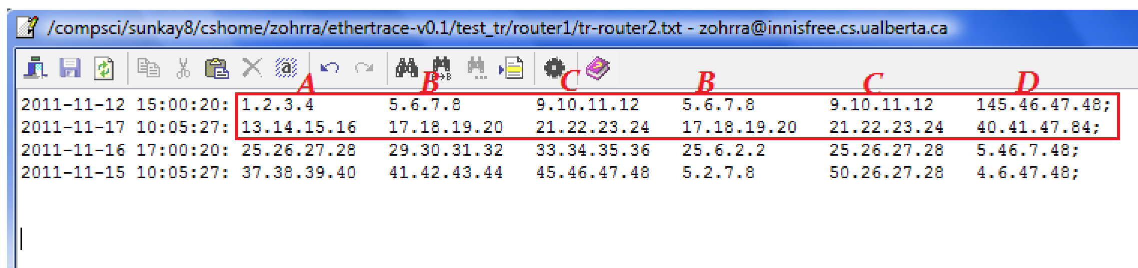
Figure 3.8

### 3.5.3 Case 3 - A loop is created in a path

If a loop is created in any path in the file, l3_tr_meta table's b_loopy column indicates it by displaying 1.

**Data**

```
2011-11-12 15:00:20: 1.2.3.4    5.6.7.8    9.10.11.12 5.6.7.8    9.10.11.12 145.46.47.48;
2011-11-17 10:05:27: 13.14.15.16    17.18.19.20    21.22.23.24    17.18.19.20    21.22.23.24
        40.41.47.84;
2011-11-16 17:00:20: 25.26.27.28    29.30.31.32    33.34.35.36    25.6.2.2    25.26.27.28    5.46.7.48;
2011-11-15 10:05:27: 37.38.39.40    41.42.43.44    45.46.47.48    5.2.7.8    50.26.27.28    4.6.47.48;
```

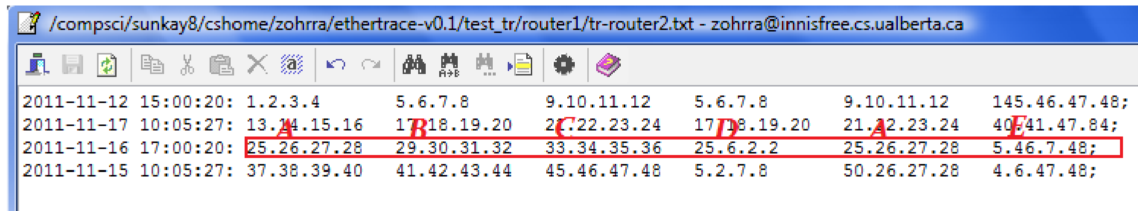


/compsci/sunkay8/cshome/zohrra/ethertrace-v0.1/test_tr/router1/tr-router2.txt - zohrra@innisfree.cs.ualberta.ca

```
2011-11-12 15:00:20: 1.2.3.4      5.6.7.8      9.10.11.12   5.6.7.8      9.10.11.12   145.46.47.48;
2011-11-17 10:05:27: 13.14.15.16  17.18.19.20  21.22.23.24  17.18.19.20  21.22.23.24  40.41.47.84;
2011-11-16 17:00:20: 25.26.27.28  29.30.31.32  33.34.35.36  25.6.2.2     25.26.27.28  5.46.7.48;
2011-11-15 10:05:27: 37.38.39.40  41.42.43.44  45.46.47.48  5.2.7.8      50.26.27.28  4.6.47.48;
```



Figure 3.9 – Each row in the box creates a loop (ABCBCD)



/compsci/sunkay8/cshome/zohrra/ethertrace-v0.1/test_tr/router1/tr-router2.txt - zohrra@innisfree.cs.ualberta.ca

```
2011-11-12 15:00:20: 1.2.3.4      5.6.7.8      9.10.11.12   5.6.7.8      9.10.11.12   145.46.47.48;
2011-11-17 10:05:27: 13.14.15.16  17.18.19.20  21.22.23.24  17.18.19.20  21.22.23.24  40.41.47.84;
2011-11-16 17:00:20: 25.26.27.28  29.30.31.32  33.34.35.36  25.6.2.2     25.26.27.28  5.46.7.48;
2011-11-15 10:05:27: 37.38.39.40  41.42.43.44  45.46.47.48  5.2.7.8      50.26.27.28  4.6.47.48;
```



Figure 3.10 - Row in the box creates a loop (ABCDAE)

Running the script

zohrra@innisfree:~/EtherTrace-v0.1$ python tr.py -d test_tr -k

**Output**

We can observe that when the system starts processing this file it initially reads all the IP addresses row by row. Then it prints 'sorting the trs –done'. In the original text file the rows not sorted by Date/Time, we can observe that the result displayed after 'sorting the trs –done' is now sorted by date-time (displays row in the following sequence: 1,4,3,2). Also, original file has six IP addresses in each row, when sorting is performed; it reads Total-1 (i.e. 6-1=5) IP addresses that are displayed in square brackets. All the IP addresses are valid otherwise they would have replaced by an asterisk (*).

```
Processing: test_tr/router1/tr-router2.txt 2011-11-12 15:00:20: 1.2.3.4 5.6.7.8
9.10.11.12 5.6.7.8 9.10.11.12 145.46.47.48;
```

```
2011-11-17  10:05:27:    13.14.15.16    17.18.19.20    21.22.23.24    17.18.19.20
21.22.23.24   40.41.47.84;

2011-11-16  17:00:20:    25.26.27.28    29.30.31.32    33.34.35.36    25.6.2.2
25.26.27.28   5.46.7.48;

2011-11-15 10:05:27:   37.38.39.40   41.42.43.44   45.46.47.48   5.2.7.8 50.26.27.28
4.6.47.48;

sorting the trs
done
test_tr/router1/tr-router2.txt:1   router1   router2   ['1.2.3.4',  '5.6.7.8',  '9.10.11.12',
'5.6.7.8', '9.10.11.12']
test_tr/router1/tr-router2.txt:4   router1   router2   ['37.38.39.40',   '41.42.43.44',
'45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router2.txt:3   router1   router2   ['25.26.27.28',   '29.30.31.32',
'33.34.35.36', '25.6.2.2', '25.26.27.28']
test_tr/router1/tr-router2.txt:2   router1   router2   ['13.14.15.16',   '17.18.19.20',
'21.22.23.24', '17.18.19.20', '21.22.23.24']
```

**Database results**

**Case 3 - l3_devs table**

| dev_id | dev_name |
|---|---|
| 0 | Router1 |
| 1 | router2 |
| 2 | 1.2.3.4 |
| 3 | 5.6.7.8 |
| 4 | 9.10.11.12 |
| 5 | 37.38.39.40 |
| 6 | 41.42.43.44 |
| 7 | 45.46.47.48 |
| 8 | 5.2.7.8 |
| 9 | 50.26.27.28 |
| 10 | 25.26.27.28 |
| 11 | 29.30.31.32 |
| 12 | 33.34.35.36 |
| 13 | 25.6.2.2 |
| 14 | 13.14.15.16 |
| 15 | 17.18.19.20 |

| 16 | 21.22.23.24 |
|---|---|

When tr.py sorts the file, it displays four rows each containing five IP addresses; total number of IP addresses are 20 but the count of distinct IP addresses is only 15 as underlined below.

test_tr/router1/tr-router2.txt:1 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '5.6.7.8', '9.10.11.12']
test_tr/router1/tr-router2.txt:4 router1 router2 ['37.38.39.40', '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32', '33.34.35.36', '25.6.2.2', '25.26.27.28']
test_tr/router1/tr-router2.txt:2 router1 router2 ['13.14.15.16', '17.18.19.20', '21.22.23.24', '17.18.19.20', '21.22.23.24']

In the l3_devs table, dev_ids 0 and 1 represent source and destination devices respectively, dev_Id 2 to 16 are assigned to each distinct IP addresses in the order they appear in the sorted file.

**Case 3 - l3_tr table**

As mentioned earlier that in the example under consideration has four rows and each row has five IP addresses. IP addresses indicate the number of hops it took to reach to the destination device, therefore five IP addresses means five hops were taken to reach to the destination device. Hence the above table assigns hop_num to each distinct IP address in l3_devs table (0 to 4).

| tr_id | hop_num | dev_id |
|---|---|---|
| 0 | 0 | 2 |
| 0 | 1 | 3 |
| 0 | 2 | 4 |
| 0 | 3 | 3 |
| 0 | 4 | 4 |
| 1 | 0 | 5 |
| 1 | 1 | 6 |
| 1 | 2 | 7 |
| 1 | 3 | 8 |
| 1 | 4 | 9 |
| 2 | 0 | 10 |
| 2 | 1 | 11 |
| 2 | 2 | 12 |
| 2 | 3 | 13 |

| 2 | 4 | 10 |
|---|---|---|
| 3 | 0 | 14 |
| 3 | 1 | 15 |
| 3 | 2 | 16 |
| 3 | 3 | 15 |
| 3 | 4 | 16 |

tr_id represents the row number, we have four tr_Id (0-3) and each tr_id is assigned to five rows. Hence we have all together 4x5=20 rows.

It means tr_Id=1, hop_num=4 should represent row number 2's fifth IP address i.e. '50.26.27.28'.

| 1 | 4 | 9 |
|---|---|---|

Row number 2 indicates row 2 after sorting is performed.

test_tr/router1/tr-router2.txt:1 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '5.6.7.8', '9.10.11.12']
test_tr/router1/tr-router2.txt:4 router1 router2 ['37.38.39.40', '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32', '33.34.35.36', '25.6.2.2', '25.26.27.28']
test_tr/router1/tr-router2.txt:2 router1 router2 ['13.14.15.16', '17.18.19.20', '21.22.23.24', '17.18.19.20', '21.22.23.24']

If we go back to l3_devs table we can do the cross check the dev_id assigned to this IP address and we can see that dev_id 13 is assigned to this IP address.

| 9 | 50.26.27.28 |
|---|---|

**Case 3 - l3_tr_sd table**

This table displays the number of pairs of source and destination. In the above example we have only one source i.e. router 1 and only one destination i.e. router 2 therefore source_destination_pair_id is 0.

| sd_pair_id | src | dst |
|---|---|---|
| 0 | 0 | 1 |

**Case 3 - l3_tr_meta table**

B-loopy displays the number of loops created in a traceroute, in this example it represents 1 in 3 rows where tr_id = 0, 2 and 3 as there are loops generated in these paths.

| tr_ id | ts | sd_pair_id | tr_len | valid_ tr_len | b_failed | b_loop y | b_has_ star_hop | description | hash_id |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011-11-12 15:00:20 | 0 | 5 | 5 | 0 | 1 | 0 | 1:txt.2retuor-rt/1retuor/rt_tset | 116bb2b 9 |
| 1 | 2011-11-15 10:05:27 | 0 | 5 | 5 | 0 | 0 | 0 | 4:txt.2retuor-rt/1retuor/rt_tset | e1c3898 6 |
| 2 | 2011-11-16 17:00:20 | 0 | 5 | 5 | 0 | 1 | 0 | 3:txt.2retuor-rt/1retuor/rt_tset | e52edf6 1 |
| 3 | 2011-11-17 10:05:27 | 0 | 5 | 5 | 0 | 1 | 0 | 2:txt.2retuor-rt/1retuor/rt_tset | e6802e5 5 |

```
mysql> select * from l3_tr_meta;
+-------+---------------------+------------+--------+---------------+----------+---------+----------------+----------------------------------+---------
| tr_id | ts                  | sd_pair_id | tr_len | valid_tr_len  | b_failed | b_loopy | b_has_star_hop | description                      | hash_id
+-------+---------------------+------------+--------+---------------+----------+---------+----------------+----------------------------------+---------
|     0 | 2011-11-12 15:00:20 |          0 |      5 |             5 |        0 |       1 |              0 | 1:txt.2retuor-rt/1retuor/rt_tset | 116bb2b
|     1 | 2011-11-15 10:05:27 |          0 |      5 |             5 |        0 |       0 |              0 | 4:txt.2retuor-rt/1retuor/rt_tset | e1c3898
|     2 | 2011-11-16 17:00:20 |          0 |      5 |             5 |        0 |       1 |              0 | 3:txt.2retuor-rt/1retuor/rt_tset | e52edf6
|     3 | 2011-11-17 10:05:27 |          0 |      5 |             5 |        0 |       1 |              0 | 2:txt.2retuor-rt/1retuor/rt_tset | e6802e5
+-------+---------------------+------------+--------+---------------+----------+---------+----------------+----------------------------------+---------
4 rows in set (0.00 sec)
```

Figure 3.11 – b-loopy contains 1

## 3.5.4 Case 4 - Redundant paths in a file

If a file contains one path (sequence of IP addresses) more than once, EtherTrace keeps record of only one path because EtherTrace is designed to trace distinct IP edges that traverse any given layer-2 edge.

**Data**

```
2011-11-12 15:00:20: 1.2.3.4     5.6.7.8     9.10.11.12 15.16.17.18     19.20.21.22     145.46.47.48;
2011-11-17 10:05:27: 13.14.15.16             17.18.19.20     21.22.23.24     7.8.9.20     21.22.23.24
                     40.41.47.84;
2011-11-16 17:00:20: 25.26.27.28             29.30.31.32     33.34.35.36     25.6.2.2     5.6.7.2     5.46.7.48;
2011-11-15 10:05:27: 37.38.39.40             41.42.43.44     45.46.47.48     5.2.7.8      50.26.27.28     4.6.47.48;
2011-11-25 15:00:20: 1.2.3.4     5.6.7.8     9.10.11.12 15.16.17.18     19.20.21.22     145.46.47.48;
2011-11-01 10:05:27: 37.38.39.40             41.42.43.44     45.46.47.48     5.2.7.8      50.26.27.28     4.6.47.48;
2011-11-30 15:00:20: 1.2.3.4     5.6.7.8     9.10.11.12 15.16.17.18     19.20.21.22     145.46.47.48;
```

Please note that rows 1, 5 and 7 have different date-time but the path taken to destination is same.
Rows 4 and 6 also have different date-time but the path taken to the destination is same.
Hence there are 7 rows in this file, but only 4 distinct paths exist.

```
/compsci/sunkay8/cshome/zohrra/ethertrace-v0.1/test_tr/router1/tr-router2.txt - zohrra@innisfree.cs.ualberta.ca
2011-11-12 15:00:20: 1.2.3.4      5.6.7.8      9.10.11.12   15.16.17.18  19.20.21.22  145.46.47.48;
2011-11-17 10:05:27: 13.14.15.16  17.18.19.20  21.22.23.24  7.8.9.20     21.22.23.24  40.41.47.84;
2011-11-16 17:00:20: 25.26.27.28  29.30.31.32  33.34.35.36  25.6.2.2     5.6.7.2      5.46.7.48;
2011-11-15 10:05:27: 37.38.39.40  41.42.43.44  45.46.47.48  5.2.7.8      50.26.27.28  4.6.47.48;
2011-11-25 15:00:20: 1.2.3.4      5.6.7.8      9.10.11.12   15.16.17.18  19.20.21.22  145.46.47.48;
2011-11-01 10:05:27: 37.38.39.40  41.42.43.44  45.46.47.48  5.2.7.8      50.26.27.28  4.6.47.48;
2011-11-30 15:00:20: 1.2.3.4      5.6.7.8      9.10.11.12   15.16.17.18  19.20.21.22  145.46.47.48;
```

Figure 3.12

Running the script

zohrra@innisfree:~/EtherTrace-v0.1$ python tr.py -d test_tr -k

**Output**

It can be observed that every row is picked up and printed in the sorted file regardless of their redundant appearance in the file.

```
Processing: test_tr/router1/tr-router2.txt 2011-11-12 15:00:20: 1.2.3.4 5.6.7.8 9.10.11.12 15.16.17.18 19.20.21.22 145.46.47.48;

2011-11-17 10:05:27: 13.14.15.16 17.18.19.20 21.22.23.24 7.8.9.20 21.22.23.24 40.41.47.84;

2011-11-16 17:00:20: 25.26.27.28 29.30.31.32 33.34.35.36 25.6.2.2 5.6.7.2 5.46.7.48;

2011-11-15 10:05:27: 37.38.39.40 41.42.43.44 45.46.47.48 5.2.7.8 50.26.27.28 4.6.47.48;

2011-11-25 15:00:20: 1.2.3.4 5.6.7.8 9.10.11.12 15.16.17.18 19.20.21.22 145.46.47.48;

2011-11-01 10:05:27: 37.38.39.40 41.42.43.44 45.46.47.48 5.2.7.8 50.26.27.28 4.6.47.48;

2011-11-30 15:00:20: 1.2.3.4 5.6.7.8 9.10.11.12 15.16.17.18 19.20.21.22 145.46.47.48;

sorting the trs
done
test_tr/router1/tr-router2.txt:6 router1 router2 ['37.38.39.40', '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router2.txt:1 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '15.16.17.18', '19.20.21.22']
```

```
test_tr/router1/tr-router2.txt:4    router1    router2    ['37.38.39.40',    '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router2.txt:3    router1    router2    ['25.26.27.28',    '29.30.31.32', '33.34.35.36', '25.6.2.2', '5.6.7.2']
test_tr/router1/tr-router2.txt:2    router1    router2    ['13.14.15.16',    '17.18.19.20', '21.22.23.24', '7.8.9.20', '21.22.23.24']
test_tr/router1/tr-router2.txt:5    router1    router2    ['1.2.3.4',    '5.6.7.8',    '9.10.11.12', '15.16.17.18', '19.20.21.22']
test_tr/router1/tr-router2.txt:7    router1    router2    ['1.2.3.4',    '5.6.7.8',    '9.10.11.12', '15.16.17.18', '19.20.21.22']
```

**Case 4 - l3_devs table**

| dev_id | dev_name |
|---|---|
| 0 | router1 |
| 1 | router2 |
| 2 | 37.38.39.40 |
| 3 | 41.42.43.44 |
| 4 | 45.46.47.48 |
| 5 | 5.2.7.8 |
| 6 | 50.26.27.28 |
| 7 | 1.2.3.4 |
| 8 | 5.6.7.8 |
| 9 | 9.10.11.12 |
| 10 | 15.16.17.18 |
| 11 | 19.20.21.22 |
| 12 | 25.26.27.28 |
| 13 | 29.30.31.32 |
| 14 | 33.34.35.36 |
| 15 | 25.6.2.2 |
| 16 | 5.6.7.2 |
| 17 | 13.14.15.16 |
| 18 | 17.18.19.20 |
| 19 | 21.22.23.24 |
| 20 | 7.8.9.20 |

When tr.py sorts the file, it displays seven rows each containing five IP addresses; total number of IP addresses are 35 but the count of distinct IP addresses is only 19 as underlined below.

test_tr/router1/tr-router2.txt:6 router1 router2 ['37.38.39.40', '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router2.txt:1 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '15.16.17.18', '19.20.21.22']
test_tr/router1/tr-router2.txt:4 router1 router2 ['37.38.39.40', '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32', '33.34.35.36', '25.6.2.2', '5.6.7.2']
test_tr/router1/tr-router2.txt:2 router1 router2 ['13.14.15.16', '17.18.19.20', '21.22.23.24', '7.8.9.20', '21.22.23.24']
test_tr/router1/tr-router2.txt:5 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '15.16.17.18', '19.20.21.22']
test_tr/router1/tr-router2.txt:7 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '15.16.17.18', '19.20.21.22']

In the l3_devs table, dev_id 0 and dev _id 1 represent source and destination devices respectively, dev_Id 2 to dev_id 20 are assigned to each distinct IP addresses in the order they appear in the sorted file.

**Case 4 - l3_tr table**

| tr_id | hop_num | dev_id |
|---|---|---|
| 0 | 0 | 2 |
| 0 | 1 | 3 |
| 0 | 2 | 4 |
| 0 | 3 | 5 |
| 0 | 4 | 6 |
| 1 | 0 | 12 |
| 1 | 1 | 13 |
| 1 | 2 | 14 |
| 1 | 3 | 15 |
| 1 | 4 | 16 |
| 2 | 0 | 17 |
| 2 | 1 | 18 |
| 2 | 2 | 19 |
| 2 | 3 | 20 |
| 2 | 4 | 19 |
| 3 | 0 | 7 |
| 3 | 1 | 8 |
| 3 | 2 | 9 |

| | | |
|---|---|---|
| 3 | 3 | 10 |
| 3 | 4 | 11 |

In this example there are seven rows and each row has five IP addresses. Excluding the rows with redundant trace results there are four rows. IP addresses indicate the number of hops it took to reach to the destination device, therefore five IP addresses means five hops were taken to reach to the destination device. Hence the above table assigns hop_num to each distinct IP address in l3_devs table (0 to 4). Tr_id represents the row number, we have four tr_Id (0-3) and each tr_id is assigned to five rows. Hence we have all together 4x5=20 rows.

It means tr_Id=2, hop_num=0 should represent row number 3's first IP address i.e. '25.26.27.28'.

| | | |
|---|---|---|
| 2 | 0 | 17 |

Row number 3 indicates row # 3 in the sorted list.

EtherTrace code is designed to read the latest record when there are redundant paths; in order to display the latest records that would be picked by the code we've strikenthrough the older redundant records in the original file:

~~2011-11-12 15:00:20: 1.2.3.4 5.6.7.8 9.10.11.12 15.16.17.18 19.20.21.22 145.46.47.48;~~<br>
2011-11-17 10:05:27: 13.14.15.16 17.18.19.20 21.22.23.24 7.8.9.20 21.22.23.24 40.41.47.84;<br>
2011-11-16 17:00:20: 25.26.27.28 29.30.31.32 33.34.35.36 25.6.2.2 5.6.7.2 5.46.7.48;<br>
2011-11-15 10:05:27: 37.38.39.40 41.42.43.44 45.46.47.48 5.2.7.8 50.26.27.28 4.6.47.48;<br>
~~2011-11-25 15:00:20: 1.2.3.4 5.6.7.8 9.10.11.12 15.16.17.18 19.20.21.22 145.46.47.48;~~<br>
~~2011-11-01 10:05:27: 37.38.39.40 41.42.43.44 45.46.47.48 5.2.7.8 50.26.27.28 4.6.47.48;~~<br>
2011-11-30 15:00:20: 1.2.3.4 5.6.7.8 9.10.11.12 15.16.17.18 19.20.21.22 145.46.47.48

The sorted file doesn't indicate date-time, however it sorts in increasing order of date-time. So we've strikenthrough the older records, we can see that we are left with four records that are shown in the following section.

~~test_tr/router1/tr-router2.txt:6 router1 router2 ['37.38.39.40', '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']~~<br>
~~test_tr/router1/tr-router2.txt:1 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '15.16.17.18', '19.20.21.22']~~<br>
test_tr/router1/tr-router2.txt:4 router1 router2 ['37.38.39.40', '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']

test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32', '33.34.35.36', '25.6.2.2', '5.6.7.2']<br>
test_tr/router1/tr-router2.txt:2 router1 router2 ['13.14.15.16', '17.18.19.20', '21.22.23.24', '7.8.9.20', '21.22.23.24']

~~test_tr/router1/tr-router2.txt:5 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '15.16.17.18', '19.20.21.22']~~
test_tr/router1/tr-router2.txt:7 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '15.16.17.18', '19.20.21.22']

The following section displays the short listed four records that will be saved in the database; now we can indicate the third row and first IP address:

test_tr/router1/tr-router2.txt:4 router1 router2 ['37.38.39.40', '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32', '33.34.35.36', '25.6.2.2', '5.6.7.2']
test_tr/router1/tr-router2.txt:2 router1 router2 [**'13.14.15.16'**, '17.18.19.20', '21.22.23.24', '7.8.9.20', '21.22.23.24']
test_tr/router1/tr-router2.txt:7 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '15.16.17.18', '19.20.21.22']

If we go back to l3_devs table we can cross check the dev_id assigned to this IP address and we see that dev_id 17 is assigned to this IP address, hence verified!

| 17 | 13.14.15.16 |
|---|---|

**Case 4 - l3_tr_sd table**

This table displays the number of pairs of source and destination. In the above example we have only one source i.e. router 1 and only one destination i.e. router 2 therefore source_destination_pair_id is 0.

| sd_pair_id | src | dst |
|---|---|---|
| 0 | 0 | 1 |

**Case 4 - l3_tr_meta table**

We can observe that the most recent records display on the meta table:

| tr_id | ts | sd_pair_id | tr_len | valid_tr_len | b_failed | b_loopy | b_has_star_hop | description | hash_id |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011-11-15 10:05:27 | 0 | 5 | 5 | 0 | 0 | 0 | 4:txt.2retuor-rt/1retuor/rt_tset | e1c38986 |
| 3 | 2011-11-30 15:00: | 0 | 5 | 5 | 0 | 0 | 0 | 7:txt.2retuor-rt/1retuor/rt_tset | 2efa6894 |
| 1 | 2011-11-16 17:00:20 | 0 | 5 | 5 | 0 | 0 | 0 | 3:txt.2retuor-rt/1retuor/rt_tset | 19c5ffa8 |
| 2 | 2011-11-17 10:05:27 | 0 | 5 | 5 | 0 | 1 | 0 | :txt.2retuor-rt/1retuor/rt_tset | 95596746 |

### 3.5.5 Case 5 - More than one source-destination pair

In all the above examples we saved only one file in the directory 'router1' so l3_tr_sd table has only one record that indicates a pair of source-destination.

In the following example we can see how to have more than one record in this table. In other words we can have more than one source to destination pairs.

Router 1 that's the directory name is the source.

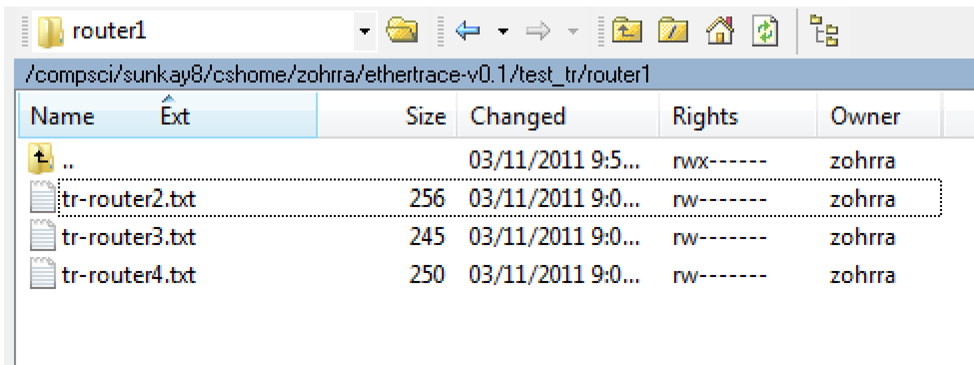
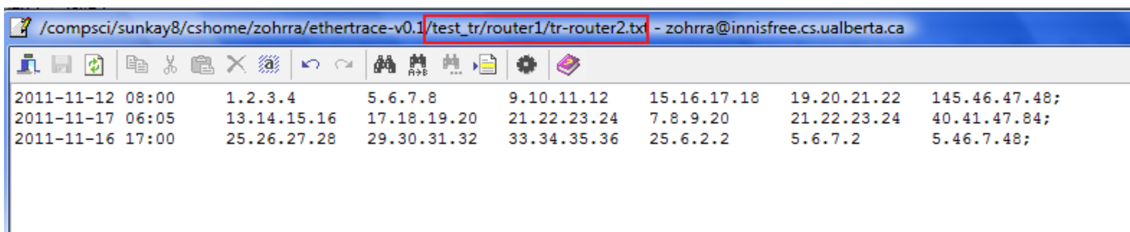



Figure 3.13 – Directory 'router 1 has three .txt files

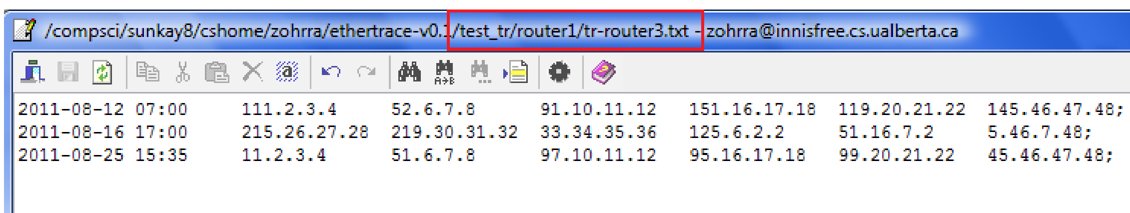
Router2 - tr-router2.txt is a file saved in the directory named 'router1' and it's one of the destination devices.




```
2011-11-12 08:00    1.2.3.4       5.6.7.8       9.10.11.12    15.16.17.18   19.20.21.22   145.46.47.48;
2011-11-17 06:05    13.14.15.16   17.18.19.20   21.22.23.24   7.8.9.20      21.22.23.24   40.41.47.84;
2011-11-16 17:00    25.26.27.28   29.30.31.32   33.34.35.36   25.6.2.2      5.6.7.2       5.46.7.48;
```

Figure 3.14 – Data saved in tr-router2.txt

Router 3 - tr-router3.txt is a file saved in the directory named 'router1' is another destination device.




```
2011-08-12 07:00    111.2.3.4     52.6.7.8      91.10.11.12   151.16.17.18  119.20.21.22  145.46.47.48;
2011-08-16 17:00    215.26.27.28  219.30.31.32  33.34.35.36   125.6.2.2     51.16.7.2     5.46.7.48;
2011-08-25 15:35    11.2.3.4      51.6.7.8      97.10.11.12   95.16.17.18   99.20.21.22   45.46.47.48;
```

Figure 3.15 – Data saved in tr-router3.txt

Router 4 - tr-router4.txt is a file saved in the directory named ‘router1’ is the third destination device.

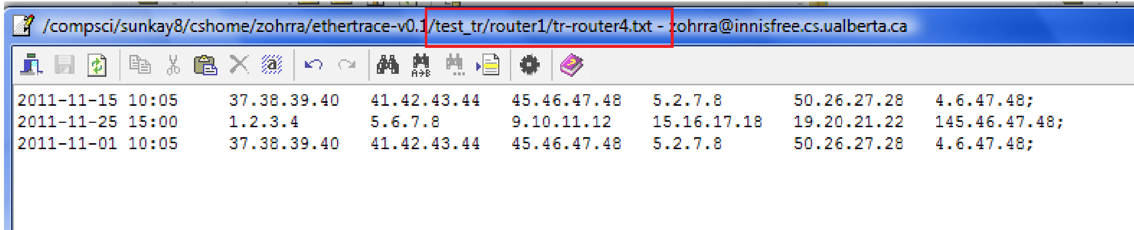



Figure 3.16 – Data saved in tr-router4.txt

Running the script

zohrra@innisfree:~/EtherTrace-v0.1$ python tr.py -d test_tr -k

**Output**

We can observe that when system starts processing this script it reads all the dates and corresponding IP addresses row by row in the first file and prints ‘sorting the trs – done’ and after this statement it prints the data in a sorted order. Then it reads and prints the dates and IP addresses of the other two files and prints the sorted result. The files are picked in the order they are saved in the directory.

Different colours are used to distinguish different files:

```
Processing: test_tr/router1/tr-router2.txt 2011-11-12 08:00    1.2.3.4 5.6.7.8 9.10.11.12
15.16.17.18    19.20.21.22    145.46.47.48;

2011-11-17  06:05          13.14.15.16       17.18.19.20       21.22.23.24       7.8.9.20
21.22.23.24    40.41.47.84;

2011-11-16 17:00       25.26.27.28     29.30.31.32     33.34.35.36     25.6.2.2       5.6.7.2
5.46.7.48;

sorting the trs
done

test_tr/router1/tr-router2.txt:1    router1    router2    ['1.2.3.4',    '5.6.7.8',    '9.10.11.12',
'15.16.17.18', '19.20.21.22']
test_tr/router1/tr-router2.txt:3     router1     router2     ['25.26.27.28',     '29.30.31.32',
'33.34.35.36', '25.6.2.2', '5.6.7.2']

test_tr/router1/tr-router2.txt:2     router1     router2     ['13.14.15.16',     '17.18.19.20',
'21.22.23.24', '7.8.9.20', '21.22.23.24']
```

```
Processing: test_tr/router1/tr-router3.txt 2011-08-12 07:00      111.2.3.4        52.6.7.8
91.10.11.12    151.16.17.18   119.20.21.22   145.46.47.48;

2011-08-16 17:00          215.26.27.28      219.30.31.32      33.34.35.36       125.6.2.2
51.16.7.2      5.46.7.48;

2011-08-25 15:35      11.2.3.4      51.6.7.8      97.10.11.12    95.16.17.18    99.20.21.22
45.46.47.48;

sorting the trs
done

test_tr/router1/tr-router3.txt:1  router1  router3  ['111.2.3.4',  '52.6.7.8',  '91.10.11.12',
'151.16.17.18', '119.20.21.22']
test_tr/router1/tr-router3.txt:2    router1    router3    ['215.26.27.28',    '219.30.31.32',
'33.34.35.36', '125.6.2.2', '51.16.7.2']
test_tr/router1/tr-router3.txt:3   router1   router3   ['11.2.3.4',   '51.6.7.8',   '97.10.11.12',
'95.16.17.18', '99.20.21.22']

Processing:   test_tr/router1/tr-router4.txt   2011-11-15   10:05              37.38.39.40
41.42.43.44    45.46.47.48    5.2.7.8 50.26.27.28    4.6.47.48;

2011-11-25 15:00          1.2.3.4  5.6.7.8  9.10.11.12        15.16.17.18       19.20.21.22
145.46.47.48;

2011-11-01 10:05       37.38.39.40     41.42.43.44     45.46.47.48     5.2.7.8 50.26.27.28
4.6.47.48;

sorting the trs
done

test_tr/router1/tr-router4.txt:3    router1    router4    ['37.38.39.40',    '41.42.43.44',
'45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router4.txt:1    router1    router4    ['37.38.39.40',    '41.42.43.44',
'45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router4.txt:2   router1   router4   ['1.2.3.4',   '5.6.7.8',   '9.10.11.12',
'15.16.17.18', '19.20.21.22']
```

**Database results**

**Case 5 - l3_devs table**

| dev_id | dev_name |
|---|---|
| 0 | router1 |
| 1 | router2 |
| 2 | 1.2.3.4 |
| 3 | 5.6.7.8 |
| 4 | 9.10.11.12 |
| 5 | 15.16.17.18 |
| 6 | 19.20.21.22 |
| 7 | 25.26.27.28 |
| 8 | 29.30.31.32 |
| 9 | 33.34.35.36 |
| 10 | 25.6.2.2 |
| 11 | 5.6.7.2 |
| 12 | 13.14.15.16 |
| 13 | 17.18.19.20 |
| 14 | 21.22.23.24 |
| 15 | 7.8.9.20 |
| 16 | router3 |
| 17 | 111.2.3.4 |
| 18 | 52.6.7.8 |
| 19 | 91.10.11.12 |
| 20 | 151.16.17.18 |
| 21 | 119.20.21.22 |
| 22 | 215.26.27.28 |
| 23 | 219.30.31.32 |
| 24 | 125.6.2.2 |
| 25 | 51.16.7.2 |
| 26 | 11.2.3.4 |
| 27 | 51.6.7.8 |
| 28 | 97.10.11.12 |
| 29 | 95.16.17.18 |
| 30 | 99.20.21.22 |
| 31 | router4 |
| 32 | 37.38.39.40 |
| 33 | 41.42.43.44 |
| 34 | 45.46.47.48 |
| 35 | 5.2.7.8 |
| 36 | 50.26.27.28 |

When tr.py sorts all the files saved in the directory (router1), it displays three rows each containing five IP addresses for file 1, 2 and 3. Therefore all together we have 45 IP addresses. L3_devs table assigns dev_id to distinct IP addresses.
Tr-router2 file has the following 14 distinct IP addresses; please note that the last IP address in the third row appears at the third place in the same row:
test_tr/router1/tr-router2.txt:1 router1 router2 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '15.16.17.18', '19.20.21.22']
test_tr/router1/tr-router2.txt:3 router1 router2 ['25.26.27.28', '29.30.31.32', '33.34.35.36', '25.6.2.2', '5.6.7.2']
test_tr/router1/tr-router2.txt:2 router1 router2 ['13.14.15.16', '17.18.19.20', '21.22.23.24', '7.8.9.20', '21.22.23.24']

In the l3_devs table dev_id 0 and dev _id 1 represent source and destination devices i.e. router1 and router2 respectively, dev_Id 2 to dev_id 15 are assigned to each distinct IP addresses in the order they appear in the sorted list of Tr-router2.

Tr-router3 file has total 15 IP addresses, the following 14 are distinct. If an IP address appeared in the first file (tr-router2) a dev_id was already assigned to it so it considers duplicate in this file and doesn't acquire a dev_id.

test_tr/router1/tr-router3.txt:1 router1 router3 ['111.2.3.4', '52.6.7.8', '91.10.11.12', '151.16.17.18', '119.20.21.22']
test_tr/router1/tr-router3.txt:2 router1 router3 ['215.26.27.28', '219.30.31.32', '33.34.35.36', '125.6.2.2', '51.16.7.2']
test_tr/router1/tr-router3.txt:3 router1 router3 ['11.2.3.4', '51.6.7.8', '97.10.11.12', '95.16.17.18', '99.20.21.22']

In the l3_devs table dev_id 16 is assigned to the destination device i.e. router3, dev_Id 17 to dev_id 30 are assigned to each distinct IP addresses in the order they appear in the sorted list of Tr-router3.

tr-router4 file has total 15 IP addresses, the following 5 are distinct. If an IP address appeared in last two files (tr-router2 and tr-router3) a dev_id was already assigned to it so it considers duplicate in this file and doesn't acquire a dev_id.

test_tr/router1/tr-router4.txt:3 router1 router4 ['37.38.39.40', '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router4.txt:1 router1 router4 ['37.38.39.40', '41.42.43.44', '45.46.47.48', '5.2.7.8', '50.26.27.28']
test_tr/router1/tr-router4.txt:2 router1 router4 ['1.2.3.4', '5.6.7.8', '9.10.11.12', '15.16.17.18', '19.20.21.22']

In the l3_devs table dev_id 31 is assigned to the destination device i.e. router4, dev_Id 32 to dev_id 36 are assigned to each distinct IP addresses in the order they appear in the sorted list of Tr-router3.

**Case 5 - l3_tr table**

| tr_id | hop_num | dev_id |
|---|---|---|
| 0 | 0 | 2 |
| 0 | 1 | 3 |
| 0 | 2 | 4 |
| 0 | 3 | 5 |
| 0 | 4 | 6 |
| 1 | 0 | 7 |
| 1 | 1 | 8 |
| 1 | 2 | 9 |
| 1 | 3 | 10 |
| 1 | 4 | 11 |
| 2 | 0 | 12 |
| 2 | 1 | 13 |
| 2 | 2 | 14 |
| 2 | 3 | 15 |
| 2 | 4 | 14 |
| 3 | 0 | 17 |
| 3 | 1 | 18 |
| 3 | 2 | 19 |
| 3 | 3 | 20 |
| 3 | 4 | 21 |
| 4 | 0 | 22 |
| 4 | 1 | 23 |
| 4 | 2 | 9 |
| 4 | 3 | 24 |
| 4 | 4 | 25 |
| 5 | 0 | 26 |
| 5 | 1 | 27 |
| 5 | 2 | 28 |
| 5 | 3 | 29 |
| 5 | 4 | 30 |
| 6 | 0 | 32 |

| | | |
|---|---|---|
| 6 | 1 | 33 |
| 6 | 2 | 34 |
| 6 | 3 | 35 |
| 6 | 4 | 36 |
| 7 | 0 | 2 |
| 7 | 1 | 3 |
| 7 | 2 | 4 |
| 7 | 3 | 5 |
| 7 | 4 | 6 |

**Case 5 - l3_tr_sd table**

| sd_pair_id | src | dst |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 0 | 16 |
| 2 | 0 | 31 |

This table displays the number of pairs of source and destination. In this example we have one source named router1 and three destination files named tr-router2, tr-router3 and tr-router4. Therefore we've the following three pairs: router1-router2, router1-router3 and router1-router4 therefore source_destination_pair_ids 0-2 are assigned to them respectively. Following are the corresponding dev_ids in the l3_devs table.

| dev_id | dev_name |
|---|---|
| 0 | router1 |
| 1 | Router2 |
| 16 | Router3 |
| 31 | Router4 |

**Case 5 - l3_tr_meta table**

tr-router2 has one path that creates loop and it displays b-loopy=1in the above table.
tr-router4 has one redundant path therefore only the latest record is saved in the table.

| tr_id | ts | sd_pair_id | tr_len | valid_tr_len | b_failed | b_loopy | b_has_star_hop | description | hash_id |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011-11-12 08:00:00 | 0 | 5 | 5 | 0 | 0 | 0 | 1:txt.2retuor-rt/1retuor/rt_tset | 2efa6894 |
| 1 | 2011-11-16 17:00:00 | 0 | 5 | 5 | 0 | 0 | 0 | 3:txt.2retuor-rt/1retuor/rt_tset | 19c5ffa8 |
| 2 | 2011-11-17 06:05:00 | 0 | 5 | 5 | 0 | 1 | 0 | 2:txt.2retuor-rt/1retuor/rt_tset \| | 95596746 |
| 3 | 2011-08-12 07:00:00 | 1 | 5 | 5 | 0 | 0 | 0 | 1:txt.3retuor-rt/1retuor/rt_tset \| | 32008e39 |
| 4 | 2011-08-16 | 1 | 5 | 5 | 0 | 0 | 0 | 2:txt.3retuor- | 4ca9cbd |

| | 17:00: | | | | | | | rt/1retuor/rt_ | e |
|---|---|---|---|---|---|---|---|---|---|
| 5 | 2011-08-25 15:35:00 | 1 | 5 | 5 | 0 | 0 | 0 | 3:txt.3retuor-rt/1retuor/rt_tset | d5d8487 9 |
| 6 | 2011-11-15 10:05:00 | 2 | 5 | 5 | 0 | 0 | 0 | 1:txt.4retuor-rt/1retuor/rt_tset | e1c3898 6 |
| 3 | 2011-11-25 15:00:00 | 2 | 5 | 5 | 0 | 0 | 0 | 2:txt.4retuor-rt/1retuor/rt_tset | 2efa689 4 |

## 3.6 Conclusion

The above results provide useful insight of traceroute files. It has been shown that EtherTrace clearly indicates the source and destination routers. E.g. while using a directory method, the name of the directory indicates the source router and name of the file saved in this directory represents the destination router. The file contains traceroute results from source to destination for various dates. When tr.py script runs it first sorts the traceroute results and then saves useful information in the tables using MYSQL database. These tables provide huge amount of useful information regarding the paths been traversed to reach to the destination e.g. how many distinct IP addresses were traversed, information of the failed paths, paths with loops, number of hops etc. This information not only can be used for determining layer-2 topology, it can also be used for various types of performance analysis and fault diagnosis.

# Chapter 4 – Summary and Future work

As part of this project I first study the paper "Characterizing VLAN-induced sharing in a campus network" [1]" that is written to discuss, measure and characterize VLAN-Induced Dependencies on a Campus Network. One of the contributions that this paper makes in cross-layer analysis is EtherTrace. EtherTrace [2] is a publicly available tool that can be used for layer-2 topology discovery. EtherTrace is implemented in Python and uses MySQL database at the backend. It uses data collected passively from Ethernet switches and routers. EtherTrace assumes that there are two hosts on the network. In order to determine the path between host 1 and host 2 they made various observations. EtherTrace works on different scenarios for determining layer-2 path, one is when hosts are on the same VLAN and other is when hosts are on different VLANs. These two scenarios use ARP tables, bridge tables and IP traceroute results.

After obtaining a general understanding of EtherTrace, I narrowed down my focus to the functionality of tr.py and tr_db.py scripts. These two scripts exclusively deal with the traceroute results. These scripts deal with determining the IP paths of layer-2 using IP-level traceroute between hosts. These are the hosts those that exist on different VLANs and uses routers for inter communication. This project doesn't use actual traceroute results thus can't be used in making an analysis or path elements determination.

However EtherTrace is publicly available but it can't be used effectively unless one has good knowledge of python and MySQL. This project will help anyone with basic or no knowledge of Python and/or MySQL and explain how to take benefit of EtherTrace in general. It specifically will help in network analysis and layer-2 topology discovery where hosts of different VLANs communicate each other.

I believe this initial study will motivate future work to use this tool for network diagnosis/analysis in a real environment and use traceroute results between two hosts on different VLANs in a complex network.

One can be interested in exploring EtherTrace further to determine how it handles other types of data sets that are bridge tables and ARP tables and how it actually finds a path between two hosts. Most importantly, a network administrator might be interested in using EtherTrace fully for examining the relationship between VLANs and IP topologies in the similar fashion as it was used for Georgia campus network. It can be used as a layer-2 network monitoring or planning tool for a wide range of network that has VLAN environment.

## REFERENCES

1. Characterizing VLAN-Induced Sharing in a Campus Network: http://www.gtnoise.net/papers/2009/tariq:vlan:imc09.pdf
2. CPR: Campus Wide Network Performance Monitoring and Recovery http://www.rnoc.gatech.edu/cpr/,2006.
3. EtherTrace. http://www.gtnoise.net/EtherTrace/.
4. http://network-tools.com/