



UNIVERSITY OF
SOUTH CAROLINA

NETWORK MANAGEMENT

Book Version: 07-08-2022

Principal Investigator: Jorge Crichigno



Award 2118311

“Cybertraining on P4 Programmable Devices using an Online Scalable Platform with Physical and Virtual Switches and Real Protocol Stacks”

Contents

Lab 1: Introduction to Mininet

Lab 2: Introduction to NetFlow

Lab 3: Introduction to IPFIX

Lab 4: Introduction to sFlow

Lab 5: Collecting and processing NetFlow, IPFIX, and sFlow data using Nfdump

Lab 6: Filtering and formatting data using Nfdump

Lab 7: Collecting and Visualizing sFlow data using GoFlow and Grafana

Lab 8: Collecting and Visualizing NetFlow data using GoFlow and Grafana



UNIVERSITY OF
SOUTH CAROLINA

NETWORK MANAGEMENT

Lab 1: Introduction to Mininet

Document Version: **07-08-2022**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Introduction to Mininet	3
2 Invoke Mininet using the CLI	5
2.1 Invoke Mininet using the default topology.....	5
2.2 Test connectivity	9
3 Build and emulate a network in Mininet using the GUI	10
3.1 Build the network topology	11
3.2 Test connectivity	13
3.3 Automatic assignment of IP addresses	16
3.4 Save and load a Mininet topology	18
References	19

Overview

This lab provides an introduction to Mininet, a virtual testbed used for testing network tools and protocols. It demonstrates how to invoke Mininet from the command-line interface (CLI) utility and build and emulate topologies using a graphical user interface (GUI) application. In this lab, you will use Containernet, a Mininet network emulator fork that uses Docker containers as hosts in emulated network topologies. However, all the concepts covered are bounded to Mininet.

Objectives

By the end of this lab, you should be able to:

1. Understand what Mininet is and why it is useful for testing network topologies.
2. Invoke Mininet from the CLI.
3. Construct network topologies using the GUI.
4. Save/load Mininet topologies using the GUI.
5. Configure the interfaces of a router using the CLI.

Lab settings

The information in Table 1 provides the credentials of the machine containing Mininet.

Table 1. Credentials to access Client machine.

Device	Account	Password
Client	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction to Mininet.
2. Section 2: Invoke Mininet using the CLI.
3. Section 3: Build and emulate a network in Mininet using the GUI.
4. Section 4: Configure router r1.

1 Introduction to Mininet

Mininet is a virtual testbed enabling the development and testing of network tools and protocols. With a single command, Mininet can create a realistic virtual network on any type of machine (Virtual Machine (VM), cloud-hosted, or native). Therefore, it provides

an inexpensive solution and streamlined development running in line with production networks¹. Mininet offers the following features:

- Fast prototyping for new networking protocols.
- Simplified testing for complex topologies without the need of buying expensive hardware.
- Realistic execution as it runs real code on the Unix and Linux kernels.
- Open-source environment backed by a large community contributing extensive documentation.

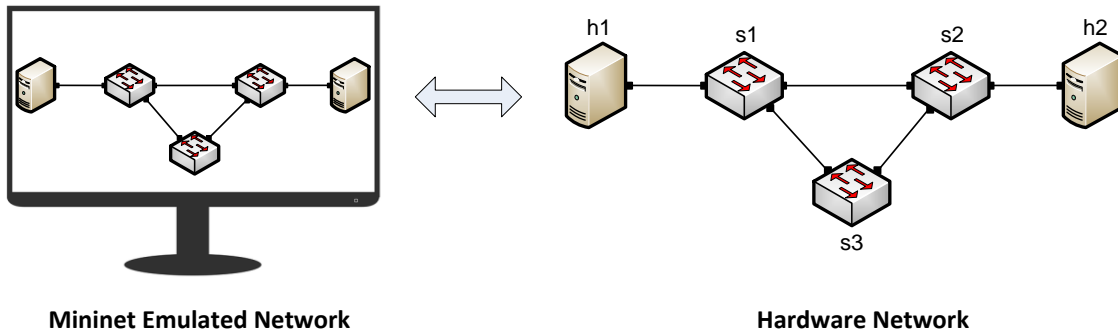


Figure 1. Hardware network vs. Mininet emulated network.

Mininet is useful for development, teaching, and research as it is easy to customize and interact with it through the CLI or the GUI. Mininet was originally designed to experiment with *OpenFlow*² and *Open Virtual Network (Open vSwitch)*³. This lab, however, only focuses on emulating a simple network environment without Open vSwitch devices.

Mininet’s logical nodes can be connected into networks. These nodes are sometimes called containers, or more accurately, *network namespaces*. Containers consume sufficiently fewer resources that networks of over a thousand nodes have created, running on a single laptop. A Mininet container is a process (or group of processes) that no longer has access to all the host system’s native network interfaces. Containers are then assigned virtual Ethernet interfaces, which are connected to other containers through a virtual switch⁴. Mininet connects a host and a switch using a virtual Ethernet (veth) link. The veth link is analogous to a wire connecting two virtual interfaces, as illustrated below.

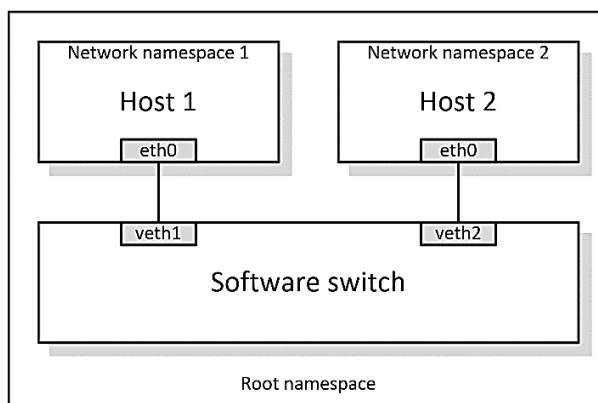


Figure 2. Network namespaces and virtual Ethernet links.

Each container is an independent network namespace, a lightweight virtualization feature that provides individual processes with separate network interfaces, routing tables, and Address Resolution Protocol (ARP) tables.

Mininet provides network emulation opposed to simulation, allowing all network software at any layer to be simply run *as is*; i.e., nodes run the native network software of the physical machine. On the other hand, in a simulated environment applications and protocol implementations need to be ported to run within the simulator before they can be used.

2 Invoke Mininet using the CLI

The first step to start Mininet using the CLI is to start a Linux terminal.

2.1 Invoke Mininet using the default topology

Step 1. Click on the Client tab to access the Client PC.

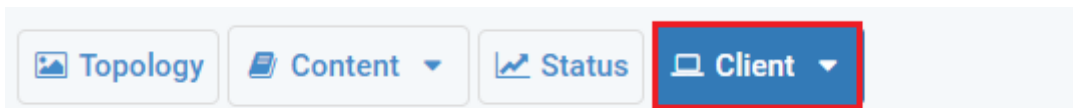


Figure 3. Accessing the Client PC.

Step 2. Launch a Linux terminal by clicking on the icon located on the taskbar.



Figure 4. Shortcut to open a Linux terminal.

The Linux terminal is a program that opens a window and permits you to interact with a command-line interface (CLI). A CLI is a program that takes commands from the keyboard and sends them to the operating system for execution.

Step 3. To start a minimal topology, enter the command shown below. When prompted for a password, type `password` and hit enter. Note that the password will not be visible as you type it.

```
sudo mn
```

```

Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
Lubuntu@admin:~$ sudo mn
[sudo] password for Lubuntu:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
containernet>

```

Figure 5. Starting Mininet using the CLI.

The above command starts Mininet with a minimal topology, which consists of a switch connected to two hosts as shown below.

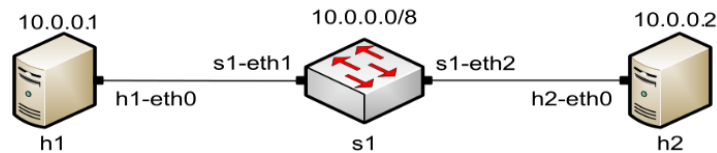


Figure 6. Mininet's default minimal topology.

When issuing the `sudo mn` command, Mininet initializes the topology and launches its command line interface which looks like this:

```
containernet>
```

Step 4. To display the list of Mininet CLI commands and examples on their usage, type the following command:

```
help
```

```

Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
containernet> help

Documented commands (type help <topic>):
=====
EOF      gterm  iperfudp  nodes    pingpair  py       switch
dpctl    help   link      noecho   pingpairfull  quit    time
dump     intfs  links     pingall  ports     sh       x
exit     iperf  net       pingallfull  px        source  xterm

You may also send a command to a node using:
  <node> command {args}
For example:
  mininet> h1 ifconfig

The interpreter automatically substitutes IP addresses
for node names when a node is the first arg, so commands
like
  mininet> h2 ping h3
should work.

Some character-oriented interactive commands require
noecho:
  mininet> noecho h2 vi foo.py
However, starting up an xterm/gterm is generally better:
  mininet> xterm h2

containernet>

```

Figure 7. Mininet's `help` command.

Step 5. To display the available nodes, type the following command:

```
nodes
```

```

Lubuntu@
File Actions Edit View Help
Lubuntu@admin: ~
containernet> nodes
available nodes are:
c0 h1 h2 s1
containernet>

```

Figure 8. Mininet's `nodes` command.

The output of this command shows that there is a controller, two hosts (host h1 and host h2), and a switch (s1).

Step 6. It is useful sometimes to display the links between the devices in Mininet to understand the topology. Issue the command shown below to see the available links.

```
net
```

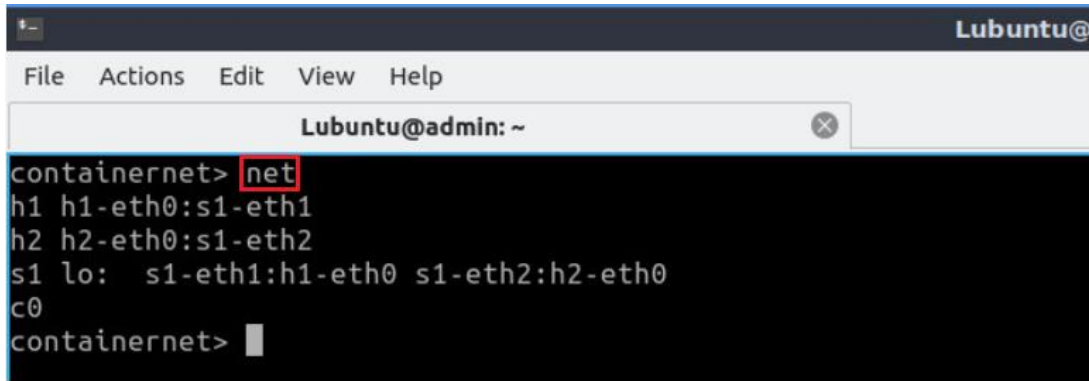
A screenshot of a terminal window titled 'Lubuntu@admin: ~'. The terminal shows the command 'net' being entered at the 'containernet>' prompt. The output lists the network topology: 'h1 h1-eth0:s1-eth1', 'h2 h2-eth0:s1-eth2', 's1 lo: s1-eth1:h1-eth0 s1-eth2:h2-eth0', and 'c0'. The prompt returns to 'containernet>'.

Figure 9. Mininet's `net` command.

The output of this command shows that:

1. Host *h1* is connected using its network interface *h1-eth0* to the switch on interface *s1-eth1*.
2. Host *h2* is connected using its network interface *h2-eth0* to the switch on interface *s1-eth2*.
3. Switch *s1*:
 - a. has a loopback interface *lo*.
 - b. connects to *h1-eth0* through interface *s1-eth1*.
 - c. connects to *h2-eth0* through interface *s1-eth2*.
4. Controller *c0* is the brain of the network, where it has a global knowledge about the network. A controller instructs the switches on how to forward/drop packets in the network.

Mininet allows you to execute commands on a specific device. To issue a command for a specific node, you must specify the device first, followed by the command.

Step 7. To proceed, issue the command:

```
h1 ifconfig
```

```

Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
containernet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
    ether fe:cc:3a:51:af:27 txqueuelen 1000 (Ethernet)
    RX packets 23 bytes 3089 (3.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 270 (270.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

containernet>

```

Figure 10. Output of `h1 ifconfig` command.

This command executes the `ifconfig` Linux command on host h1. The command shows host h1's interfaces. The display indicates that host h1 has an interface `h1-eth0` configured with IP address 10.0.0.1, and another interface `lo` configured with IP address 127.0.0.1 (loopback interface).

2.2 Test connectivity

Mininet's default topology assigns the IP addresses 10.0.0.1/8 and 10.0.0.2/8 to host h1 and host h2, respectively. To test connectivity between them, you can use the command `ping`. The `ping` command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to the remote computer and waiting for a response. Information available includes how many responses are returned and how long it takes for them to return.

Step 1. On the CLI, type the command shown below. This command tests the connectivity between host h1 and host h2. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test.

```
h1 ping 10.0.0.2
```

```

Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
containernetwork> h1 ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=14.7 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.281 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.049 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 5ms
rtt min/avg/max/mdev = 0.049/5.005/14.687/6.846 ms
containernetwork>
    
```

Figure 11. Connectivity test between host h1 and host h2.

Step 2. Stop the emulation by typing the following command:

```
exit
```

```

Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
containernetwork> exit
*** Stopping 1 controllers
c0
*** Stopping 2 links
..
*** Stopping 1 switches
s1
*** Stopping 2 hosts
h1 h2
*** Done
completed in 520.383 seconds
Lubuntu@admin:~$
    
```

Figure 12. Stopping the emulation using `exit`.

The command `sudo mn -c` is often used on the Linux terminal (not on the Mininet CLI) to clean a previous instance of Mininet (e.g., after a crash).

3 Build and emulate a network in Mininet using the GUI

In this section, you will use the application MiniEdit⁵ to deploy the topology illustrated below. MiniEdit is a simple GUI network editor for Mininet.

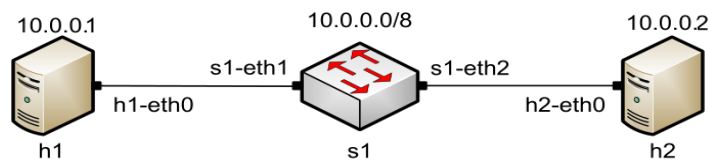


Figure 13. Lab topology.

3.1 Build the network topology

Step 1. A shortcut to MiniEdit is located on the machine's desktop. Start MiniEdit by clicking on MiniEdit's shortcut. When prompted for a password, type `password`.

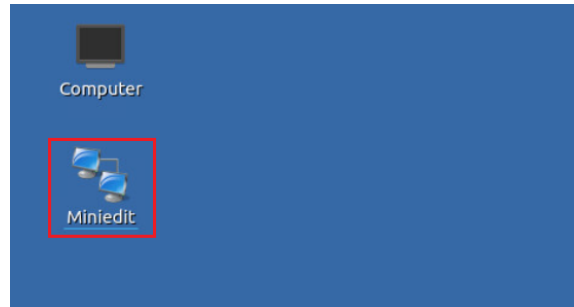


Figure 14. MiniEdit desktop shortcut.

MiniEdit will start, as illustrated below.

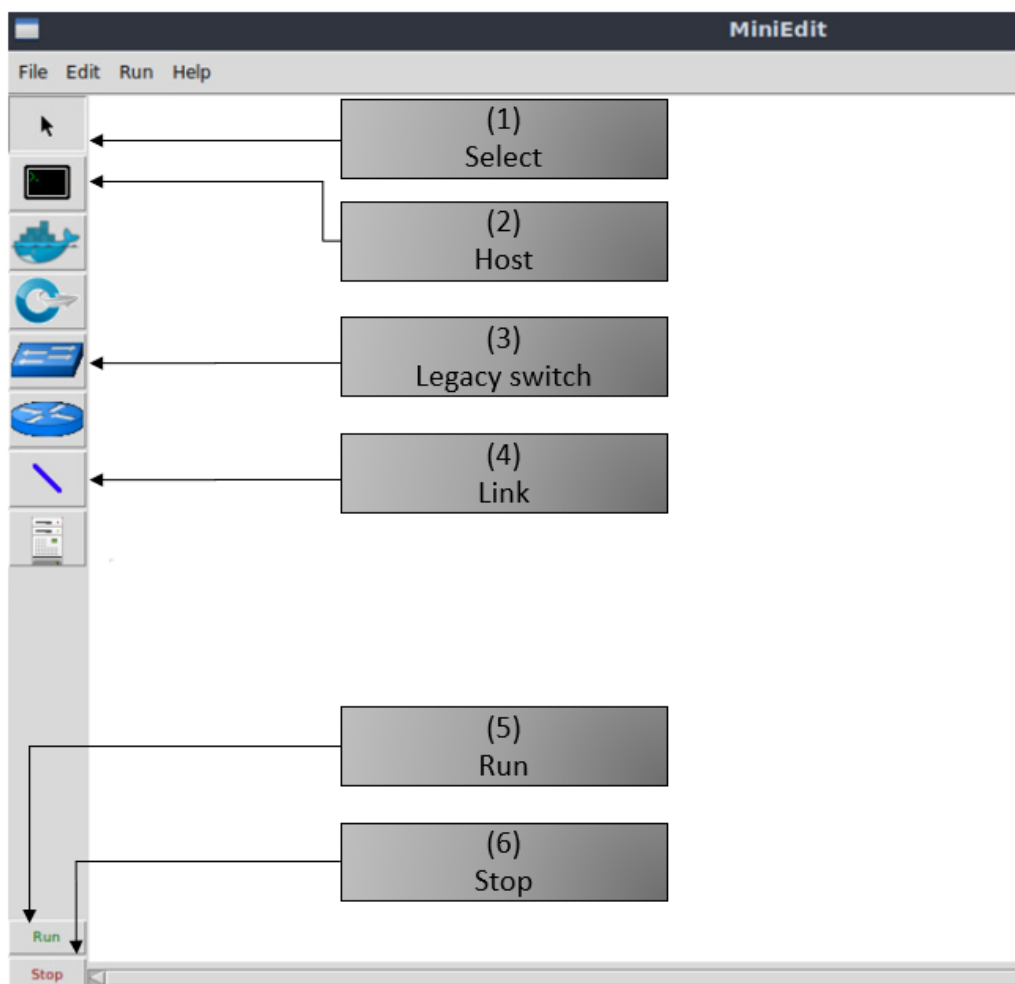


Figure 15. MiniEdit Graphical User Interface (GUI).

The main buttons in this lab are:

1. *Select*: allows selection/movement of the devices. Pressing *Del* on the keyboard after selecting the device removes it from the topology.
2. *Host*: allows addition of a new host to the topology. After clicking this button, click anywhere in the blank canvas to insert a new host.
3. *Legacy switch*: allows addition of a new legacy switch to the topology. After clicking this button, click anywhere in the blank canvas to insert the switch.
4. *Link*: connects devices in the topology (mainly switches and hosts). After clicking this button, click on a device and drag to the second device to which the link is to be established.
5. *Run*: starts the emulation. After designing and configuring the topology, click the run button.
6. *Stop*: stops the emulation.

Step 2. To build the topology illustrated in Figure 13, two hosts and one switch must be deployed. Deploy these devices in MiniEdit, as shown below.

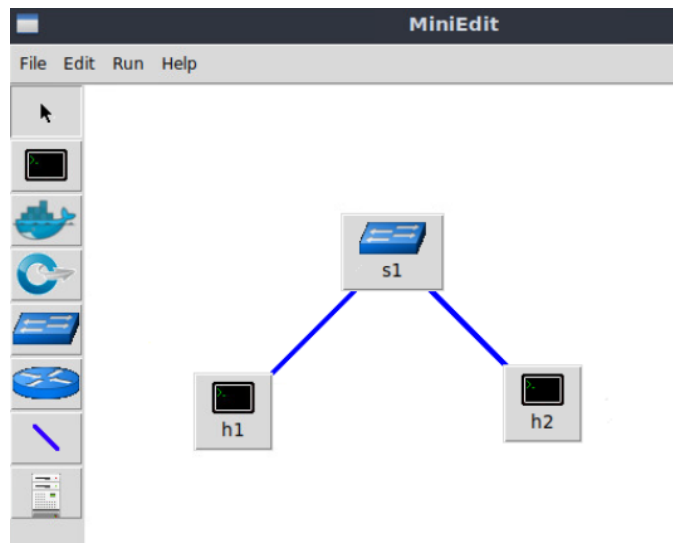


Figure 16. MiniEdit's topology.

Use the buttons described in the previous step to add and connect devices. The configuration of IP addresses is described in Step 3.

Step 3. Configure the IP addresses of host h1 and host h2. Host h1's IP address is 10.0.0.1/8 and host h2's IP address is 10.0.0.2/8. A host can be configured by holding the right click and selecting properties on the device. For example, host h2 is assigned the IP address 10.0.0.2/8 in the figure below.

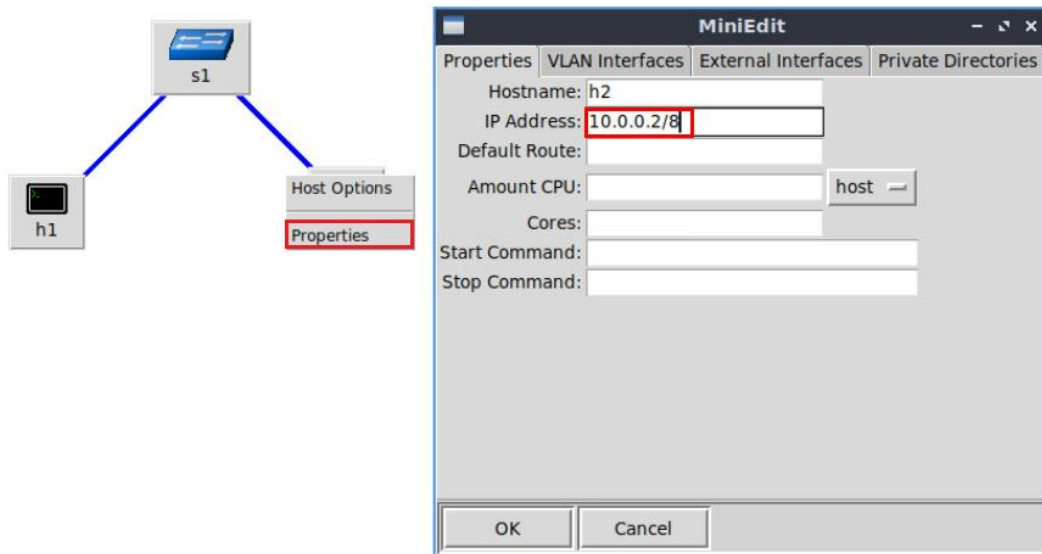


Figure 17. Configuration of a host's properties.

3.2 Test connectivity

Before testing the connection between host h1 and host h2, the emulation must be started.

Step 1. Click on the *Run* button to start the emulation. The emulation will start and the buttons of the MiniEdit panel will gray out, indicating that they are currently disabled.



Figure 18. Starting the emulation.

Step 2. Open a terminal on host h1 by holding the right click on host h1 and selecting *Terminal*. This opens a terminal on host h1 and allows the execution of commands on the host h1. Repeat the procedure on host h2.

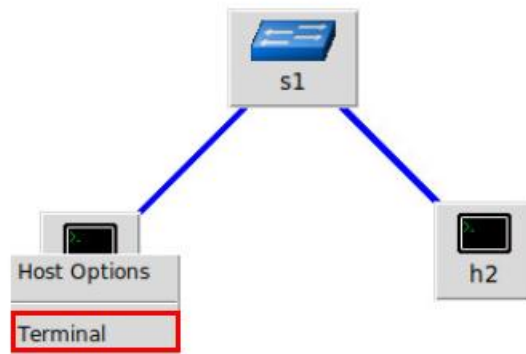


Figure 19. Opening a terminal on host h1.

The network and terminals at host h1 and host h2 will be available for testing.

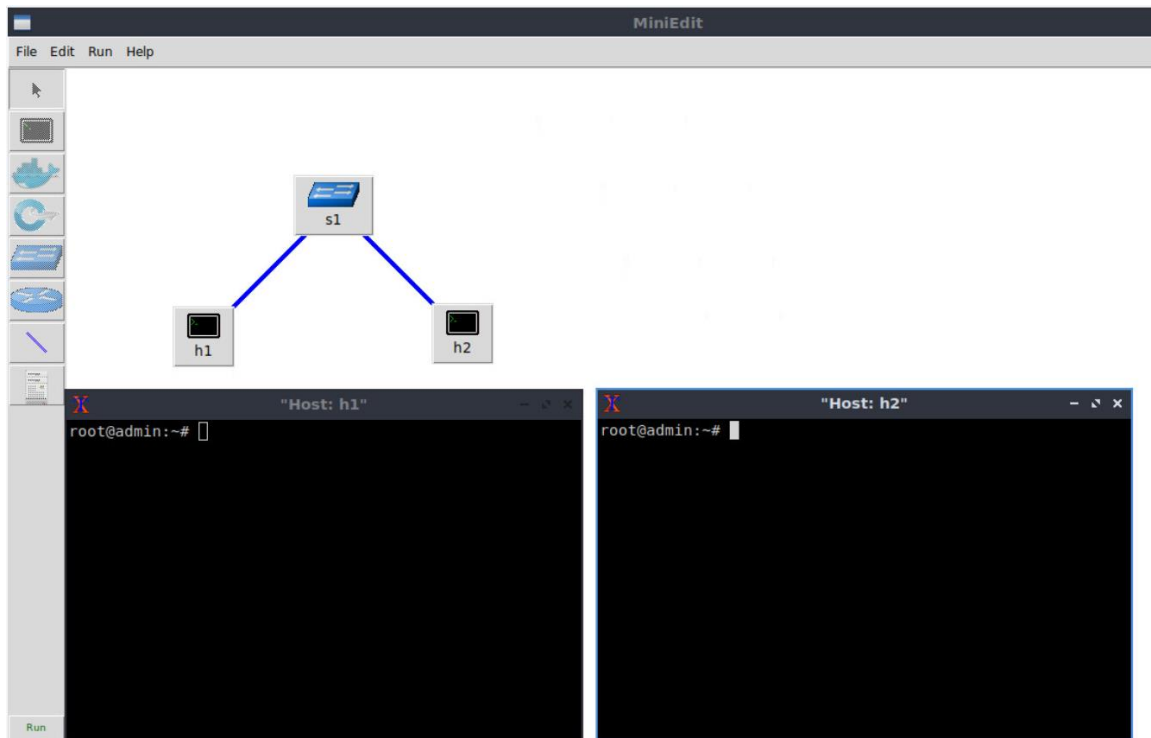


Figure 20. Terminals at host h1 and host h2.

Step 3. On host h1's terminal, type the command shown below to display its assigned IP addresses. The interface *h1-eth0* at host h1 should be configured with the IP address 10.0.0.1 and subnet mask 255.0.0.0.

```
ifconfig
```

```

Host: h1
root@admin:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
    ether 8a:06:a5:6b:1e:9b txqueuelen 1000 (Ethernet)
    RX packets 24 bytes 3179 (3.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3 bytes 270 (270.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@admin:~#

```

Figure 21. Output of `ifconfig` command on host h1.

Repeat Step 3 on host h2. Its interface `h2-eth0` should be configured with IP address 10.0.0.2 and subnet mask 255.0.0.0.

Step 4. On host h1's terminal, type the command shown below. This command tests the connectivity between host h1 and host h2. To stop the test, press `Ctrl+c`. The figure below shows a successful connectivity test.

```
ping 10.0.0.2
```

```

Host: h1
root@admin:~# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.540 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.049 ms
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 41ms
rtt min/avg/max/mdev = 0.047/0.212/0.540/0.231 ms
root@admin:~#

```

Figure 22. Connectivity test using `ping` command.

Step 5. Stop the emulation by clicking on the *Stop* button.



Figure 23. Stopping the emulation.

3.3 Automatic assignment of IP addresses

In the previous section, you manually assigned IP addresses to host h1 and host h2. An alternative is to rely on Mininet for an automatic assignment of IP addresses (by default, Mininet uses automatic assignment), which is described in this section.

Step 1. Remove the manually assigned IP address from host h1. Hold right-click on host h1, *Properties*. Delete the IP address, leaving it unassigned, and press the *OK* button as shown below. Repeat the procedure on host h2.

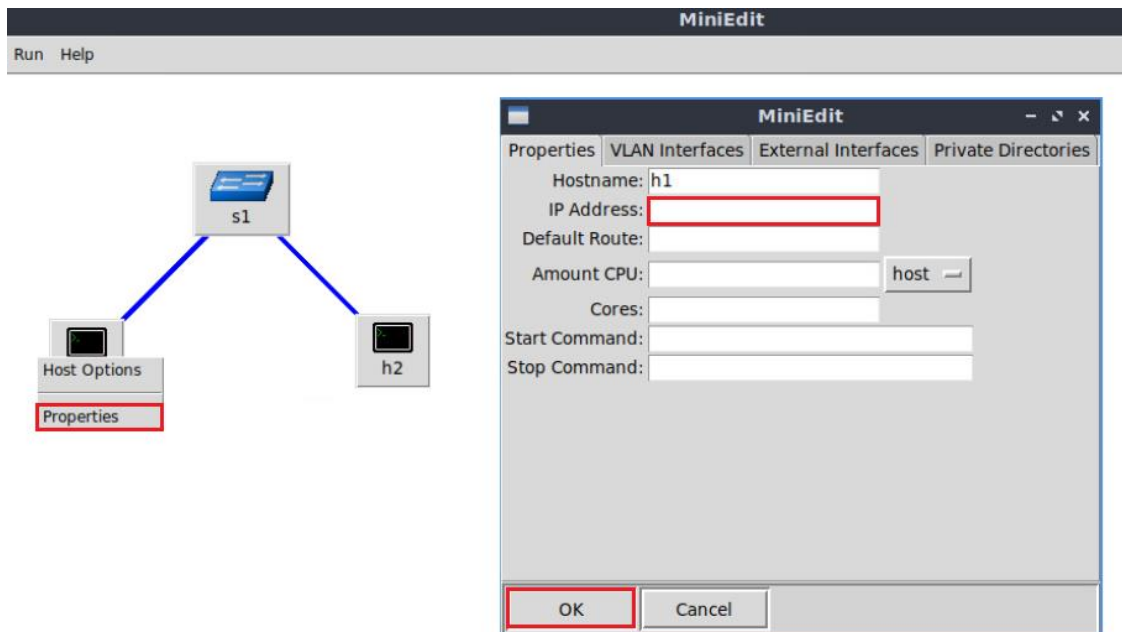


Figure 24. Host h1 properties.

Step 2. Click on *Edit, Preferences* button. The default IP base is 10.0.0.0/8. Modify this value to 15.0.0.0/8, and then press the *OK* button.

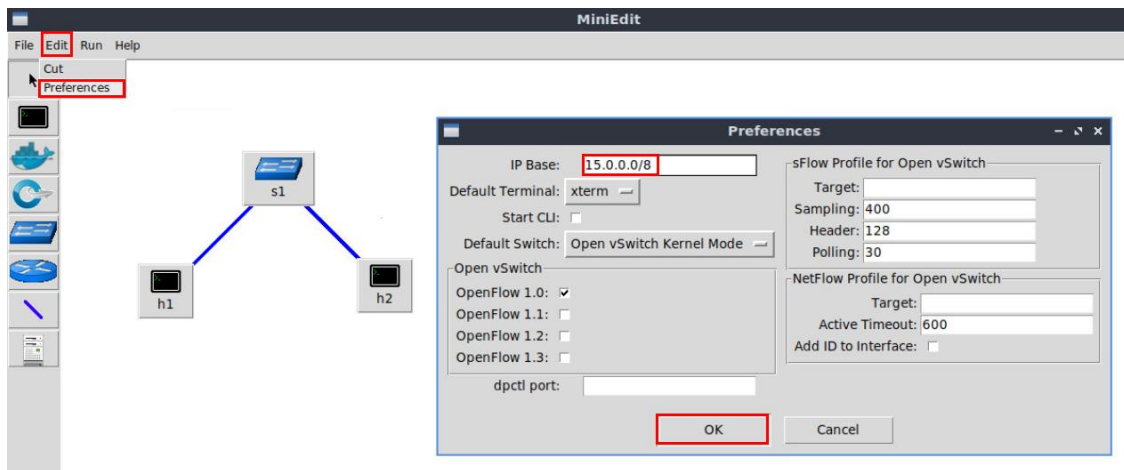


Figure 25. Modification of the IP Base (network address and prefix length).

Step 3. Run the emulation again by clicking on the *Run* button. The emulation will start and the buttons of the MiniEdit panel will be disabled.

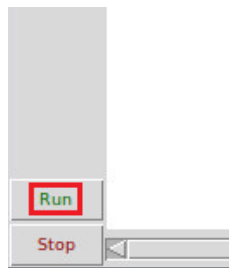


Figure 26. Starting the emulation.

Step 4. Open a terminal on host h1 by holding the right click on host h1 and selecting Terminal.

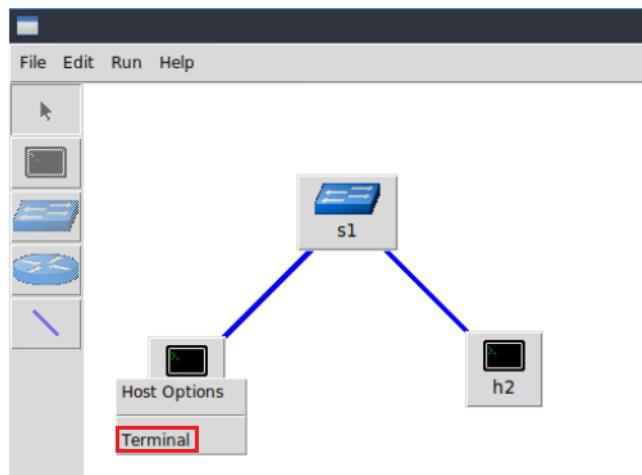


Figure 27. Opening a terminal on host h1.

Step 5. Type the command shown below to display the IP addresses assigned to host h1. The interface *h1-eth0* at host h1 now has the IP address 15.0.0.1 and subnet mask 255.0.0.0.

```
ifconfig
```

```

X "Host: h1"
root@admin:~# ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 15.0.0.1 netmask 255.0.0.0 broadcast 0.0.0.0
ether ae:25:f6:62:66:d2 txqueuelen 1000 (Ethernet)
RX packets 18 bytes 2554 (2.5 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 3 bytes 270 (270.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@admin:~# █

```

Figure 28. Output of `ifconfig` command on host h1.

You can also verify the IP address assigned to host h2 by repeating Steps 4 and 5 on host h2's terminal. The corresponding interface `h2-eth0` at host h2 has now the IP address 15.0.0.2 and subnet mask 255.0.0.0.

Step 6. Stop the emulation by clicking on *Stop* button.



Figure 29. Stopping the emulation.

3.4 Save and load a Mininet topology

In this section you will save and load a Mininet topology. It is often useful to save the network topology, particularly when its complexity increases. MiniEdit enables you to save the topology to a file.

Step 1. Save the current topology by clicking on *File* then *Save*. A new window will emerge. Provide a name for the topology and save it in the local folder. In this case, we used *myTopology* as the topology name.

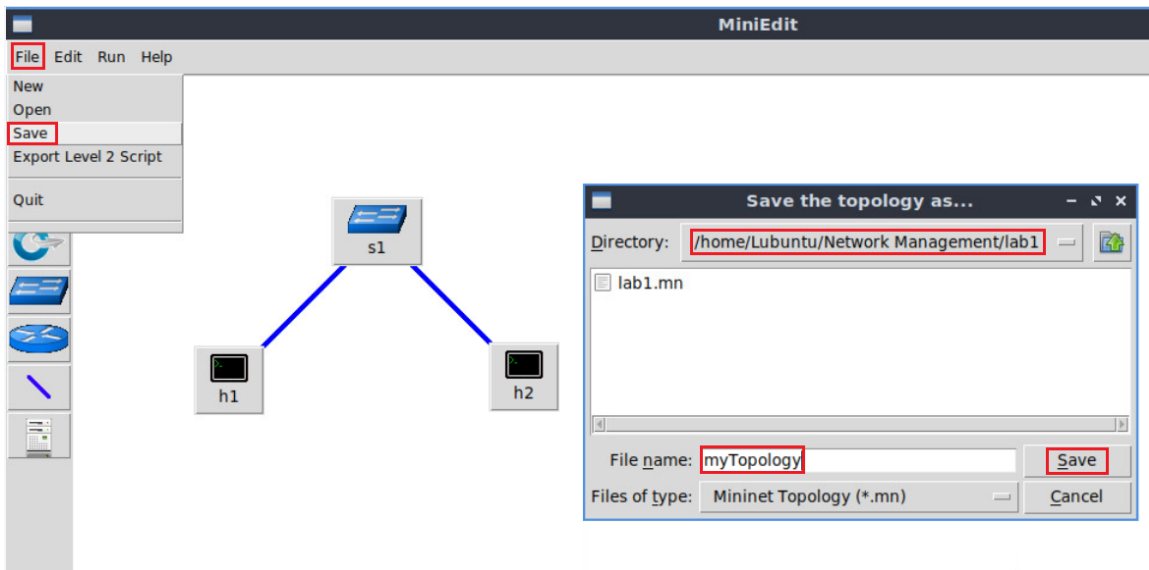


Figure 30. Saving the topology.

Step 2. Load the topology by clicking on *File* then *Open*. Open the directory *lab 1* and search for the topology file called *lab1.mn*. Then, click on *Open*. A new topology will be loaded to MiniEdit.

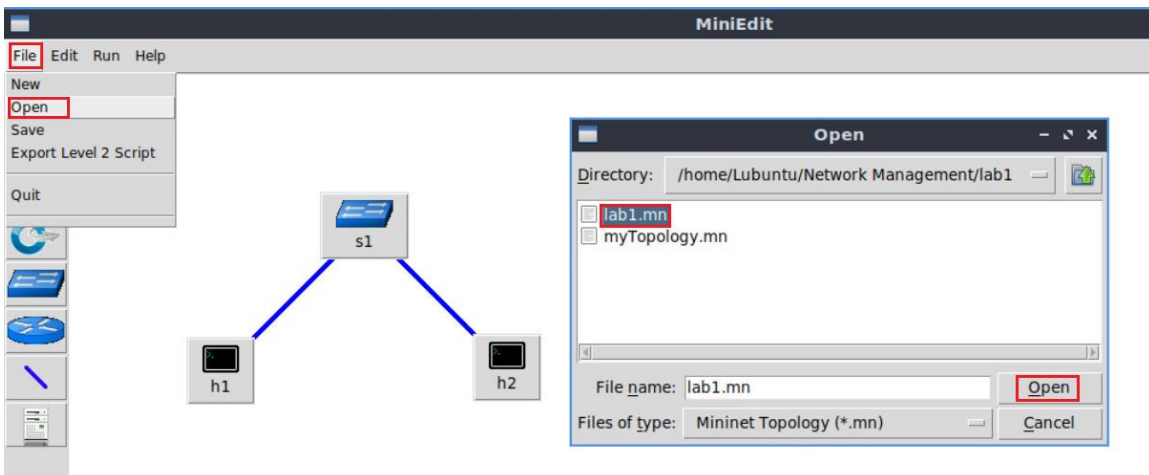


Figure 31. Opening a topology.

This concludes Lab 1. Stop the emulation and then exit out of MiniEdit and Linux terminal.

References

1. Mininet walkthrough. [Online]. Available: <http://Mininet.org>.
2. N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow," ACM SIGCOMM computer Communication review, vol. 38, no. 2, p. 69, 2008.
3. Linux foundation, [Online]. Available: <http://openvSwitch.org>.
4. P. Dordal, "An iintroduction to computer networks,". [Online]. Available: <https://intronetworks.cs.luc.edu/>.

5. B. Lantz, G. Gee, "*MiniEdit: a simple network editor for Mininet*," 2013. [Online]. Available: <https://github.com/Mininet/Mininet/blob/master/examples>.



UNIVERSITY OF
SOUTH CAROLINA

NETWORK MANAGEMENT

Lab 2: Introduction to NetFlow

Document Version: **07-08-2022**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Introduction	3
1.1 Introduction to NetFlow.....	4
1.2 NetFlow in Open vSwitch	4
1.3 NetFlow format output fields.....	5
2 Lab topology.....	6
2.1 Lab settings.....	7
2.2 Loading a topology	7
3 Launching NetFlow exporter.....	10
4 Analyzing NetFlow records using Wireshark	12
4.1 Launching Wireshark.....	12
4.2 Performing a connectivity test	14
4.3 Visualizing NetFlow packets.....	15
References	19

Overview

This lab introduces NetFlow which is a network protocol developed by Cisco for collecting IP traffic information and monitoring network flow. NetFlow enabled switches or routers are called NetFlow exporters, which examine each packet and create flows from these packets. These collected flows are exported to an external device known as NetFlow collector which then organizes the flow records into a format that allows the administrator to further analyze the traffic². The focus of this lab is to explore how NetFlow exporter and collector work in Open Virtual Switch (Open vSwitch) and analyze the collected flows using Wireshark packet analyzer.

Objectives

By the end of this lab, you should be able to:

1. Understand the concept of NetFlow.
2. Understand how NetFlow works in Open vSwitch.
3. Enable NetFlow in Open vSwitch.
4. Analyze NetFlow records using Wireshark.

Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Launching NetFlow exporter.
4. Section 4: Analyzing NetFlow records using Wireshark.

1 Introduction

Monitoring IP traffic flows facilitates more accurate capacity planning. It enables resource alignment which ensures that resources are used appropriately in support of

organizational goals. When the network behavior is understood, it improves the business process, reduces vulnerability of the network, and allows efficient operation of the network¹. Among many of the network traffic monitoring tools, Cisco's NetFlow is widely used in organizations since it provides some additional features like attack detection, lower cost of network security monitoring and enhance network visibility⁵.

1.1 Introduction to NetFlow

NetFlow provides a detailed view of application flows on the network. Initially, NetFlow was created for billing and accounting of network traffic and to measure other IP traffic characteristics such as bandwidth utilization and application performance. Nowadays, NetFlow is also used for security analysis. NetFlow allows the administrator to see what is happening across the network, identify DDoS attacks and monitor network usage⁶.

NetFlow is one-way traffic technology. When a request from a client to the server is sent, NetFlow exporter looks into the packet header and creates a flow record. The flow record contains information about the source address, destination addresses, ports, and all other information. When the server responds to the client, another flow record is created. The subsequent packets with the same attributes update the previously created flow records (e.g.: number of bytes, duration of communication). When the communication is over, flow records are sent to NetFlow collector⁷.

Among all the NetFlow versions, version 5 and version 9 are widely used. NetFlow v5 is the most popular version and is still supported by many router brands. It offers a fixed packet format, making NetFlow traffic monitoring and reporting easier since the contents of each packet are quickly identifiable⁸. NetFlow v9 is template based where users can freely choose which fields to have in the exported NetFlow packets. It is mostly used to report flows such as IPv6, Multiprotocol Label Switching (MPLS) and Border Gateway Protocol (BGP)⁶.

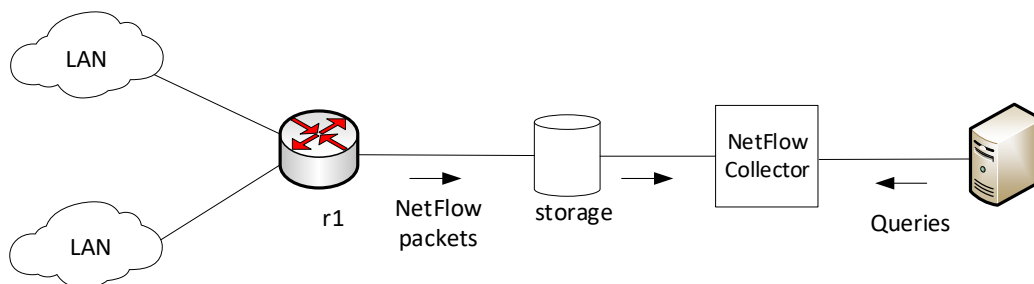


Figure 1. NetFlow architecture.

Consider Figure 1. Router r1 is acting as NetFlow exporter which creates network traffic records. Flows are stored in a local database called NetFlow cache. Once all the flows are collected, the records are transmitted to the NetFlow collector.

1.2 NetFlow in Open vSwitch

Open vSwitch NetFlow support is all about visibility into the virtual switch infrastructure. It allows users to monitor both incoming and outgoing traffic of the Open vSwitch. It significantly improves the ability to secure virtual environments, provides an analysis, diagnosis, and problem-solving platform for the virtual network activities. A basic list of information elements that are exported in NetFlow includes source IP address, destination IP address, source port, destination port, protocol, packets, bytes, class of service³.

Open vSwitch only supports NetFlow version 5. The ovs-vsctl command-line tool is used to configure NetFlow. There are two steps involved in attaching a NetFlow monitor to a switch: defining the monitor and linking a switch to it⁴.

1.3 NetFlow format output fields

This section focuses on detailed explanation of packet formats and fields. Consider Figure 2. The Packet header is the first part of an export packet and provides basic information about the packet. Information included in the packet header is the NetFlow version, number of flow records, flow sequence, timestamp, type of flow switching engine (EngineType), slot number of flow switching engine (EngineId)⁶.

Version	Count
SysUptime	
Timestamp	
Flow sequence	
EngineType	EngineId

Figure 2. NetFlow packet header format.

Consider Figure 3. The figure shows the format of a flow record which includes source and destination IP address, next-hop IP address, in port and out port, total packets, total bytes and other information⁶.

Source IP address			
Destination IP address			
Next-hop IP address			
Input ifIndex		Output ifIndex	
Packets			
Bytes			
Source port		Destination port	
Padding	TCP flags	IP protocol	TOS
Source AS		Destination AS	
Source mask length	Dest. mask length	Padding	

Figure 3. NetFlow flow record format.

2 Lab topology

Consider Figure 4. There are three switches, two end hosts and a docker container. Switch s3 is acting as NetFlow exporter and the docker will collect the flows.

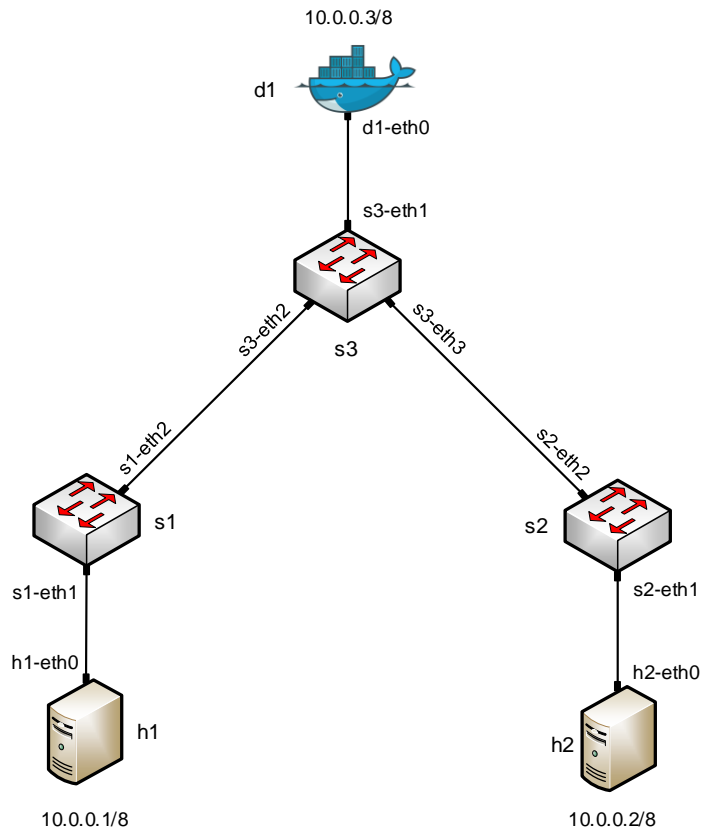


Figure 4. Lab topology.

2.1 Lab settings

The devices should be configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
d1	d1-eth0	10.0.0.3	/8

2.2 Loading a topology

Step 1. Click on the Client tab to access the Client PC.

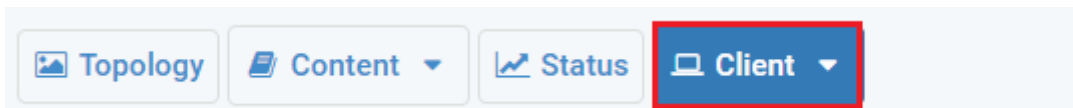


Figure 5. Accessing the Client PC.

Step 2. Start by launching MiniEdit by clicking on desktop's shortcut. When prompted for a password, type `password`.

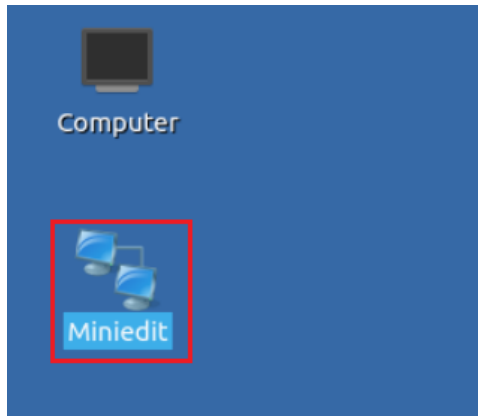


Figure 6. MiniEdit shortcut.

Step 3. On MiniEdit's menu bar, click on *File* then *open* to load the lab's topology. Open the directory called *lab2* and select the file *lab2.mn*. Then, click on *Open* to open the topology.

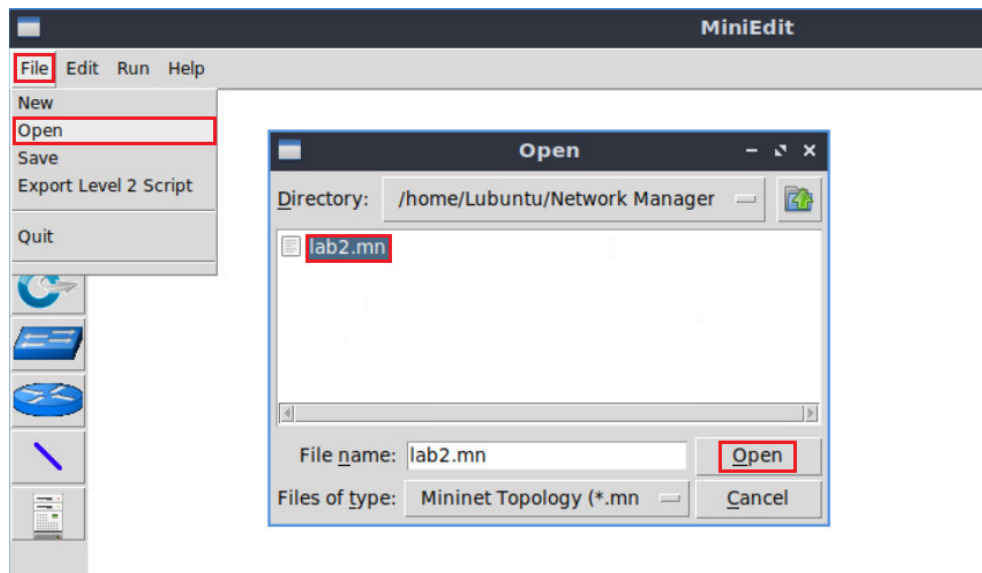


Figure 7. MiniEdit's Open dialog.

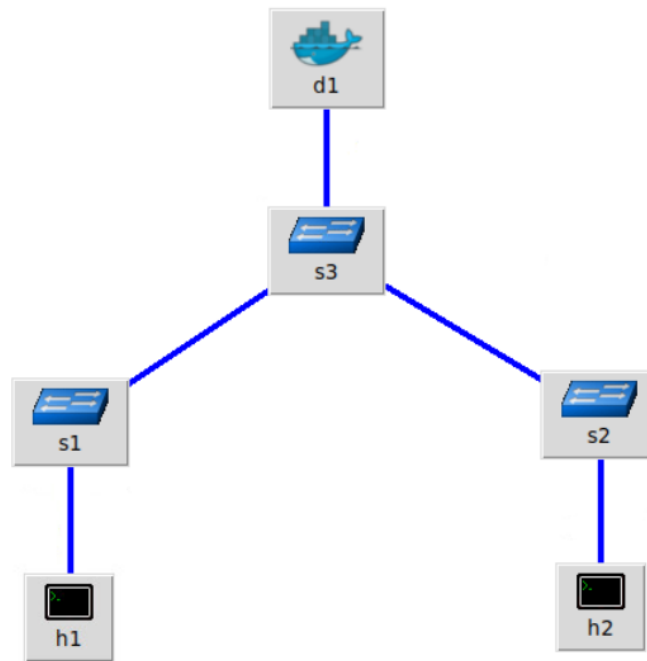


Figure 8. MiniEdit's topology.

Step 4. To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 9. Starting the emulation.

Step 5. Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

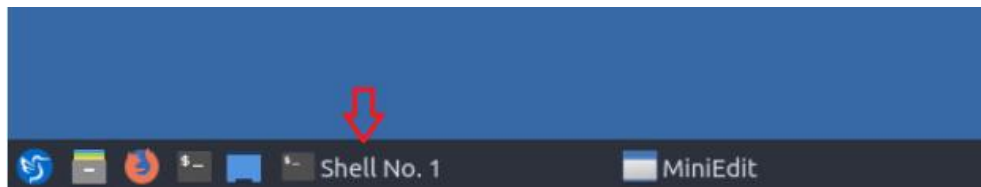


Figure 10. Opening Mininet's terminal.

Step 6. Issue the following command to display the interface names and connections.

```
links
```

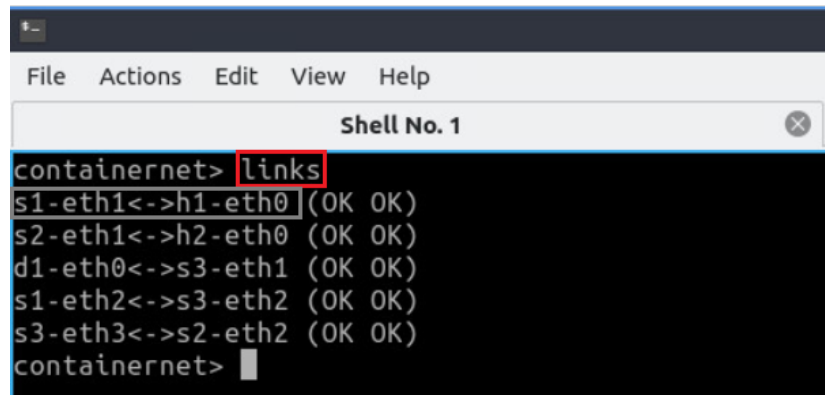


Figure 11. Displaying network interfaces.

In figure 11, the link displayed within the gray box indicates that interface eth1 of switch s1 connects to interface eth0 of host h1 (i.e., *s1-eth1<->h1-eth0*).

3 Launching NetFlow exporter

Step 1. Open the Linux terminal.



Figure 12. Opening Linux terminal.

Step 2. Execute the following script in order to start NetFlow exporter. When prompted for a password, type `password`.

```
sudo ./netflow.sh
```

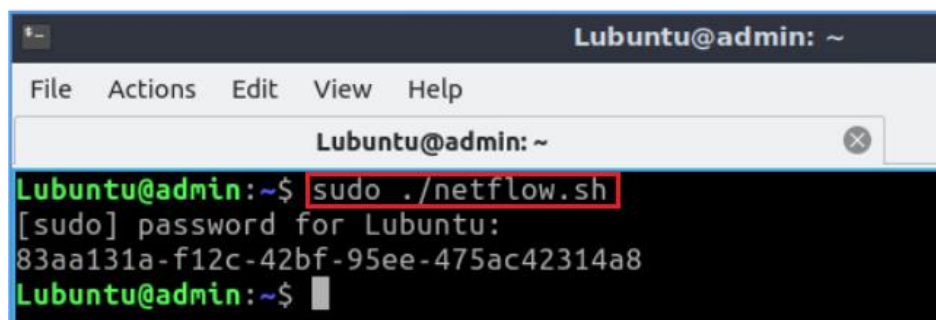


Figure 13. Starting NetFlow exporter.

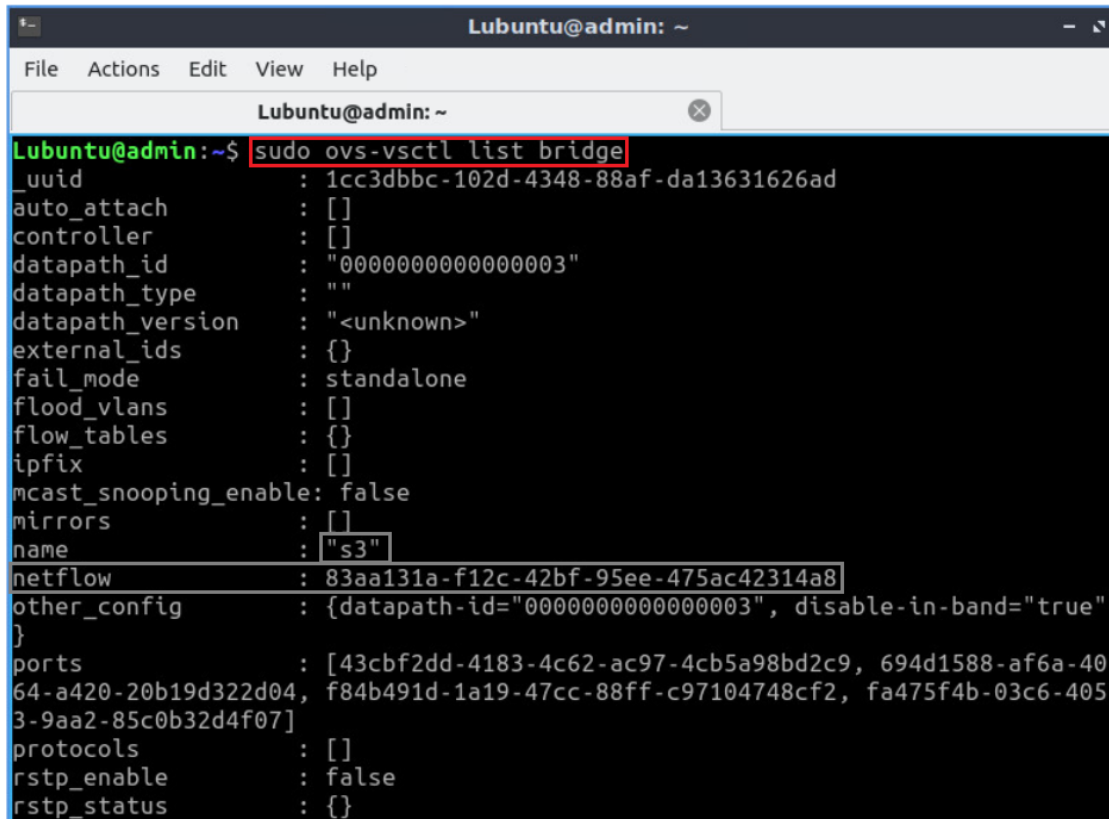
The following commands were executed in the script.

```
ovs-vsctl -- set Bridge s3 netflow=@nf -- --id=@nf create netflow targets=\"10.0.0.3:9995\"
```

The command creates a NetFlow ID and attaches it to switch s3. Switch s3 is acting as an exporter and transmits data to the collector. Docker d1 is the collector IP and the port is the default UDP port 9995.

Step 3. Type the following command to verify the NetFlow configuration.

```
sudo ovs-vsctl list bridge
```

A terminal window titled 'Lubuntu@admin: ~' showing the command 'sudo ovs-vsctl list bridge' and its output. The output lists various configuration parameters for a bridge named 's3'. The 'netflow' parameter is highlighted with a red box, showing the ID '83aa131a-f12c-42bf-95ee-475ac42314a8'.

```
Lubuntu@admin:~$ sudo ovs-vsctl list bridge
_name            : s3
_uuid           : 1cc3dbbc-102d-4348-88af-da13631626ad
auto_attach     : []
controller      : []
datapath_id     : "000000000000000003"
datapath_type   : ""
datapath_version : "<unknown>"
external_ids    : {}
fail_mode       : standalone
flood_vlans     : []
flow_tables     : {}
ipfix           : []
mcast_snooping_enable : false
mirrors         : []
name            : "s3"
netflow         : 83aa131a-f12c-42bf-95ee-475ac42314a8
other_config    : {datapath-id="000000000000000003", disable-in-band="true"}
ports           : [43cbf2dd-4183-4c62-ac97-4cb5a98bd2c9, 694d1588-af6a-4064-a420-20b19d322d04, f84b491d-1a19-47cc-88ff-c97104748cf2, fa475f4b-03c6-4053-9aa2-85c0b32d4f07]
protocols       : []
rstp_enable     : false
rstp_status     : {}
```

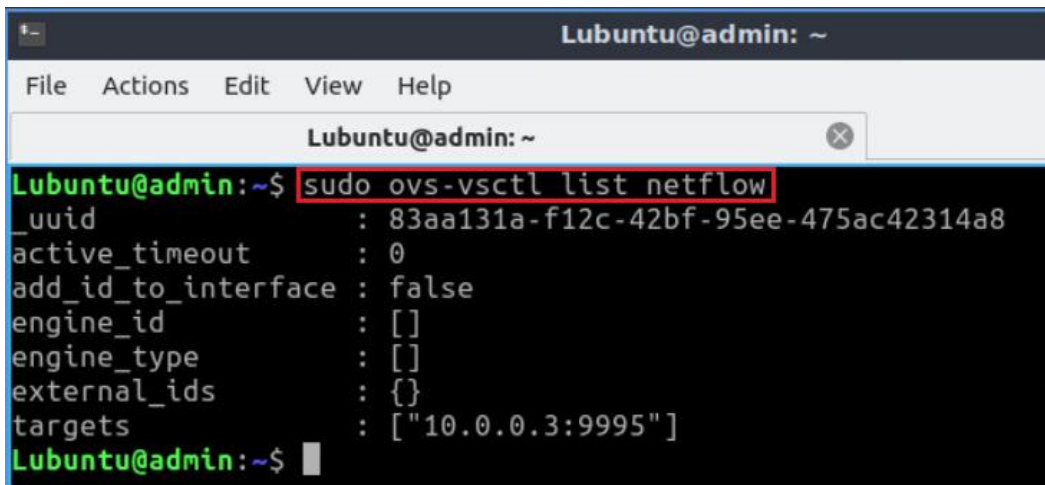
Figure 14. Verifying switch configuration.

Consider the figure above. The figure listed all the existing Open vSwitches. You will notice switch s3 has NetFlow enabled with the ID (83aa131a-f12c-42bf-95ee-475ac42314a8).

You might notice a different NetFlow ID since it is generated randomly each time you enable an exporter.

Step 4. Type the following command to verify NetFlow configuration.

```
sudo ovs-vsctl list netflow
```



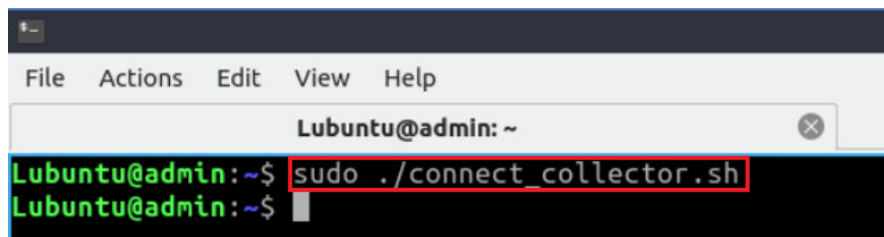
```
Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
Lubuntu@admin:~$ sudo ovs-vsctl list netflow
_uuid          : 83aa131a-f12c-42bf-95ee-475ac42314a8
active_timeout : 0
add_id_to_interface : false
engine_id      : []
engine_type    : []
external_ids   : {}
targets        : ["10.0.0.3:9995"]
Lubuntu@admin:~$
```

Figure 15. Verifying NetFlow configuration.

Consider the figure above. One exporter is running, target collector IP is 10.0.0.3 and the port is 9995.

Step 5. Type the following command to execute a script so that the exporter can send flows to the collector.

```
sudo ./connect_collector.sh
```



```
Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
Lubuntu@admin:~$ sudo ./connect_collector.sh
Lubuntu@admin:~$
```

Figure 16. Connecting collector to the exporter.

The following command was executed in the script.

```
ip route add 10.0.0.0/8 via 172.17.0.1
```

4 Analyzing NetFlow records using Wireshark

In this section, you will analyze NetFlow records in Wireshark.

4.1 Launching Wireshark

Step 1. In Linux terminal, start Wireshark packet analyzer by issuing the following command. A new window will emerge.

```
sudo wireshark
```

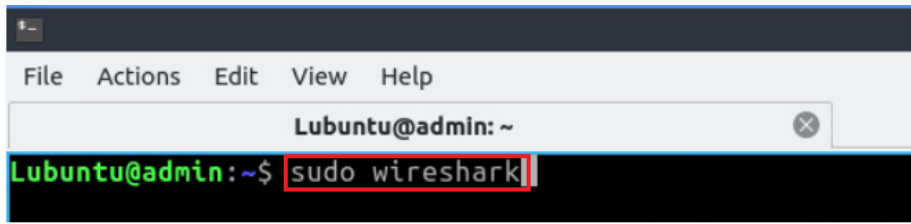


Figure 17. Starting Wireshark packet analyzer.

Step 2. Click on the icon located on the upper left-hand side to start capturing packets on *docker0* interface.

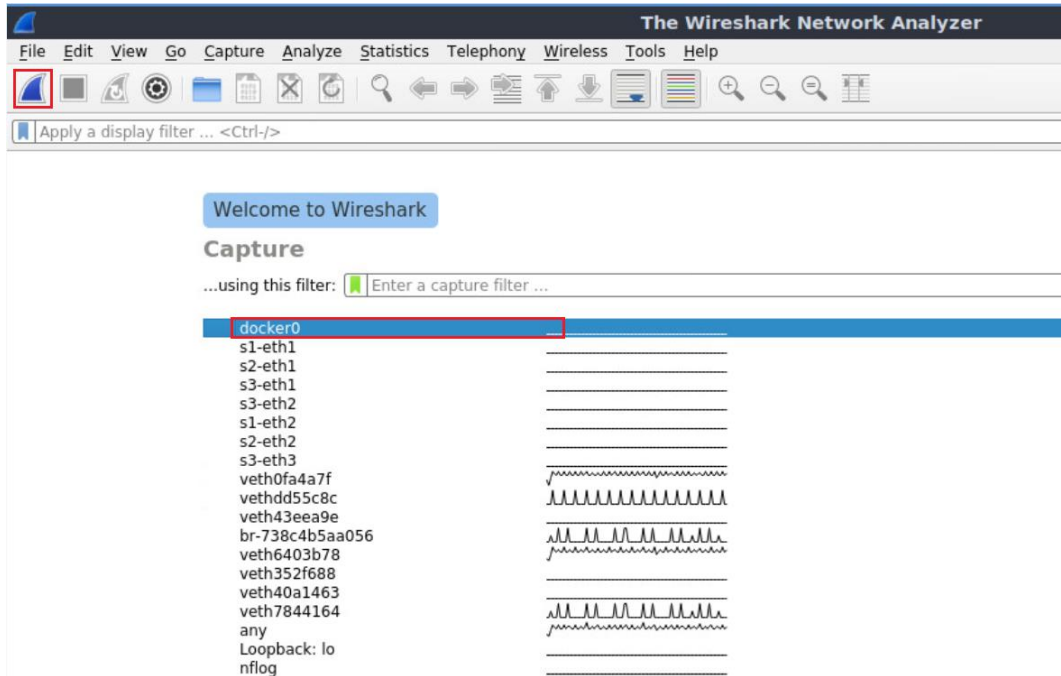


Figure 18. Starting packet capture.

Step 3. In the filter box located on the upper left-hand side, type *udp* to filter UDP packets. Then, press *Enter* to apply the filter.

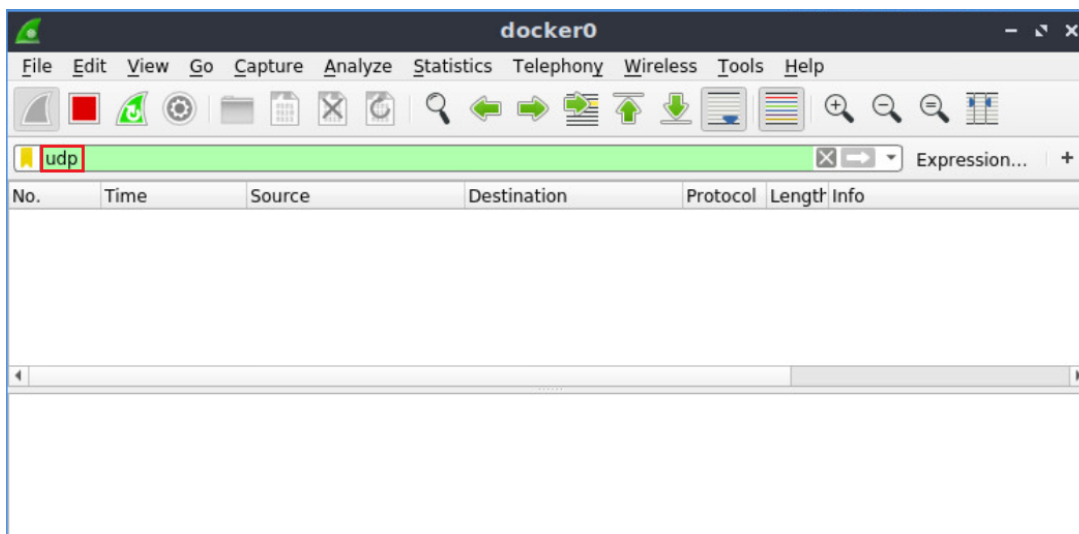


Figure 19. Filtering UDP packets.

4.2 Performing a connectivity test

Step 1. Go back to MiniEdit and hold right-click on host h2 and select *Terminal*. This opens the terminal of host h2 and allows the execution of commands on that host.

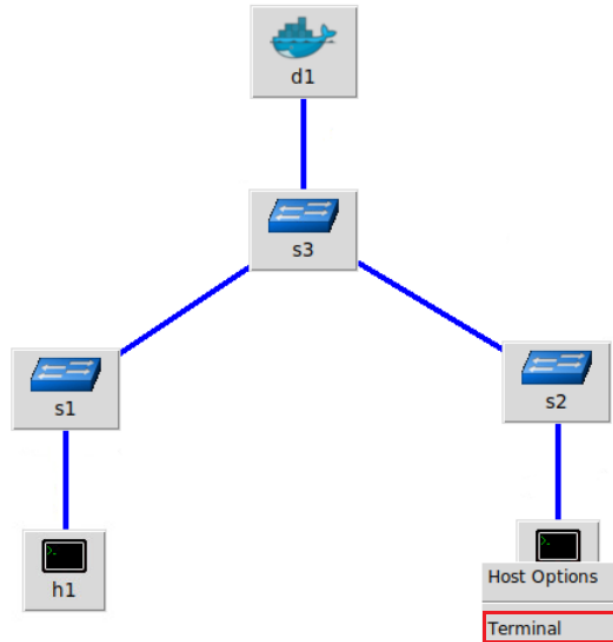


Figure 20. Opening a terminal on host h2.

Step 2. In host h2 terminal, type the following command to run the host in server mode.

```
iperf3 -s
```

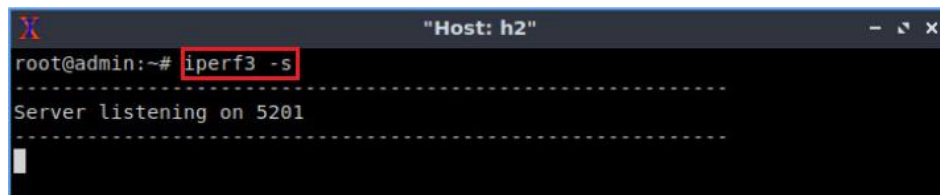


Figure 21. Running host h2 in server mode.

Consider the figure above. The figure shows that host h2 is acting as a server and listening to port 5201.

Step 3. In host h1 terminal, type the following command to run the host in client mode and run an iperf3 test between host h1 and h2.

```
iperf3 -c 10.0.0.2
```



```

Host: h1
root@admin:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 7] local 10.0.0.1 port 36418 connected to 10.0.0.2 port 5201
[ ID] Interval          Transfer          Bitrate          Retr  Cwnd
[ 7]  0.00-1.00        sec  4.22 GBytes      36.3 Gbits/sec   2048  1.28 MBytes
[ 7]  1.00-2.00        sec  4.66 GBytes      40.0 Gbits/sec    740  1.17 MBytes
[ 7]  2.00-3.00        sec  3.70 GBytes      31.8 Gbits/sec     0  1.17 MBytes
[ 7]  3.00-4.00        sec  4.57 GBytes      39.2 Gbits/sec    766  1.21 MBytes
[ 7]  4.00-5.00        sec  4.41 GBytes      37.9 Gbits/sec     0  1.33 MBytes
[ 7]  5.00-6.00        sec  4.85 GBytes      41.6 Gbits/sec    90  1.39 MBytes
[ 7]  6.00-7.00        sec  4.77 GBytes      41.0 Gbits/sec   607  1.28 MBytes
[ 7]  7.00-8.00        sec  4.01 GBytes      34.5 Gbits/sec     0  1.44 MBytes
[ 7]  8.00-9.00        sec  4.21 GBytes      36.1 Gbits/sec   577  1.27 MBytes
[ 7]  9.00-10.00       sec  4.71 GBytes      40.4 Gbits/sec   611  1.38 MBytes
-----
[ ID] Interval          Transfer          Bitrate          Retr
[ 7]  0.00-10.00       sec  44.1 GBytes      37.9 Gbits/sec   5439
sender
    
```

Figure 22. Running host h1 in client mode.

Consider the figure above. The test runs for ten seconds with interval of one second.

4.3 Visualizing NetFlow packets

Step 1. By default, NetFlow records are exported using User Datagram Protocol (UDP).

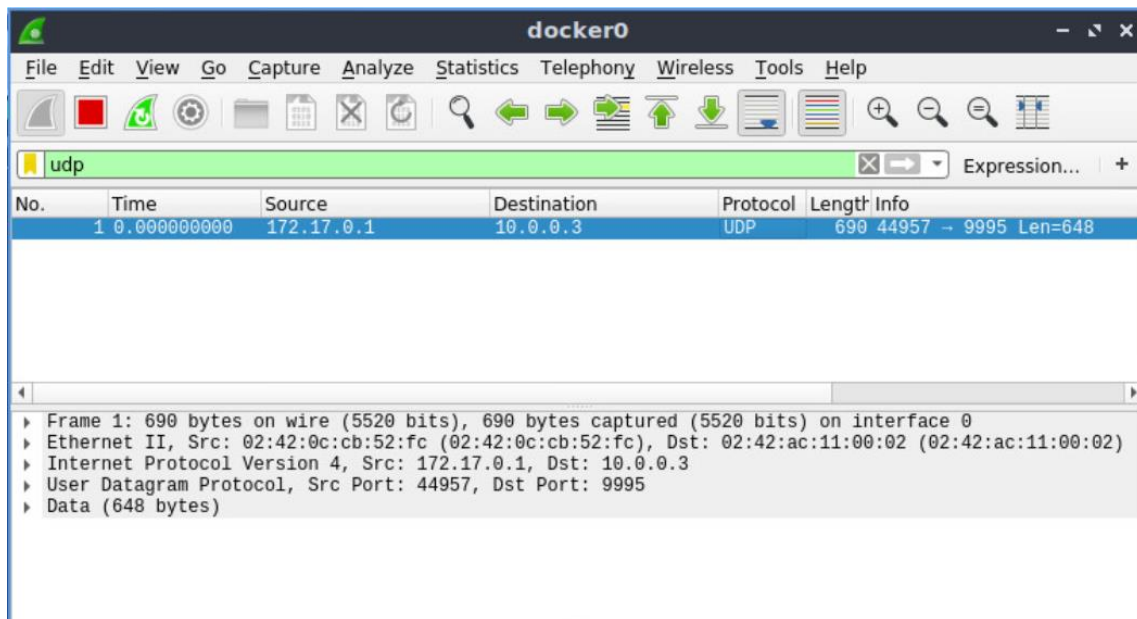


Figure 23. Verifying packet capture.

Consider the figure above. You will notice a UDP packet.

Step 2. Wireshark provides a very powerful feature of decoding the captured packets into user specified formats. Right-click on UDP packet and select decode as.

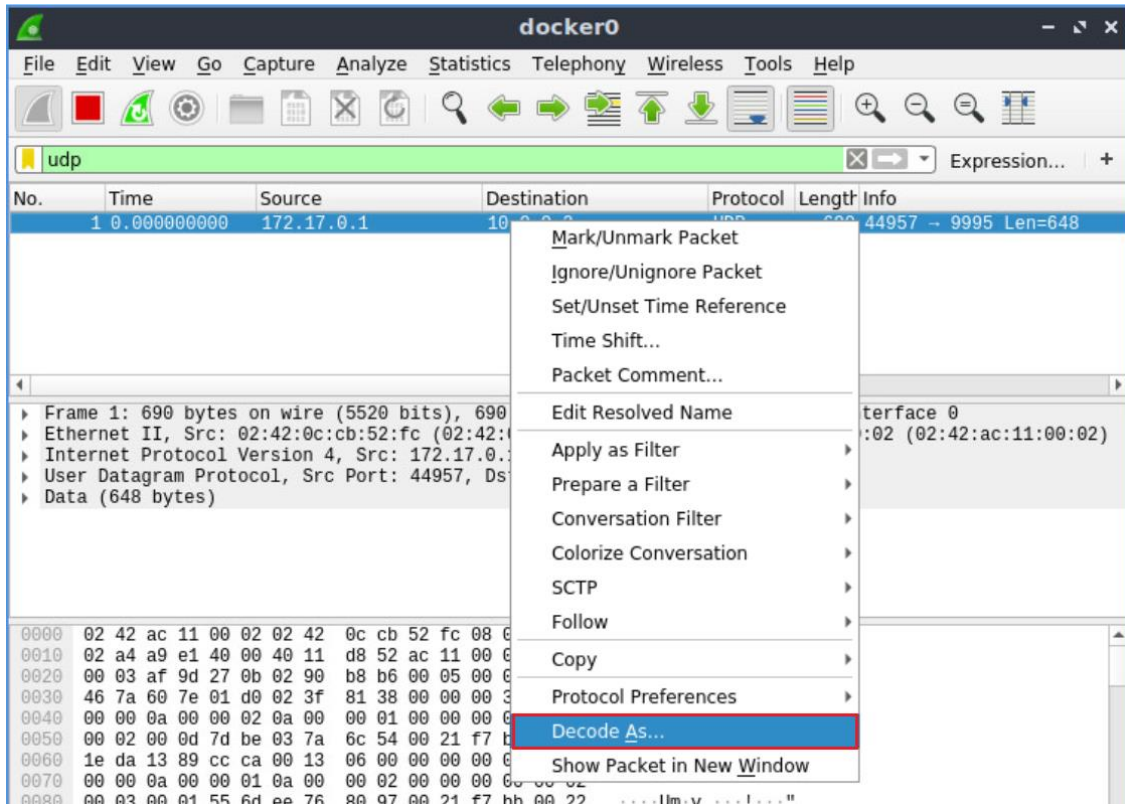


Figure 24. Decoding UDP packet.

Step 3. From the drop-down options, click on *none* and select CFLOW for the current field. Then, click OK.

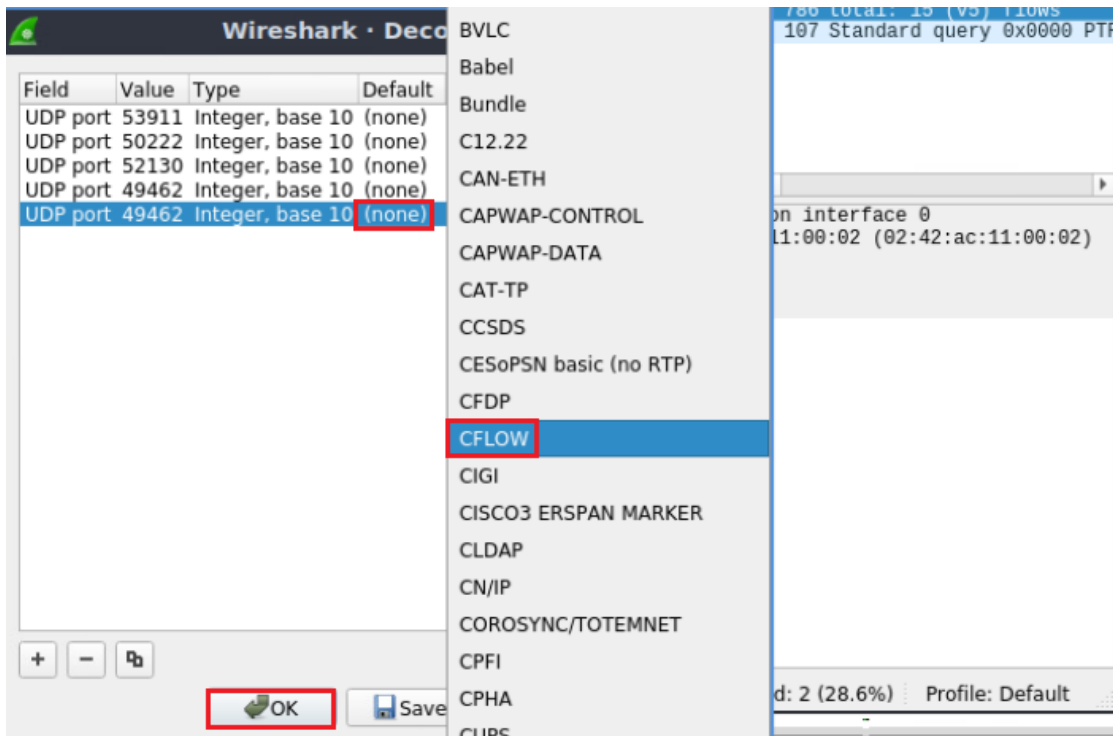


Figure 25. Decoding as CFLOW.

The decode functionality of Wireshark temporarily diverts the specific protocol dissections. CFLOW shows all the NetFlow information.

Step 4. You will notice a new field called Cisco NetFlow/IPFIX which includes all the information regarding NetFlow.

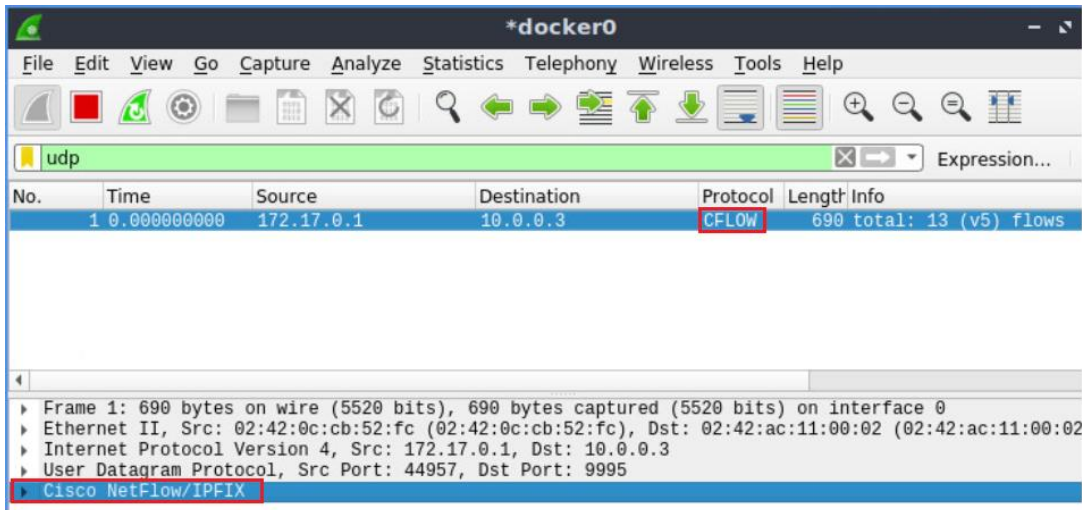


Figure 26. Verifying NetFlow information.

Step 5. Click on the arrow located on the leftmost side of the field called *Cisco NetFlow/IPFIX*. A list will be displayed.

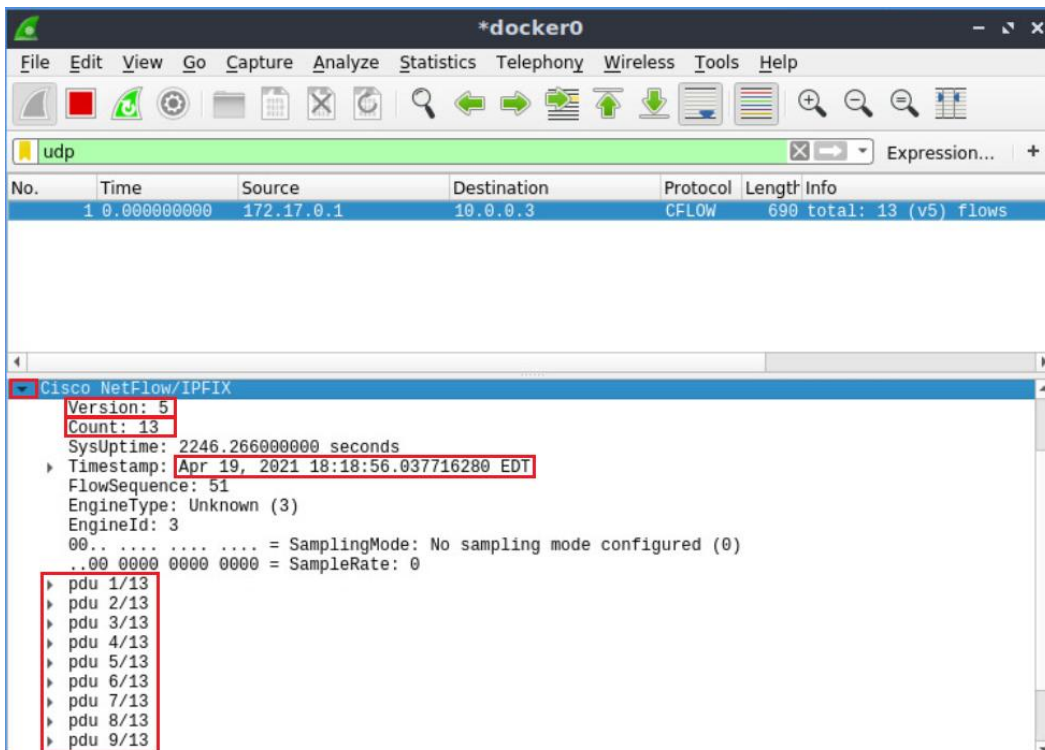


Figure 27. Verifying NetFlow information.

Consider the figure above. The figure shows the NetFlow version (version 5), total flow count (13), Timestamp and other information of the packet header. You will also notice all the flows listed there.

You might get different number of flows.

Step 6. Click on flow 1 (pdu 1/13).

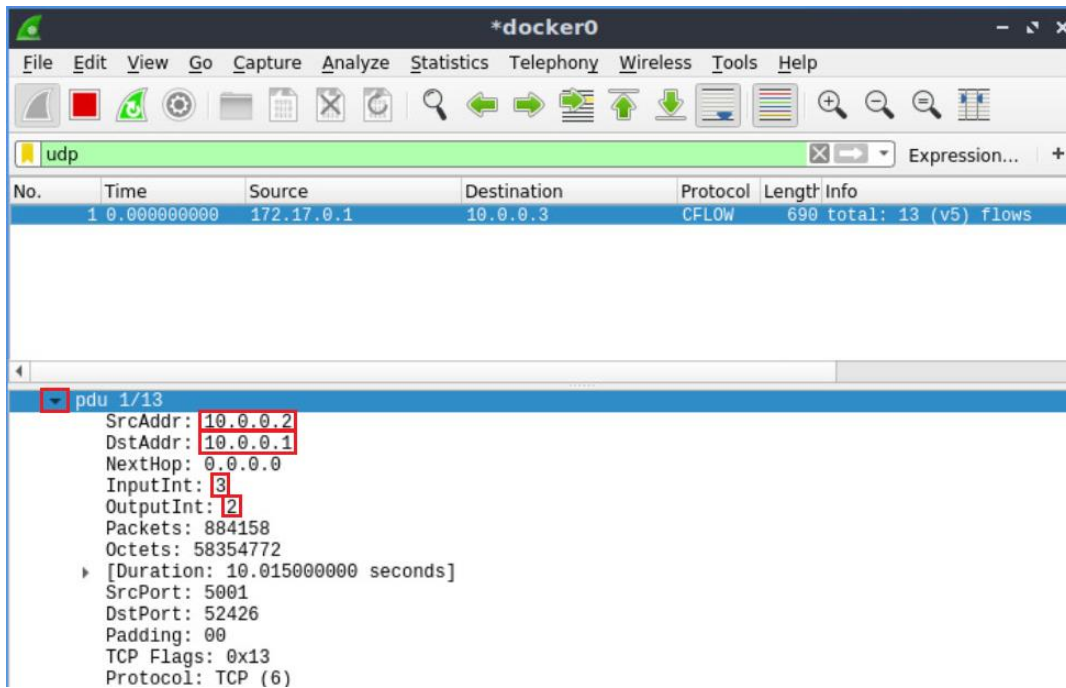


Figure 28. Verifying flow record.

Consider the figure above. The figure shows the list of NetFlow information for a single flow which includes source and destination IP addresses, in port, out port, total packets, total bytes (octets), duration, source and destination ports and all other information. For the flow, source address is 10.0.0.2, destination address is 10.0.0.1, in_port=3 and out_port=2. Switch s3 is the exporter. When traffic is coming from host h2, in_port is s3-eth3 and forwarding towards host h1 using out_port s3-eth2.

Step 7. Verify flow 13 (pdu 13/13).

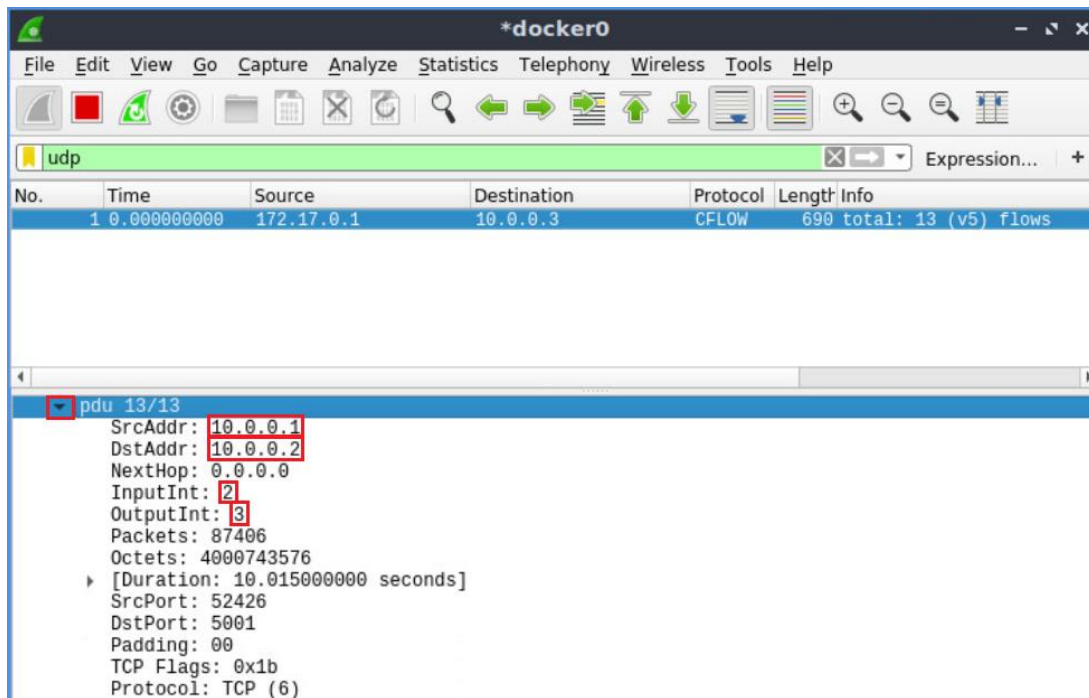


Figure 29. Verifying flow record.

Consider the figure above. For the flow, source address is 10.0.0.1, destination address is 10.0.0.2, in_port=2 and out_port=3.

If you get different number of flows then check out any two flows and verify two records (from 10.0.0.1 to 10.0.0.2 and 10.0.0.2 to 10.0.0.1).

This concludes Lab 2. Close Wireshark window, stop the emulation and then exit out of MiniEdit and the Linux terminal.

References

1. Cisco, "Introduction to Cisco IOS NetFlow – A technical overview", [Online].
https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html
2. B. Claise, "RFC 3954: Cisco systems NetFlow services export version 9", (2004).
<https://tools.ietf.org/html/rfc3954>
3. Plixer, "Open vSwitch NetFlow", [Online].
<https://www.plixer.com/blog/open-vswitch-netflow/>
4. FlowTraQ, "NetFlow export configuration", [Online].
http://support.flowtraq.com/Documentation/Q3_16/webhelp/content/device_configuration.html
5. Vivek Ratan, Kin Fun Li, "NetFlow: Network monitoring and intelligence gathering", International conference on P2P, parallel, grid, cloud and Internet computing, (2016).
6. Omar Santos, "Network security with NetFlow and IPFIX", (2016).
7. Flowmon, "NetFlow / IPFIX monitoring", [Online].

<https://www.flowmon.com/en/solutions/network-and-cloud-operations/netflow-ipfix>

8. ManageEngine NetFlow Analyzer, "*What is NetFlow?*", [Online].

<https://www.manageengine.com/products/netflow/what-is-netflow.html>



UNIVERSITY OF
SOUTH CAROLINA

NETWORK MANAGEMENT

Lab 3: Introduction to IPFIX

Document Version: **07-08-2022**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Introduction	3
1.1 Introduction to IPFIX	4
1.2 IPFIX architecture	5
2 Lab topology.....	5
2.1 Lab settings.....	6
2.2 Loading a topology	6
3 Launching IPFIX exporter	9
4 Analyzing IPFIX records using Wireshark	11
4.1 Launching Wireshark.....	11
4.2 Performing a connectivity test	13
4.3 Visualizing IPFIX packets.....	14
References	21

Overview

This lab introduces IP Flow Information Export (IPFIX) which is a network protocol used to collect IP flow statistics and generate relevant data records. IPFIX enabled switches or routers are called IPFIX exporters, which examine each packet and create flows from these packets. These collected flows are exported to an external device known as IPFIX collector which then organizes the flow records into a format that allows the administrator to further analyze the traffic². The focus of this lab is to explore how IPFIX exporter and collector work in Open Virtual Switch (Open vSwitch) and analyze the collected flows using Wireshark packet analyzer.

Objectives

By the end of this lab, you should be able to:

1. Understand the concept of IPFIX.
2. Understand how IPFIX works in Open vSwitch.
3. Enable IPFIX in Open vSwitch.
4. Analyze IPFIX records using Wireshark.

Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Launching IPFIX exporter.
4. Section 4: Analyzing IPFIX records using Wireshark.

1 Introduction

IPFIX was introduced for the desire of vendors to push away from the Cisco-driven standards and to provide a much more open and flexible flow gathering datagram and

environment. It is an enhanced version of NetFlow v9, considered as NetFlow v10. IPFIX introduces several extensions such as integrating more information into its exporting process. This means customers don't need to invest in an additional device to handle more complicated aspects of data collection and can run more efficient tests on their networks. IPFIX has a wide range of applications: Information from the program can help network administrators monitor bandwidth, keep track of threats to network security, and figure out usage amounts for various users. NetFlow v9 defines 127 field types but IPFIX defines 238. It can use variable-length fields which allows IPFIX to collect data like URLs and messages¹.

1.1 Introduction to IPFIX

IPFIX protocol provides network administrators with access to IP flow information. It is a network flow standard defined by Internet Engineering Task Force (IETF). It was initiated to create a common, universal standard of export for flow information which defines how flow information should be laid out and transferred from an exporter to a collector. IPFIX supports Stream Control Transmission Protocol (SCTP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP)¹.

IPFIX uses templates to provide access to observations of IP packet flows in a flexible and extensible manner. Since the template mechanism is flexible, it allows the export of only the required fields from the flows to the collector. This helps to reduce the exported flow data volume and provides possible memory savings for the exporter and the collector³. IPFIX-enabled devices can send IPFIX messages to the collector. Each IPFIX message contains a message header and one or more template or data sets. A template provides the description of the fields that will be present in the future data sets. Templates are composed of Information Element (IE) and length pairs. IE provides field type information for each template. The sets can be any of these three possible types:

Template Sets: It contains one or more templates used to describe the layout of flow records which includes all flow collection use cases such as the traditional five-tuple (source IP address, destination IP address, source port, destination port, IP protocol), various counters (packet delta counts, total connection counts, top talkers), flow meta data information such as ingress, egress interfaces and flow direction¹.

Options Template sets: The Options Template record gives the exporter the ability to provide additional information to the collector that would not be possible with flow records alone such as information about the collection infrastructure, flow keys used by the exporter and so on².

Data sets: Data records are sent in data sets.

IPFIX exporter sends templates to the collector periodically which provides a flexible design to the record format. The collector receives flows with packets and uses templates to decode the information in the packets.

1.2 IPFIX architecture

An Observation Point is a location in the network where packets can be observed. A Flow is defined as a set of packets or frames passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties. These common properties may include packet header fields, such as source and destination IP addresses, port numbers, packet properties, and information derived by IP packet forwarding. Every observation point is associated with an observation domain. Each IPFIX device has an observation domain ID. By default, the domain ID is 0². IPFIX uses the following architecture terminology:

Metering Process (MP): Consists of a set of functions that includes packet header capturing, timestamping, sampling, classifying, and maintaining flow records at an observation point. It also passes complete flow records to an Exporting Process (EP)².

EP: Sends IPFIX messages to one or more Collecting Processes (CPs). The flow records in the messages are generated by one or more MPs.

CP: receives IPFIX Messages from one or more EPs.

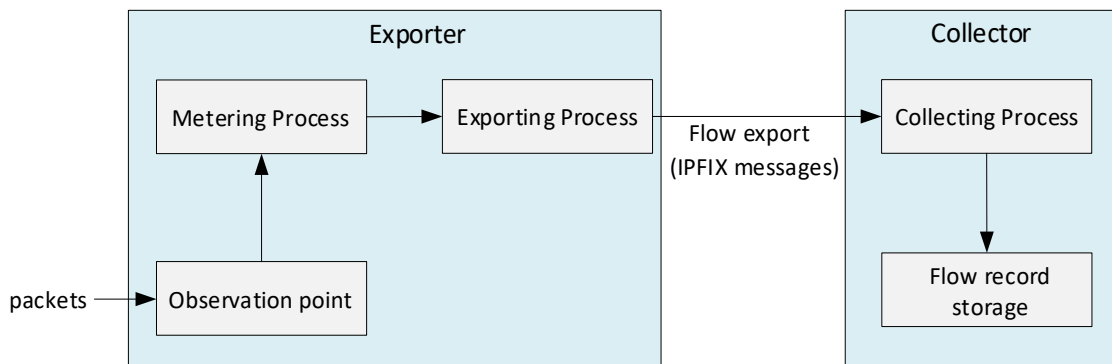


Figure 1. IPFIX architecture.

Figure 1 shows the architecture of IPFIX. The Exporter observes packets, turns packets into flow records and sends them to the collector as IPFIX messages. The collector receives the IPFIX messages, identifies, decodes, and stores them for analysis.

2 Lab topology

Consider Figure 2. There are three switches, two end hosts and a docker container. Switch s3 is acting as IPFIX exporter and the docker d1 will collect the flows.

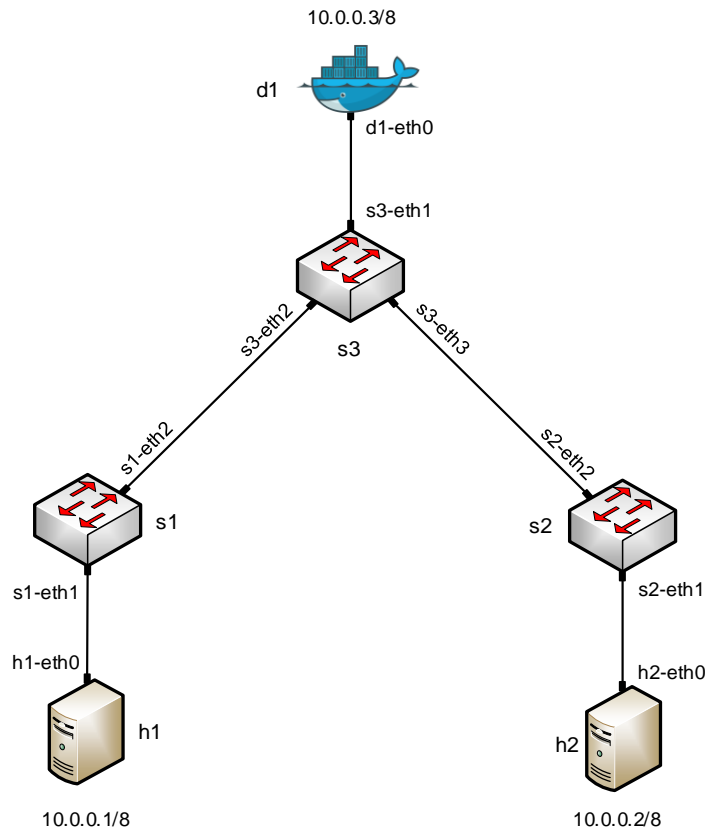


Figure 2. Lab topology.

2.1 Lab settings

The devices should be configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
d1	d1-eth0	10.0.0.3	/8

2.2 Loading a topology

Step 1. Click on the Client tab to access the Client PC.

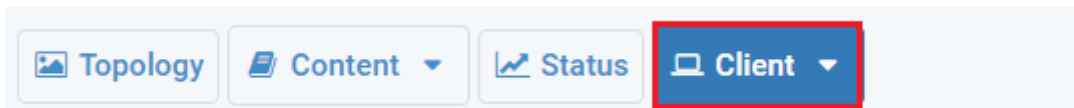


Figure 3. Accessing the Client PC.

Step 2. Start by launching MiniEdit by clicking on desktop's shortcut. When prompted for a password, type `password`.

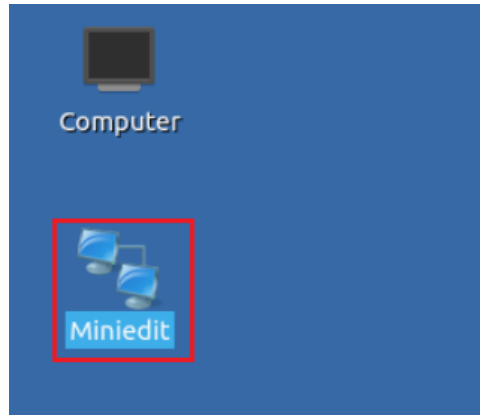


Figure 4. MiniEdit shortcut.

Step 3. On MiniEdit's menu bar, click on *File* then open to load the lab's topology. Open the directory called *lab3* and select the file *lab3.mn*. Then, click on *Open* to open the topology.

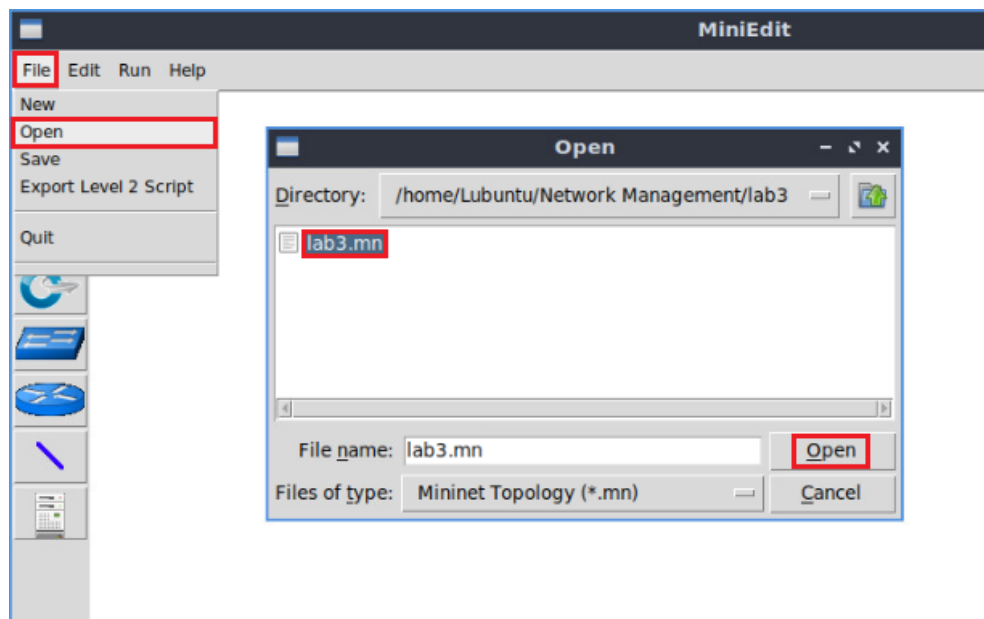


Figure 5. MiniEdit's Open dialog.

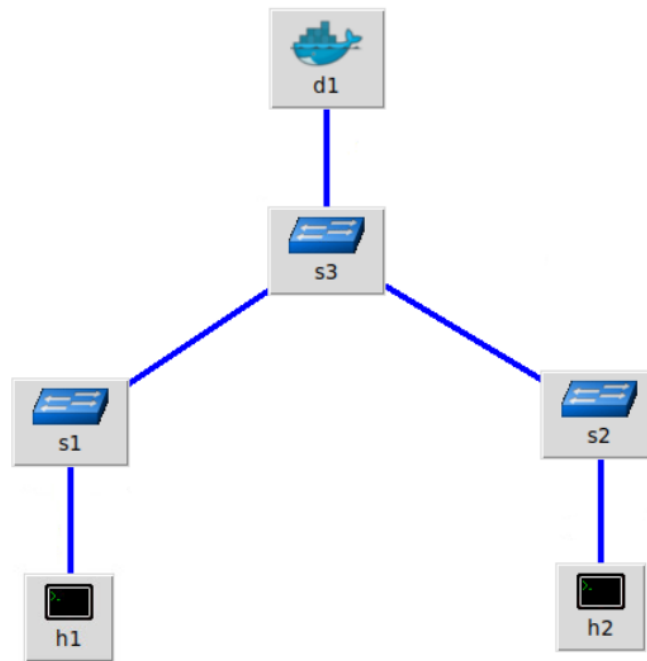


Figure 6. MiniEdit's topology.

Step 4. To proceed with the emulation, click on the *Run* button located on the lower left-hand side.

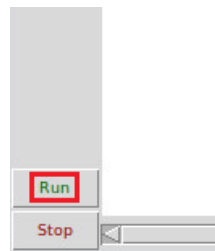


Figure 7. Starting the emulation.

Step 5. Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

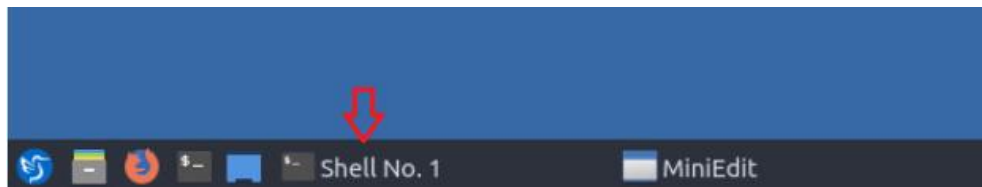


Figure 8. Opening Mininet's terminal.

Step 6. Issue the following command to display the interface names and connections.

```
links
```

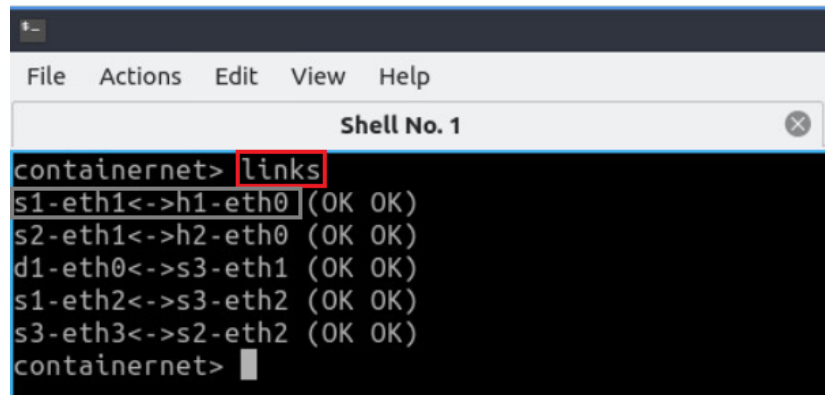


Figure 9. Displaying network interfaces.

In figure 9, the link displayed within the gray box indicates that interface eth1 of switch s1 connects to interface eth0 of host h1 (i.e., *s1-eth1<->h1-eth0*).

3 Launching IPFIX exporter

Step 1. Open the Linux terminal.



Figure 10. Opening Linux terminal.

Step 2. Execute the following script in order to start IPFIX exporter. When prompted for a password, type `password`.

```
sudo ./ipfix.sh
```

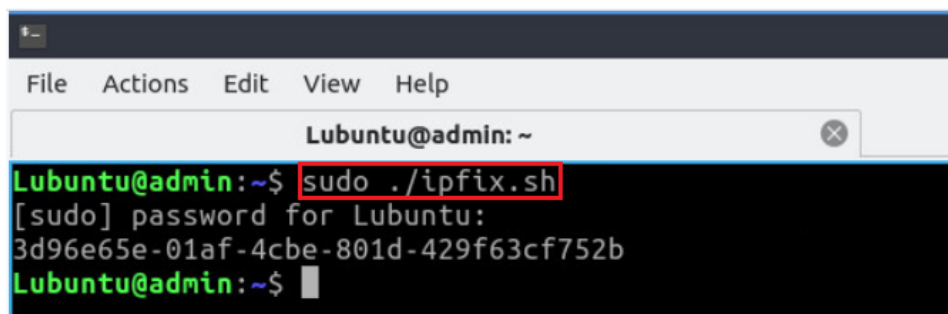


Figure 11. Starting IPFIX exporter.

The following commands were executed in the script.

```
ovs-vsctl -- set bridge s3 ipfix=@if -- --id=@if create IPFIX targets="\10.0.0.3:9995\"
```

The command creates an IPFIX ID and attaches it to switch s3. Switch s3 is acting as an exporter and transmits data to the collector. Docker d1 is the collector IP and the port is the UDP port 9995.

Step 3. Type the following command to show switch configuration.

```
sudo ovs-vsctl list bridge
```

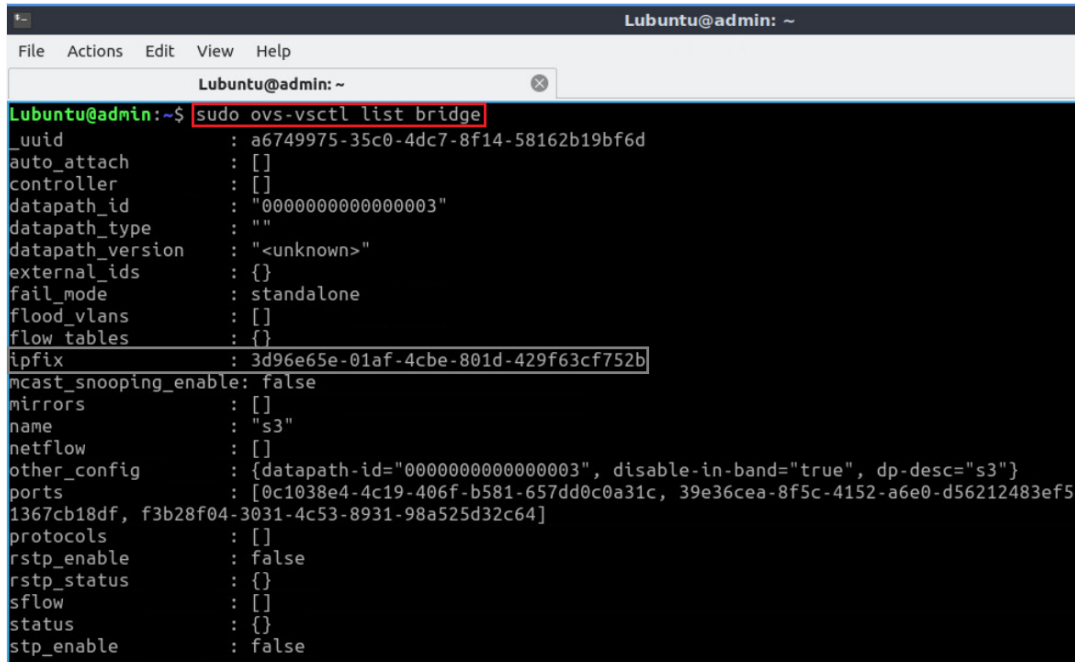


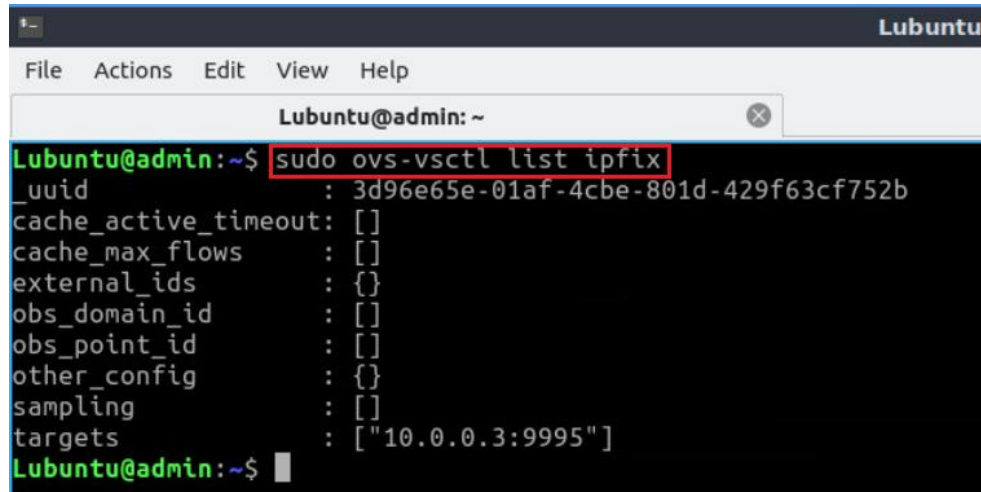
Figure 12. Verifying switch configuration.

Consider the figure above. The figure listed all the existing Open vSwitches. You will notice switch s3 has IPFIX enabled with an ID (3d96e65e-01af-4cbe-801d-429f63cf752b).

You might notice a different IPFIX ID since it is generated randomly each time you enable an exporter.

Step 4. Type the following command to verify IPFIX configuration.

```
sudo ovs-vsctl list ipfix
```

```

Lubuntu@admin:~$ sudo ovs-vsctl list ipfix
_uuid          : 3d96e65e-01af-4cbe-801d-429f63cf752b
cache_active_timeout: []
cache_max_flows  : []
external_ids    : {}
obs_domain_id   : []
obs_point_id    : []
other_config    : {}
sampling        : []
targets         : ["10.0.0.3:9995"]
Lubuntu@admin:~$

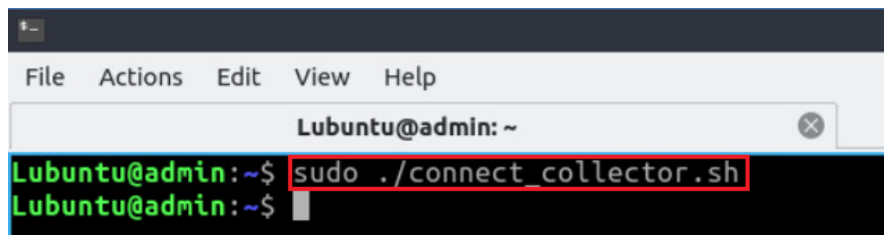
```

Figure 13. Verifying IPFIX configuration.

Consider the figure above. One exporter is running, target collector IP is 10.0.0.3 and the port is 9995.

Step 5. Type the following command to execute a script so that the exporter can send flows to the collector.

```
sudo ./connect_collector.sh
```



```

Lubuntu@admin:~$ sudo ./connect_collector.sh
Lubuntu@admin:~$

```

Figure 14. Connecting collector to the exporter.

The following command was executed in the script.

```
ip route add 10.0.0.0/8 via 172.17.0.1
```

4 Analyzing IPFIX records using Wireshark

In this section, you will analyze IPFIX records in Wireshark.

4.1 Launching Wireshark

Step 1. In Linux terminal, start Wireshark packet analyzer by issuing the following command. A new window will emerge.

```
sudo wireshark
```

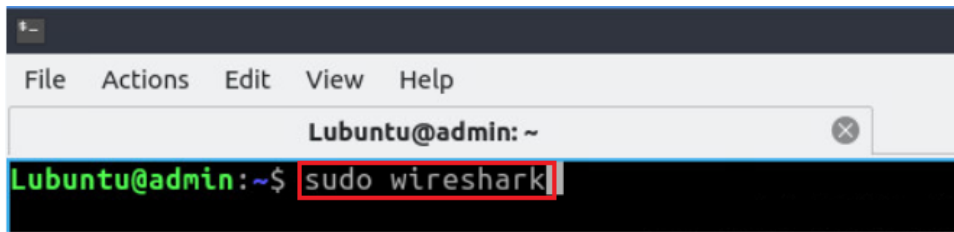


Figure 15. Starting Wireshark packet analyzer.

Step 2. Click on the icon located on the upper left-hand side to start capturing packets on docker0.

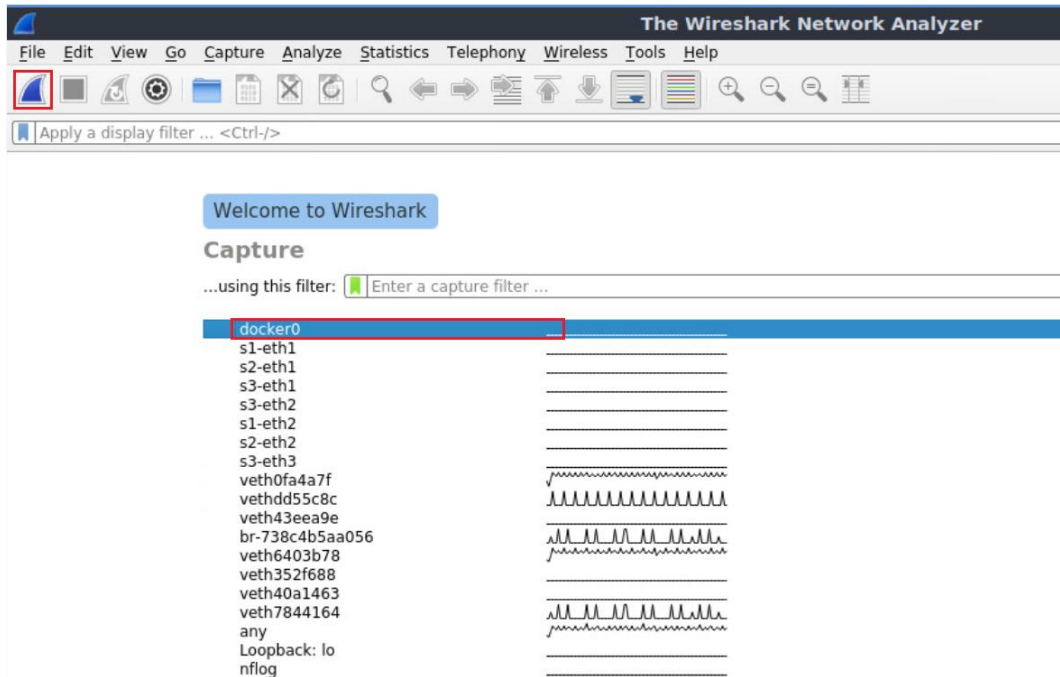


Figure 16. Starting packet capture.

Step 3. In the filter box located on the upper left-hand side, type *udp* to filter UDP packets. Then, press *Enter* to apply the filter.

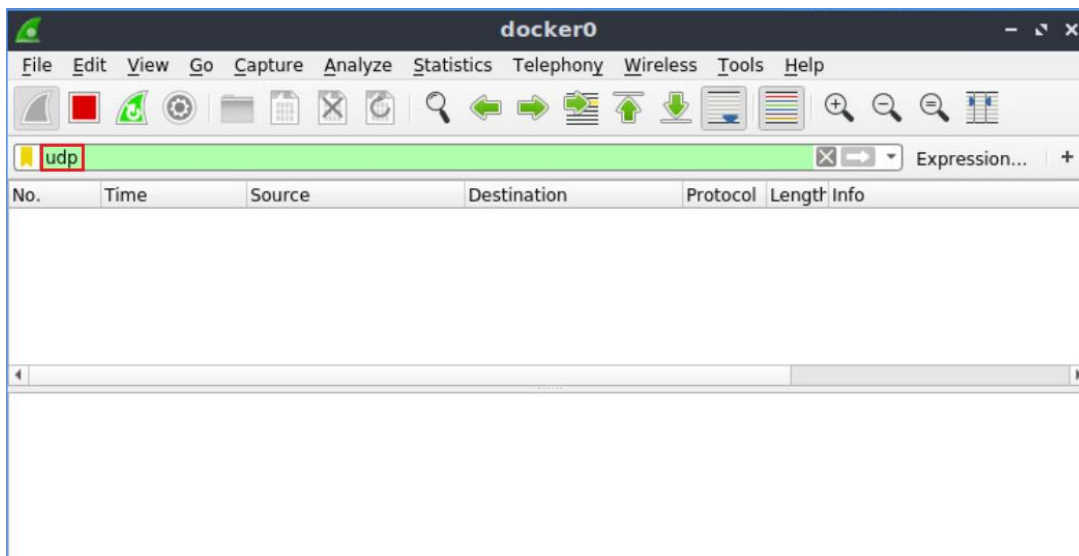


Figure 17. Filtering UDP packets.

4.2 Performing a connectivity test

Step 1. Go back to MiniEdit and hold right-click on host h2 and select *Terminal*. This opens the terminal of host h2 and allows the execution of commands on that host.

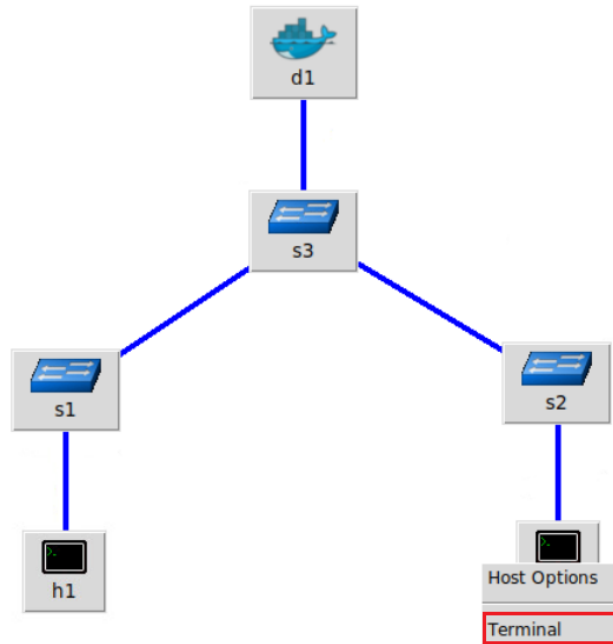


Figure 18. Opening a terminal on host h2.

Step 2. In host h2 terminal, type the following command to run the host in server mode.

```
iperf3 -s
```

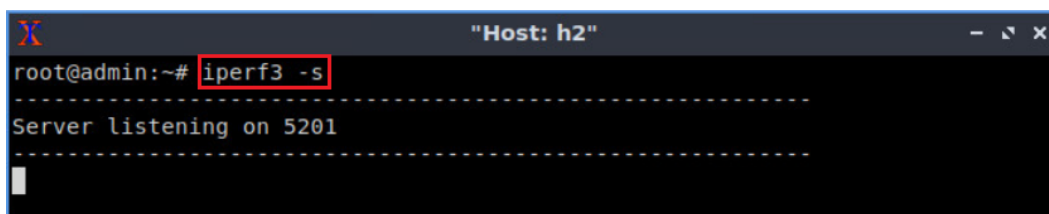


Figure 19. Running host h2 in server mode.

Consider the figure above. The figure shows that host h2 is acting as a server and listening to port 5201.

Step 3. Follow step 1 to open a terminal in host h1. Type the following command to run the host in client mode and run an iperf test between hosts h1 and h2.

```
iperf3 -c 10.0.0.2 -t 30
```

```

Host: h1
root@admin:~# iperf3 -c 10.0.0.2 -t 30
Connecting to host 10.0.0.2, port 5201
[ 7] local 10.0.0.1 port 59340 connected to 10.0.0.2 port 5201
[ ID] Interval           Transfer             Bitrate             Retr  Cwnd
[ 7]  0.00-1.00   sec  4.46 GBytes        38.3 Gbits/sec      0    7.41 MBytes
[ 7]  1.00-2.00   sec  4.58 GBytes        39.3 Gbits/sec      0    8.16 MBytes
[ 7]  2.00-3.00   sec  4.27 GBytes        36.7 Gbits/sec     212   4.00 MBytes
[ 7]  3.00-4.00   sec  4.89 GBytes        42.0 Gbits/sec      0    4.20 MBytes
[ 7]  4.00-5.00   sec  4.66 GBytes        40.0 Gbits/sec      0    4.35 MBytes
[ 7] 26.00-27.00  sec  4.99 GBytes        42.9 Gbits/sec     436   1.89 MBytes
[ 7] 28.00-29.00  sec  4.30 GBytes        36.9 Gbits/sec      0    1.69 MBytes
[ 7] 29.00-30.00  sec  4.68 GBytes        40.2 Gbits/sec      0    1.89 MBytes
-----
[ ID] Interval           Transfer             Bitrate             Retr
[ 7]  0.00-30.00  sec  137 GBytes        39.1 Gbits/sec     4101
[ 7]  0.00-30.04  sec  136 GBytes        39.0 Gbits/sec
r
iperf Done.
root@admin:~#
  
```

Figure 20. Running host h1 in client mode.

Consider the figure above. The test runs for 30 seconds with interval of one second.

4.3 Visualizing IPFIX packets

Step 1. Verify packet capturing in Wireshark. You will notice that IPFIX records are exported using User Datagram Protocol (UDP).

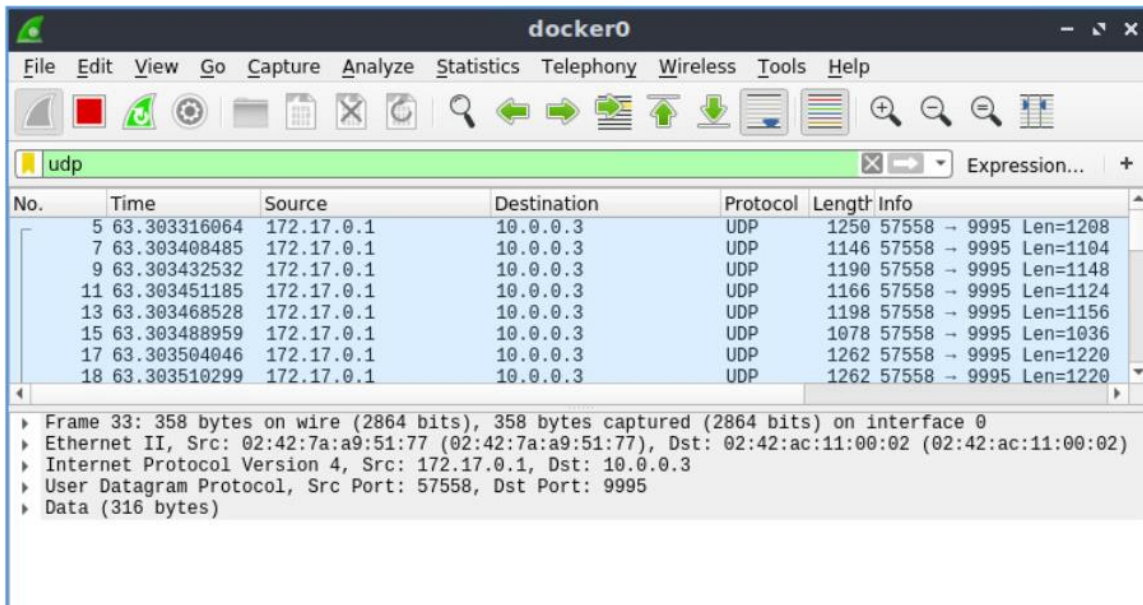


Figure 21. Verifying packet capture.

Step 2. Wireshark provides a very powerful feature of decoding the captured packets into user specified formats. Right-click on first UDP packet (length 1250) and select decode as.

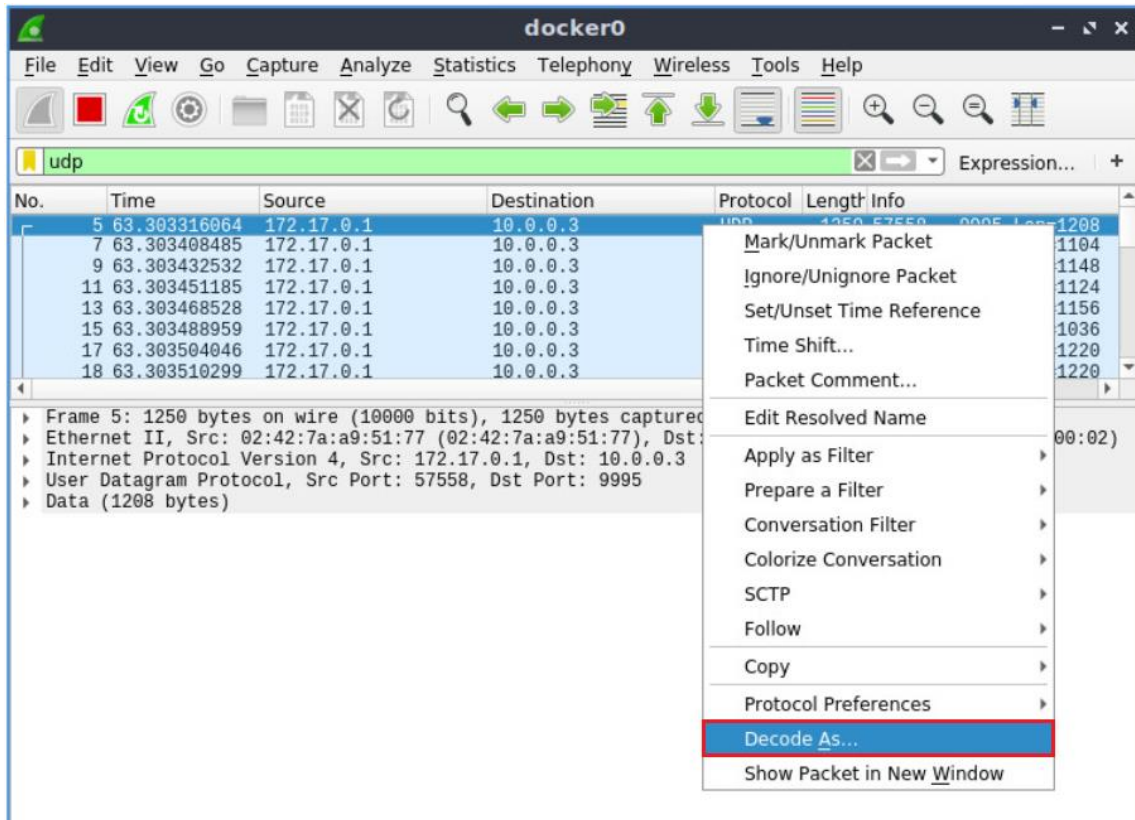


Figure 22. Decoding UDP packet.

Step 3. Select the last entry and click on none. From the drop-down options select CFLOW for the current field and click OK.

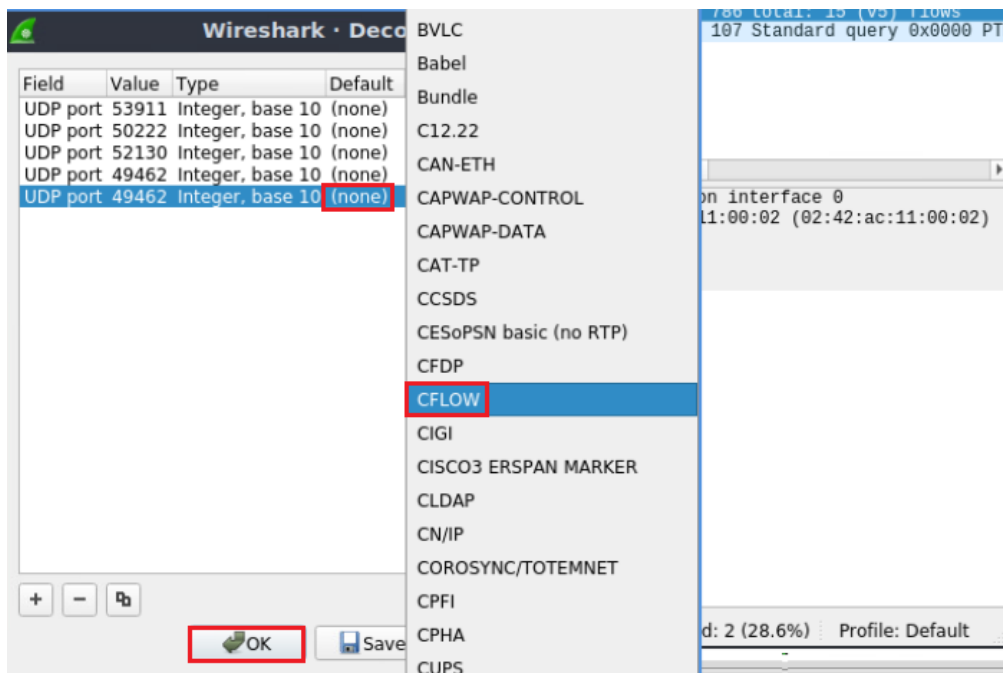


Figure 23. Decoding as CFLOW.

The decode functionality of Wireshark temporarily diverts the specific protocol dissections. CFLOW shows all the NetFlow/IPFIX information.

It may take some moments to decode all the packets.

Step 4. You will notice a new field called Cisco NetFlow/IPFIX which includes all the information regarding IPFIX.

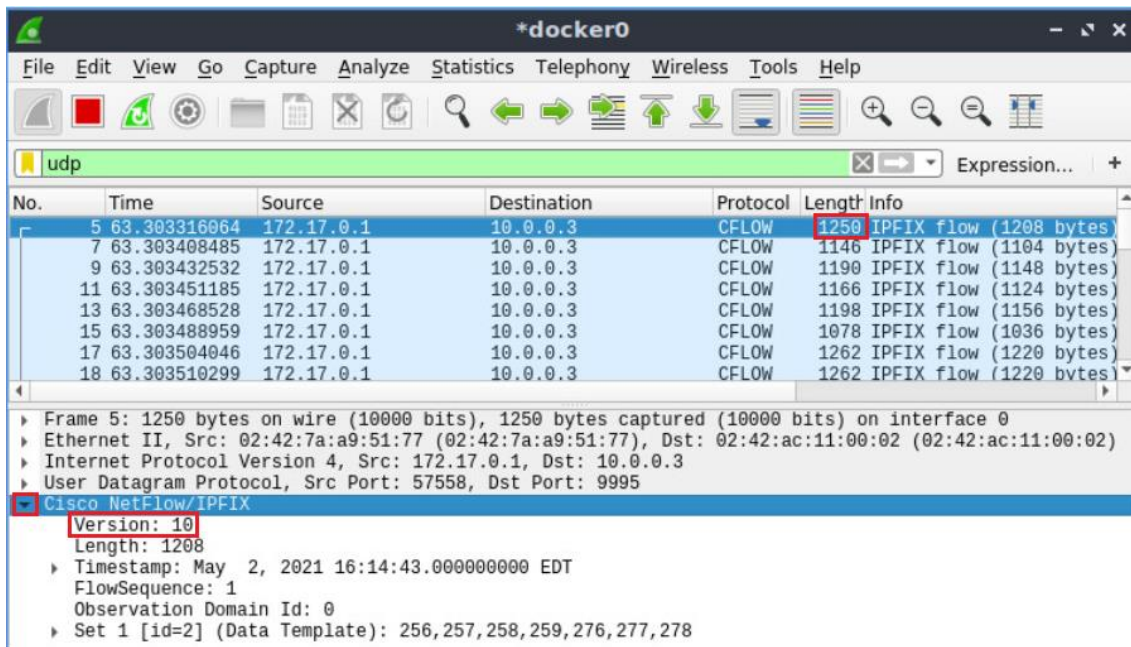


Figure 24. Verifying IPFIX information.

Consider the figure above. The figure shows the NetFlow version (version 10), length of the packet, timestamp, and other information of the packet header. You will also notice all the data template listed here.

Step 5. Click on the arrow located on the leftmost side of the field called Set 1. A list will be displayed.

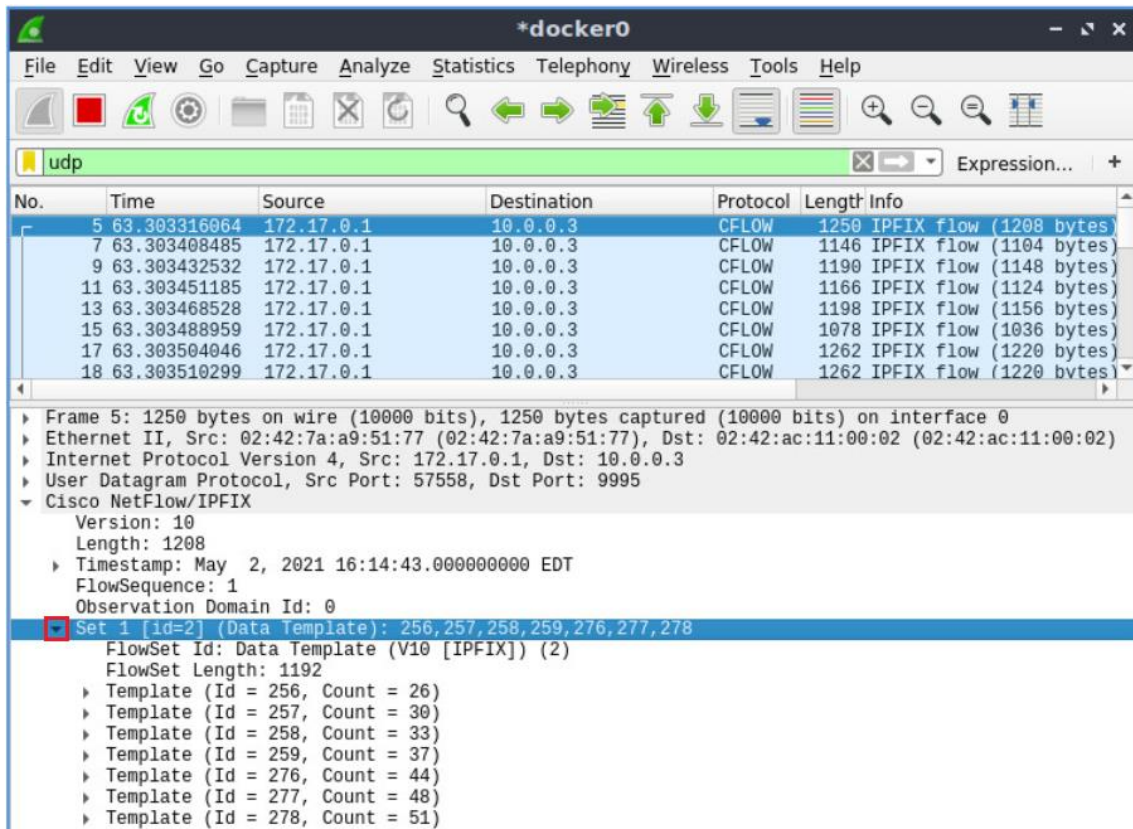


Figure 25. Verifying IPFIX templates.

Consider the figure above. The figure shows IPFIX templates.

Step 6. Click on the arrow located on the leftmost side of the field called *Template (Id=256, Count=26)*. A list will be displayed.

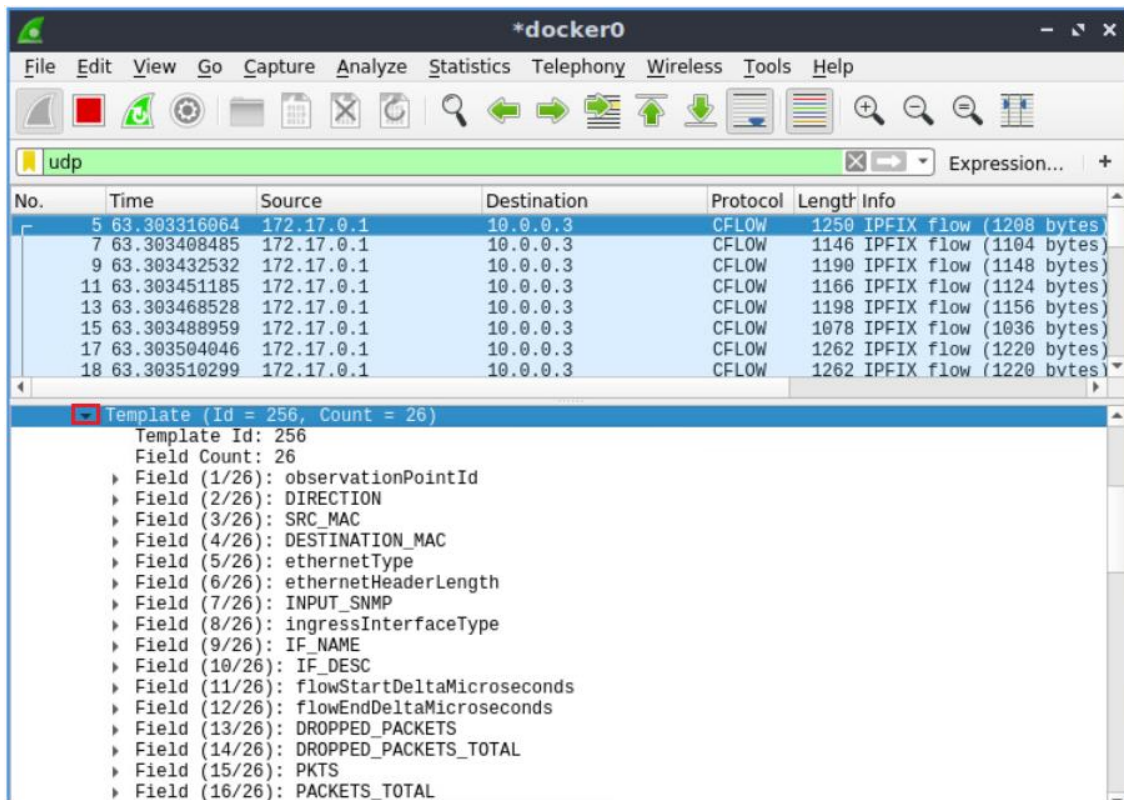


Figure 26. Verifying IPFIX template.

Consider the figure above. The figure shows the template for IPFIX information which includes observation point ID, source and destination MAC address, source and destination IP address, interface name and other information.

Step 7. Click on the CFLOW packet that has a length of 358 bytes.

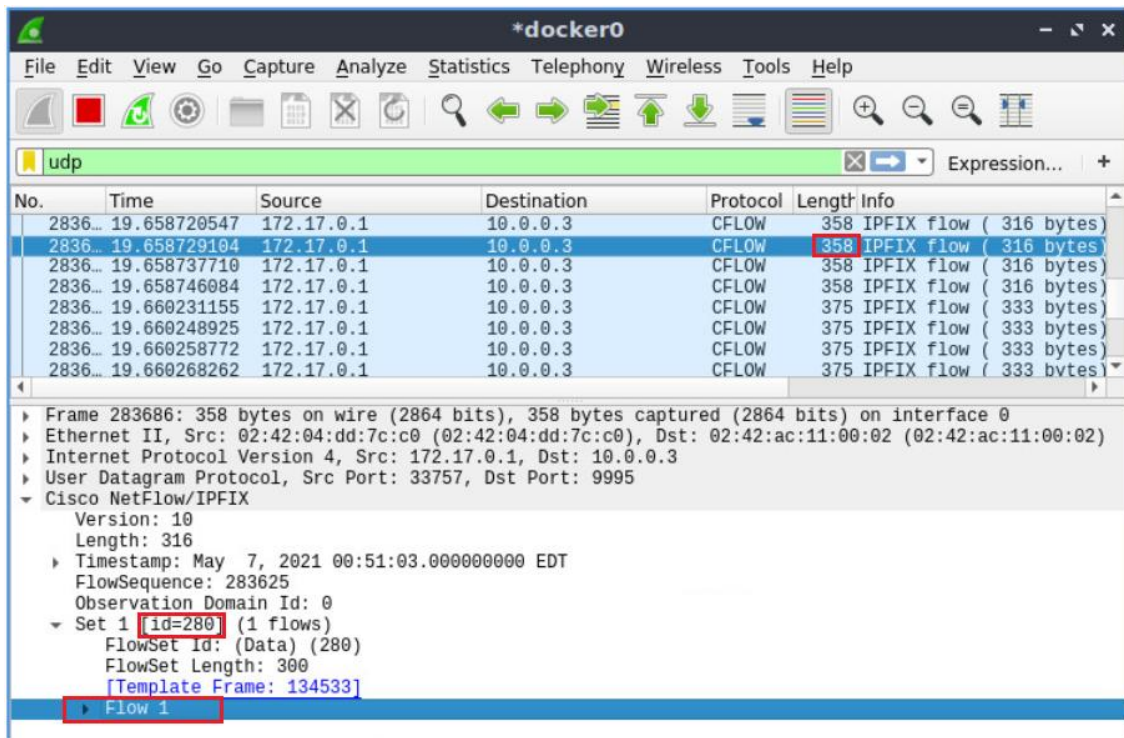


Figure 27. Verifying flow record.

Consider the figure above. The figure shows a flow has been detected. Data ID is 280 which means the template ID is 280.

Step 8. Click on the arrow located on the leftmost side of the field called *Flow 1*. A list will be displayed.

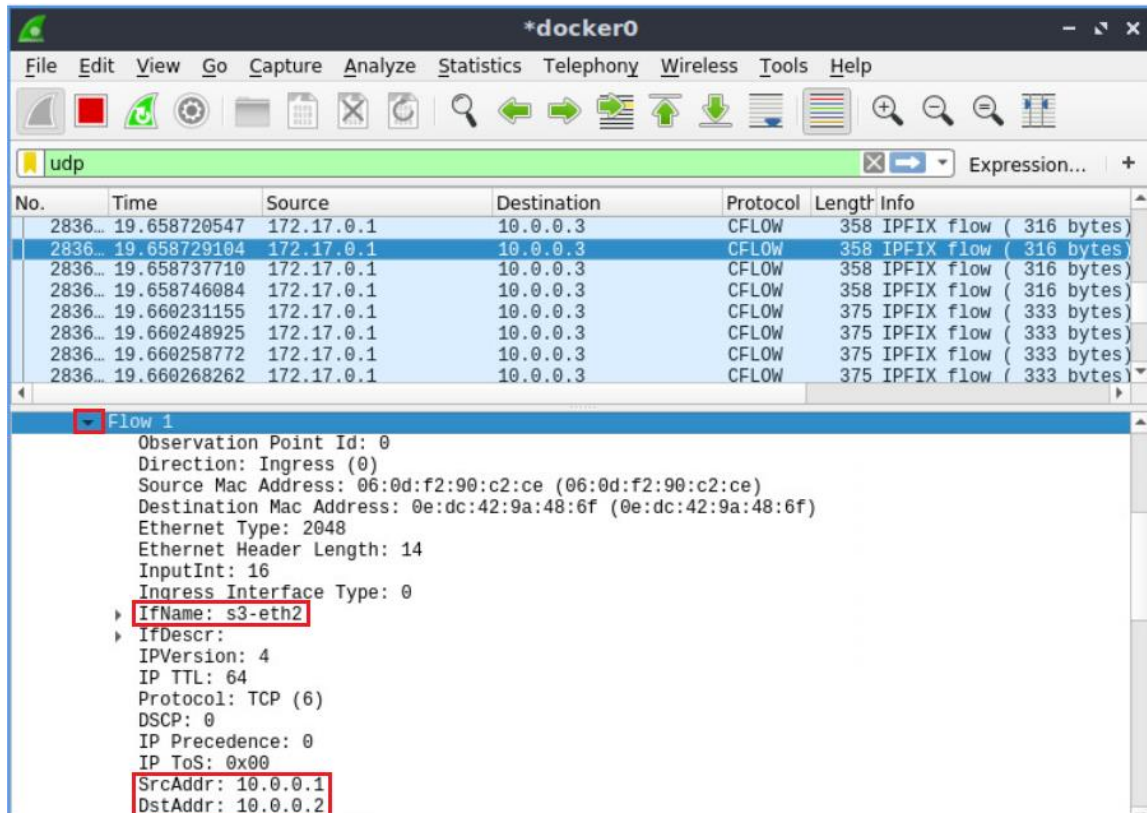


Figure 28. Verifying flow information.

Consider the figure above. The figure shows the list of IPFIX information for a single flow. For the flow, source address is 10.0.0.1, destination address is 10.0.0.2, ingress port is s3-eth2, IP version is 4, Protocol is TCP. You can scroll down to see the whole list of the flow information.

Step 9. Double click on the template frame to verify that the template fields match with the flow list.

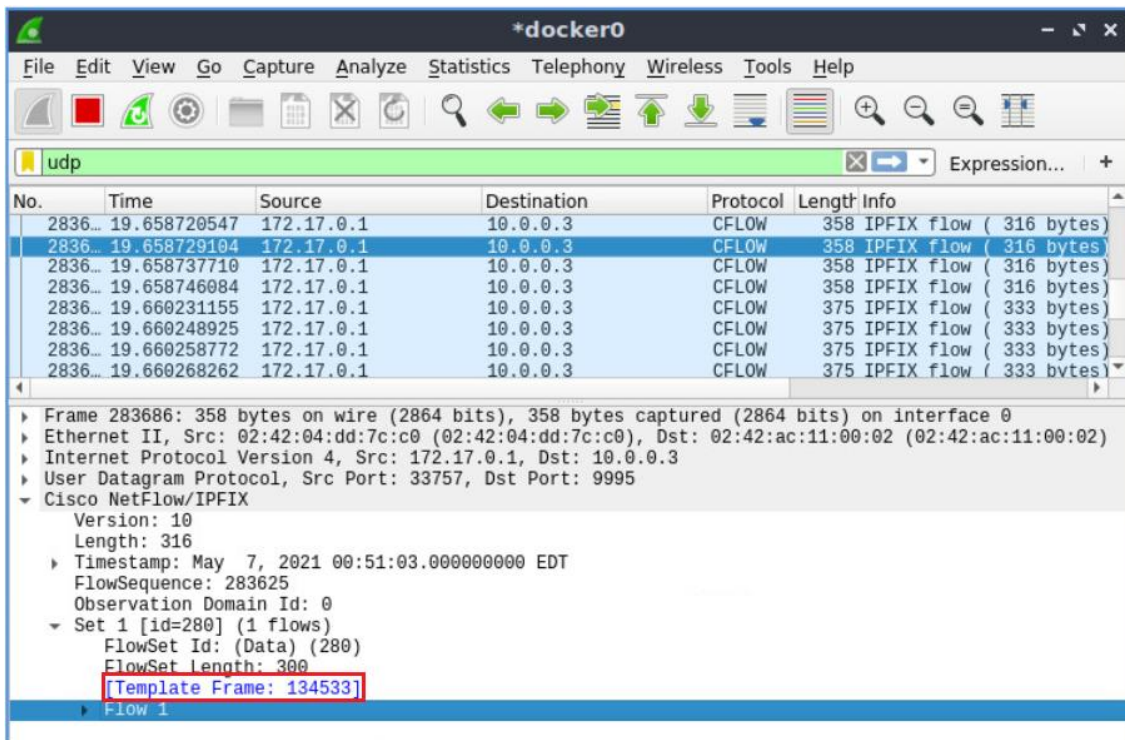


Figure 29. Navigating to the actual template frame.

Step 10. You will see the last template frame. The template ID will be the same as data ID (280).

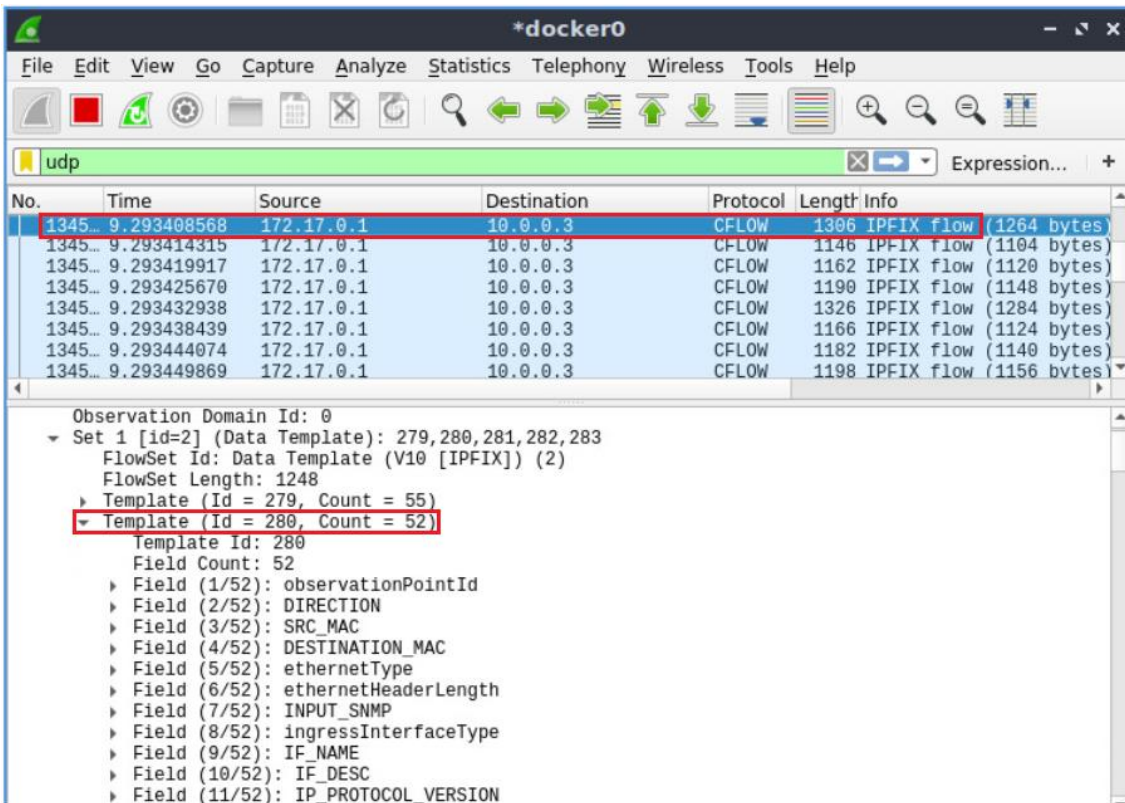


Figure 30. Verifying template ID and fields.

Consider the figure above. The fields are the same as the flow list.

This concludes Lab 3. Close the Wireshark window, stop the emulation and then exit out of MiniEdit and the Linux terminal.

References

1. Omar Santos, *“Network security with NetFlow and IPFIX”*, 2016.
2. B. Claise, Cisco Systems, B. Trammell, *“Specification of the IP Flow Information Export (IPFIX) protocol for the exchange of flow information”*, Sep 2013.
3. B. Claise, Cisco Systems, *“Cisco Systems NetFlow services export Version 9”*, Oct 2004.
4. IBM, *“IPFIX Information Elements”*, [Online]. Available: <https://www.ibm.com/docs/en/npi/1.3.1?topic=versions-ipfix-information-elements>
5. IBM, *“IPFIX”*, [Online]. Available: <https://www.ibm.com/docs/en/qradar-on-cloud?topic=sources-ipfix>
6. Flowmon, *“NetFlow / IPFIX monitoring”*, [Online]. Available: <https://www.flowmon.com/en/solutions/network-and-cloud-operations/netflow-ipfix>



UNIVERSITY OF
SOUTH CAROLINA

NETWORK MANAGEMENT

Lab 4: Introduction to sFlow

Document Version: **07-08-2022**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Introduction	3
1.1 Introduction to sFlow	4
1.2 Advantages of sFlow	5
2 Lab topology.....	5
2.1 Lab settings.....	6
2.2 Loading a topology	6
3 Launching sFlow agent.....	9
4 Analyzing sFlow sampling records using Wireshark	11
4.1 Launching Wireshark.....	11
4.2 Performing a connectivity test	12
4.3 Visualizing sFlow packets	14
References	20

Overview

This lab introduces Sampled Flow (sFlow), the leading, multi-vendor, standard for monitoring high-speed switched and routed networks. The technology is built into network equipment and gives complete visibility into network activity, enabling effective management and control of network resources¹. The focus of this lab is to explore how sFlow works in Open Virtual Switch (Open vSwitch) and analyze the collected flows using Wireshark packet analyzer.

Objectives

By the end of this lab, you should be able to:

1. Understand the concept of sFlow.
2. Enable sFlow in Open vSwitch.
3. Analyze sFlow sampling records using Wireshark.

Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Launching sFlow agent.
4. Section 4: Analyzing sFlow sampling records using Wireshark.

1 Introduction

Flow-based monitoring provides significant advantages over other network monitoring methods. NetFlow and IPFIX protocols are used for flow analysis. NetFlow collects and aggregates information about network traffic flowing through an exporter. Since sFlow stands for Sampled Flow, actual packets are being sampled here instead of flows. Netflow is restricted to IP traffic only – this is where sFlow has the greater advantage in terms of

analysis, as it can collect, monitor, and analyze traffic from OSI Layers 2, 3, 4, 5, 6 and 7². NetFlow exports datagrams that contain multiple flow records. This stateful session tracking requires memory resources. sFlow has less resource impact on devices since it only performs packet sampling and does not have to identify and keep track of sessions as is the case with NetFlow.

1.1 Introduction to sFlow

The sFlow monitoring system consists of an sFlow Agent (embedded in a switch or router) and a central sFlow Collector. The sFlow agent uses sampling technology to capture traffic statistics from the device it is monitoring. It captures packet headers and partial packet payload data into sFlow datagrams that are then exported to collectors for analysis³. Based on a defined sampling rate, an average of 1 out of N packets is randomly sampled. Since sFlow captures the entire packet headers, it is able to provide full layer 2–7 visibility into all types of traffic flowing across the network including MAC addresses, VLANs, and MPLS labels⁴.

The techniques used in the sFlow monitoring system were designed for providing continuous site-wide (and enterprise-wide) traffic monitoring of high speed switched and routed networks. It is capable of monitoring networks at 10Gbps, 100Gbps and beyond. Thousands of devices can be monitored by a single sFlow Collector. The sFlow agent is simple to implement and adds negligible cost to a switch or router⁴.

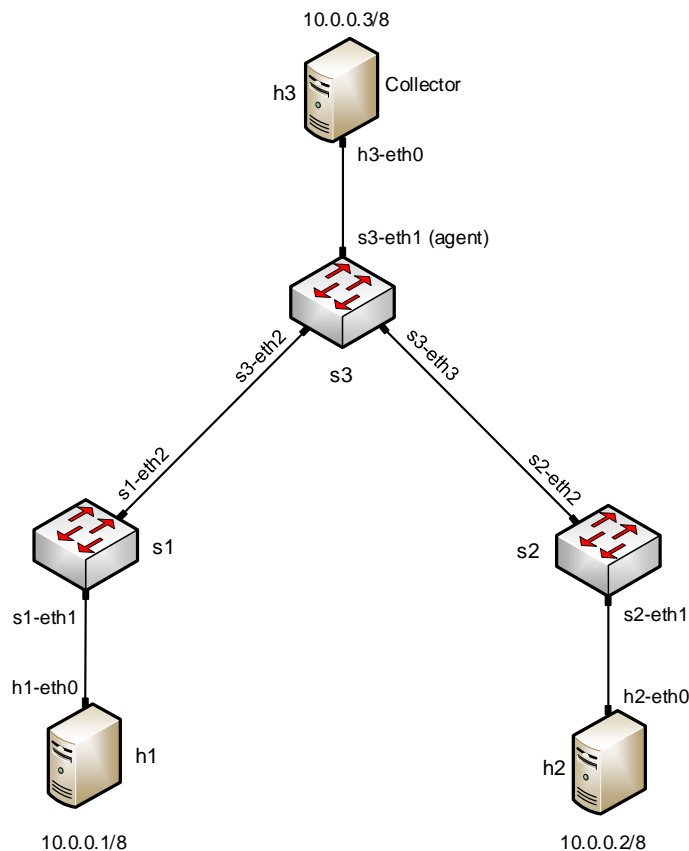


Figure 1. sFlow sample capturing.

Consider the figure above. The figure shows an sFlow agent running in switch s3, whereas host h3 is acting as an sFlow collector. The sFlow agent obtains traffic statistics from interface s3-eth1 using sampling and sends the sFlow packets to the collector. The collector analyzes these sFlow packets and displays traffic statistics in a report.

1.2 Advantages of sFlow

Troubleshooting network problems: Traffic problems are seen in abnormal traffic patterns. sFlow makes these patterns to be seen with sufficient details for quick identification, diagnosis, and correction.

Traffic jam control: Since sFlow monitors all the flows continuously on all ports, it can be used to instantly highlight the congested links, identifying the source of this traffic. It also provides the information needed to establish effective controls.

Security analysis: Assuming that security attacks and threats originate from unknown sources, an effective monitoring requires complete network surveillance with alerts for any suspicious activity. sFlow provides a comprehensive audit trail for the whole network. The constant monitoring throughout the network and route records provided by sFlow allow that threats and attacks from internal or external sources to be quickly tracked and controlled.

Accounting and billing for use: The detailed network usage is required to collect accurate values for network services and to recover costs from value-added service. The sFlow data may be used to account and charge for the use of network by clients. They can also be used to present to the client a breakdown of their total traffic, highlighting the users and applications that most consumed. This information gives the customer confidence in the accuracy of the rates and provides better cost control⁴.

2 Lab topology

Consider Figure 2. There are three switches, two end hosts and a docker container. Switch s3 is acting as sFlow exporter and the docker will collect sample flows.

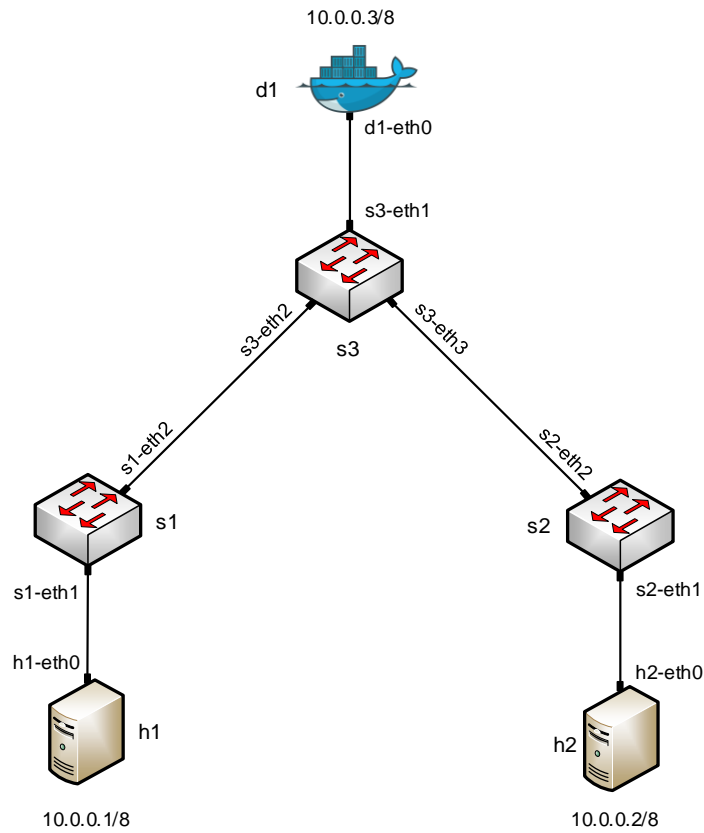


Figure 2. Lab topology.

2.1 Lab settings

The devices should be configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
d1	d1-eth0	10.0.0.3	/8

2.2 Loading a topology

Step 1. Click on the Client tab to access the Client PC.

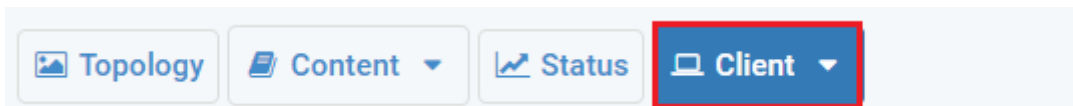


Figure 3. Accessing the Client PC.

Step 2. Start by launching MiniEdit by clicking on desktop's shortcut. When prompted for a password, type `password`.

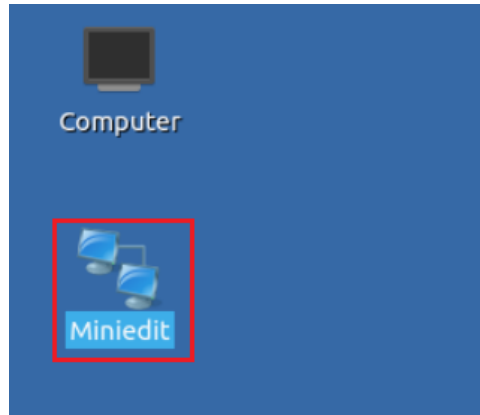


Figure 4. MiniEdit shortcut.

Step 3. On MiniEdit's menu bar, click on *File* then open to load the lab's topology. Open the directory called *lab4* and select the file *lab2.mn*. Then, click on *Open* to open the topology.

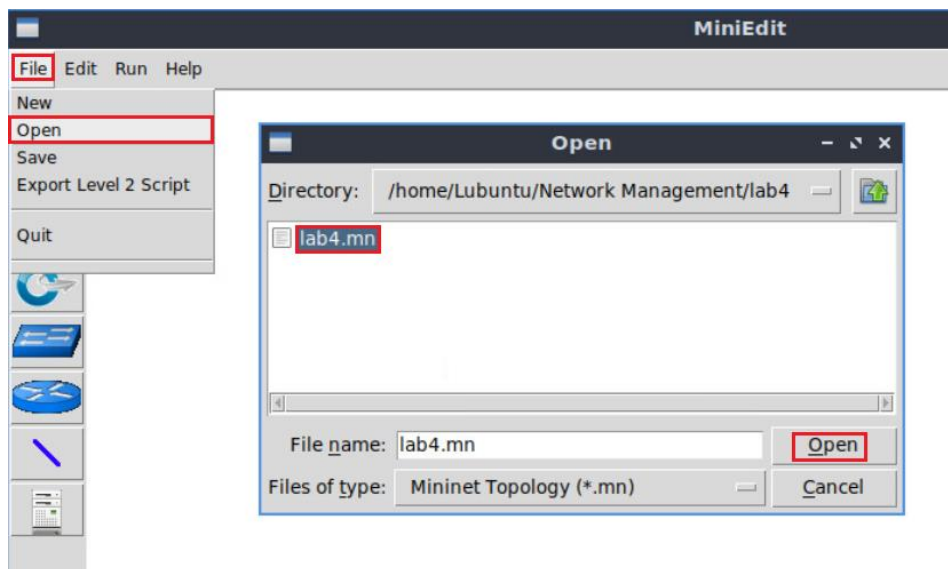


Figure 5. MiniEdit's Open dialog.

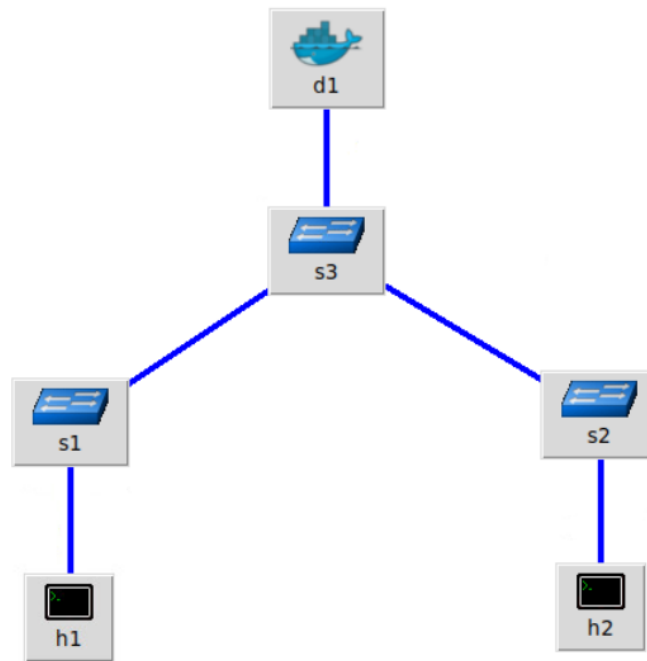


Figure 6. MiniEdit's topology.

Step 4. To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 7. Starting the emulation.

Step 5. Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

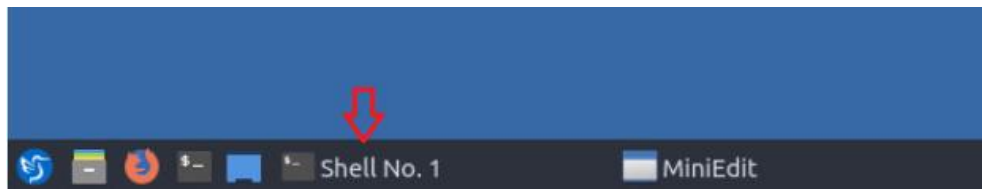


Figure 8. Opening Mininet's terminal.

Step 6. Issue the following command to display the interface names and connections.

```
links
```

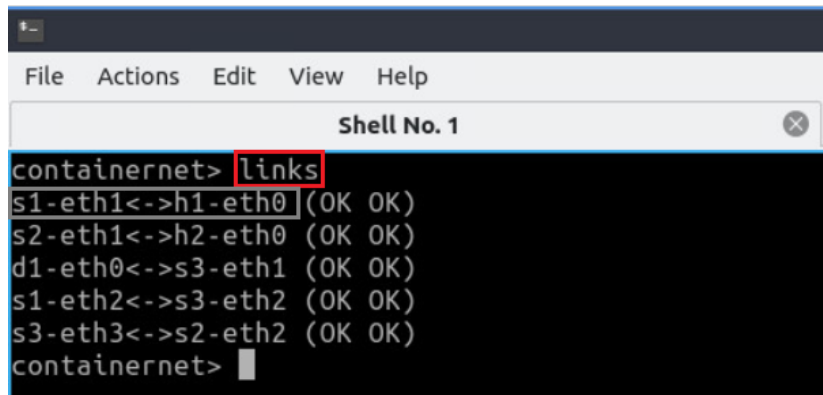


Figure 9. Displaying network interfaces.

In figure 9, the link displayed within the gray box indicates that interface eth1 of switch s1 connects to interface eth0 of host h1 (i.e., *s1-eth1<->h1-eth0*).

3 Launching sFlow agent

Step 1. Open the Linux terminal.



Figure 10. Opening Linux terminal.

Step 2. Execute the following command to start sFlow exporter. When prompted for a password, type `password`.

```
sudo ovs-vsctl -- --id=@s create sFlow agent=s3-eth1 target="\10.0.0.3:9995\" sampling=64 polling=10 -- set Bridge s3 sflow=@s
```

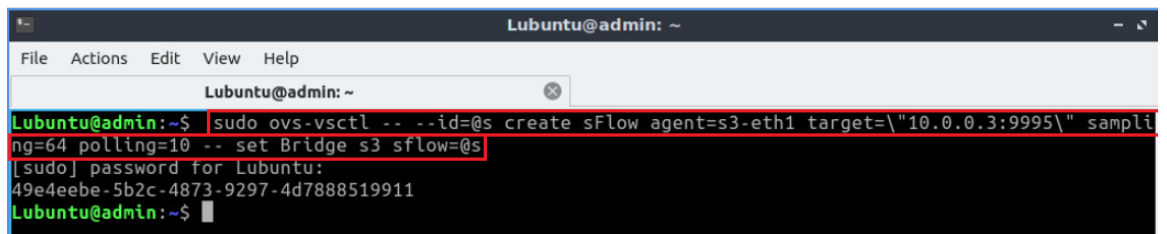


Figure 11. Enabling sFlow exporter.

Consider the figure above. The figure shows that sFlow collector in switch s3 is enabled where the target collector is docker d1 (10.0.0.3). Interface s3-eth1 is responsible for sending sFlow messages to the collector. Sampling rate is 64 which means 1 in 64 packets will be sampled. Polling interval is set to 10 which means sFlow records will be sent to the collector every 10 seconds.

Step 3. Type the following command to verify sFlow configuration.

```
sudo ovs-vsctl list bridge
```

```
Lubuntu@admin: ~$ sudo ovs-vsctl list bridge
_name
_uuid          : 3f9b2196-b970-435b-bf16-b7faef296f7
auto_attach    : []
controller     : []
datapath_id    : "0000000000000003"
datapath_type  : ""
datapath_version : "<unknown>"
external_ids   : {}
fail_mode     : standalone
flood_vlans    : []
flow_tables   : {}
ipfix         : []
mcast_snooping_enable: false
mirrors       : []
name         : s3
netflow      : []
other_config : {datapath-id="0000000000000003", disable-in-band="true", dp-desc=s3}
ports       : [3648af47-c428-4494-aa92-9b4355c896da, 540c3a06-a9c0-4d96-991e-5e6f1615af56, 56c
ff44f-8aaf-42a5-9c5f-3354fed8c27c, f69d4717-304d-474c-8b6c-0faef3aa904b]
protocols    : []
rstp_enable  : false
rstp_status  : {}
sflow       : 49e4eebe-5b2c-4873-9297-4d7888519911
status      : {}
stp_enable   : false
```

Figure 12. Verifying switch configuration.

Consider the figure above. The figure listed all the existing Open vSwitches. You will notice switch s3 has sFlow enabled with the ID (49e4eebe-5b2c-4873-9297-4d7888519911).

You might notice a different sFlow ID since it is generated randomly each time you enable sFlow agent.

Step 4. Type the following command to verify sFlow configuration.

```
sudo ovs-vsctl list sflow
```

```
Lubuntu@admin: ~$ sudo ovs-vsctl list sflow
_agent
_external_ids : {}
_header      : []
_polling     : 10
_sampling    : 64
_targets     : ["10.0.0.3:9995"]
Lubuntu@admin: ~$
```

Figure 13. Verifying sFlow configuration.

Consider the figure above. One sFlow agent is running, target collector IP is 10.0.0.3 and the port is 9995.

Step 5. Type the following command to execute a script so that the exporter can send flows to the collector.

```
sudo ./connect_collector.sh
```

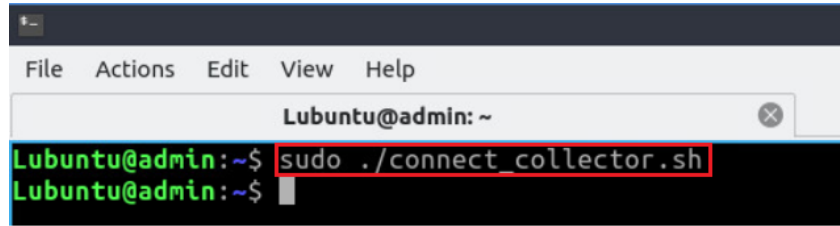


Figure 14. Connecting collector to the exporter.

The following command was executed in the script.

```
ip route add 10.0.0.0/8 via 172.17.0.1
```

4 Analyzing sFlow sampling records using Wireshark

In this section, you will analyze sFlow sampling records in Wireshark.

4.1 Launching Wireshark

Step 1. In Linux terminal, start Wireshark packet analyzer by issuing the following command. A new window will emerge.

```
sudo wireshark
```

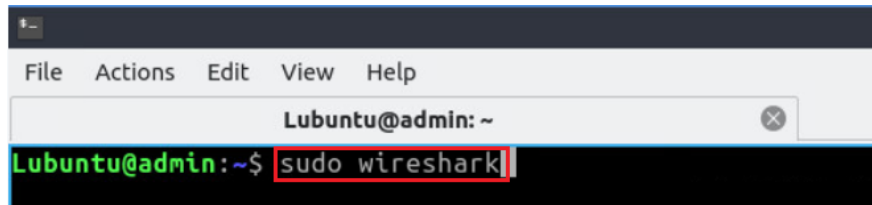


Figure 15. Starting Wireshark packet analyzer.

Step 2. Click on the icon located on the upper left-hand side to start capturing packets on docker0 interface.

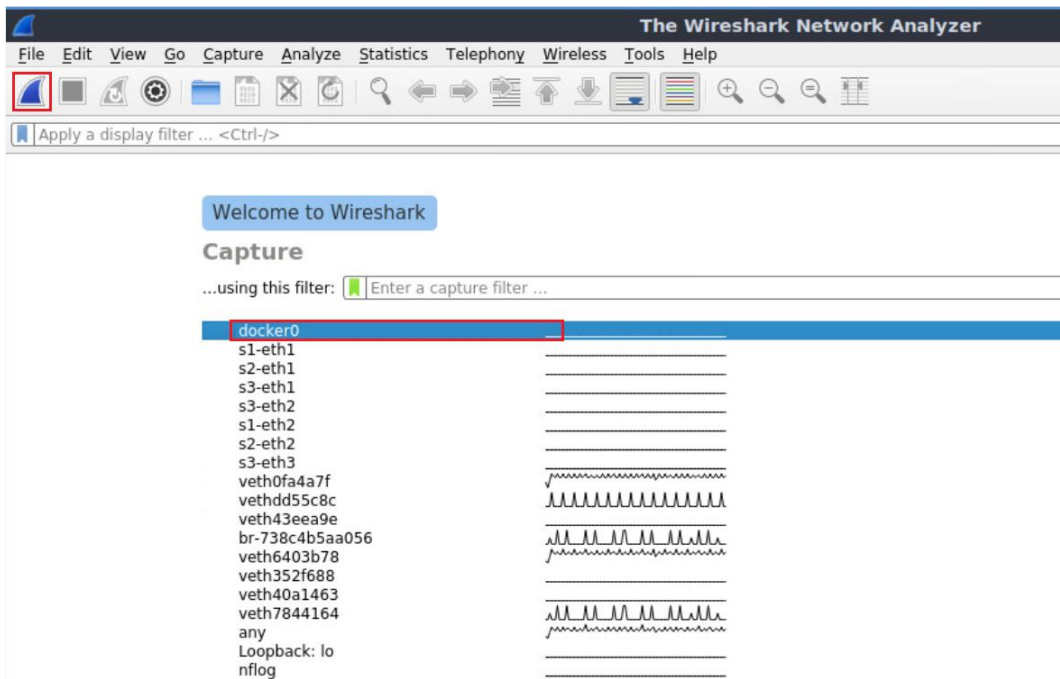


Figure 16. Starting packet capture.

Step 3. In the filter box located on the upper left-hand side, type *udp* to filter UDP packets. Press *Enter* to apply the filter.

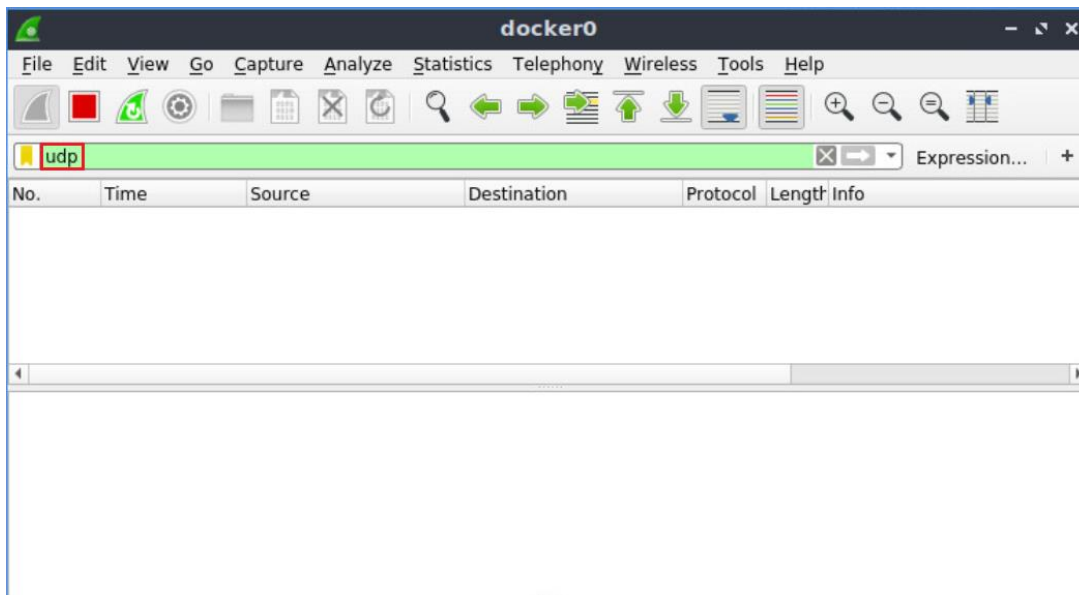


Figure 17. Filtering UDP packets.

4.2 Performing a connectivity test

Step 1. Go back to MiniEdit. Hold right-click on host h2 and select *Terminal*. This opens the terminal of host h2 and allows the execution of commands on that host.

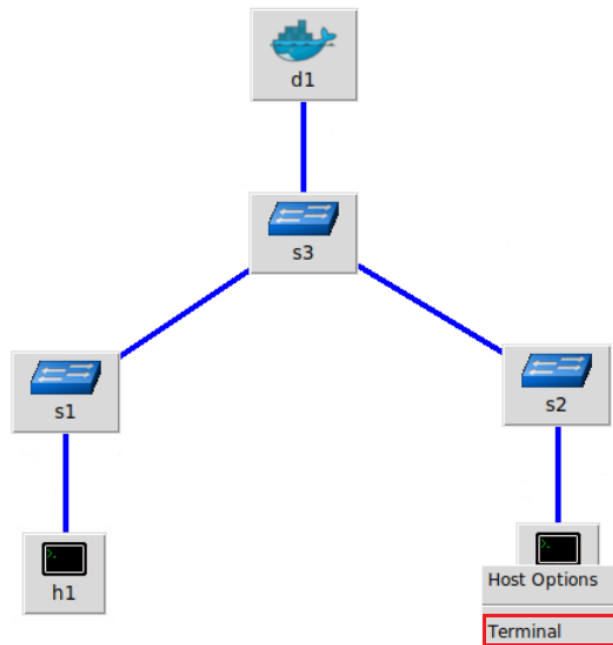


Figure 18. Opening a terminal on host h2.

Step 2. In host h2 terminal, type the following command to run the host in server mode.

```
iperf3 -s
```

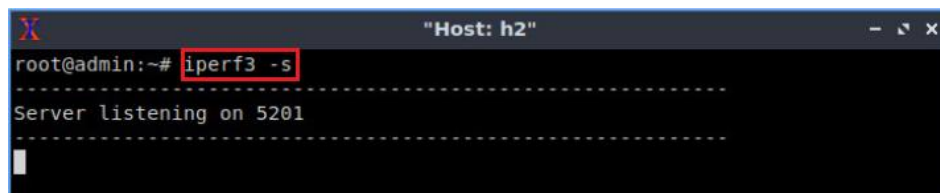


Figure 19. Running host h2 in server mode.

Consider the figure above. The figure shows that host h2 is acting as a server and listening to port 5201.

Step 3. Follow step 1 and open a terminal in host h1. Type the following command to run an iperf3 test between hosts h1 and h2, host h1 is running in client mode.

```
iperf3 -c 10.0.0.2
```



```

Host: h1
root@admin:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 7] local 10.0.0.1 port 36418 connected to 10.0.0.2 port 5201
[ ID] Interval          Transfer          Bitrate          Retr   Cwnd
[ 7]  0.00-1.00        sec  4.22 GBytes      36.3 Gbits/sec   2048   1.28 MBytes
[ 7]  1.00-2.00        sec  4.66 GBytes      40.0 Gbits/sec    740   1.17 MBytes
[ 7]  2.00-3.00        sec  3.70 GBytes      31.8 Gbits/sec     0    1.17 MBytes
[ 7]  3.00-4.00        sec  4.57 GBytes      39.2 Gbits/sec    766   1.21 MBytes
[ 7]  4.00-5.00        sec  4.41 GBytes      37.9 Gbits/sec     0    1.33 MBytes
[ 7]  5.00-6.00        sec  4.85 GBytes      41.6 Gbits/sec    90    1.39 MBytes
[ 7]  6.00-7.00        sec  4.77 GBytes      41.0 Gbits/sec   607   1.28 MBytes
[ 7]  7.00-8.00        sec  4.01 GBytes      34.5 Gbits/sec     0    1.44 MBytes
[ 7]  8.00-9.00        sec  4.21 GBytes      36.1 Gbits/sec   577   1.27 MBytes
[ 7]  9.00-10.00       sec  4.71 GBytes      40.4 Gbits/sec   611   1.38 MBytes
-----
[ ID] Interval          Transfer          Bitrate          Retr
[ 7]  0.00-10.00       sec  44.1 GBytes      37.9 Gbits/sec   5439
sender
    
```

Figure 20. Running host h1 in client mode.

Consider the figure above. The test runs for ten seconds with interval of one second.

4.3 Visualizing sFlow packets

Step 1. Stop the packet capturing by clicking the red stop button.

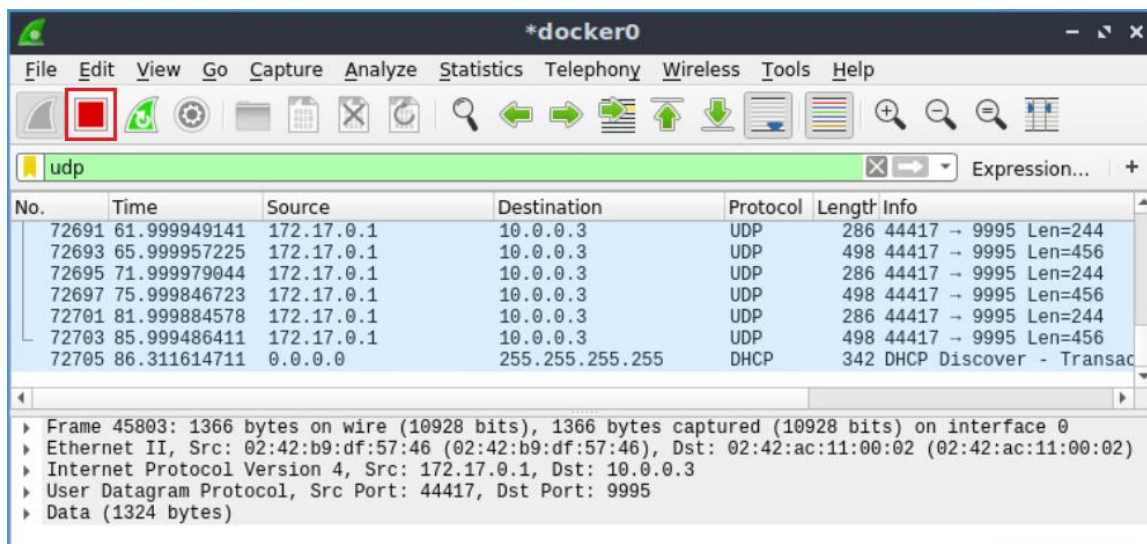


Figure 21. Stopping packet capture.

Step 2. sFlow records are exported using User Datagram Protocol (UDP). Wireshark provides a very powerful feature of decoding the captured packets into user specified formats. Right-click on any UDP packet and select decode as.

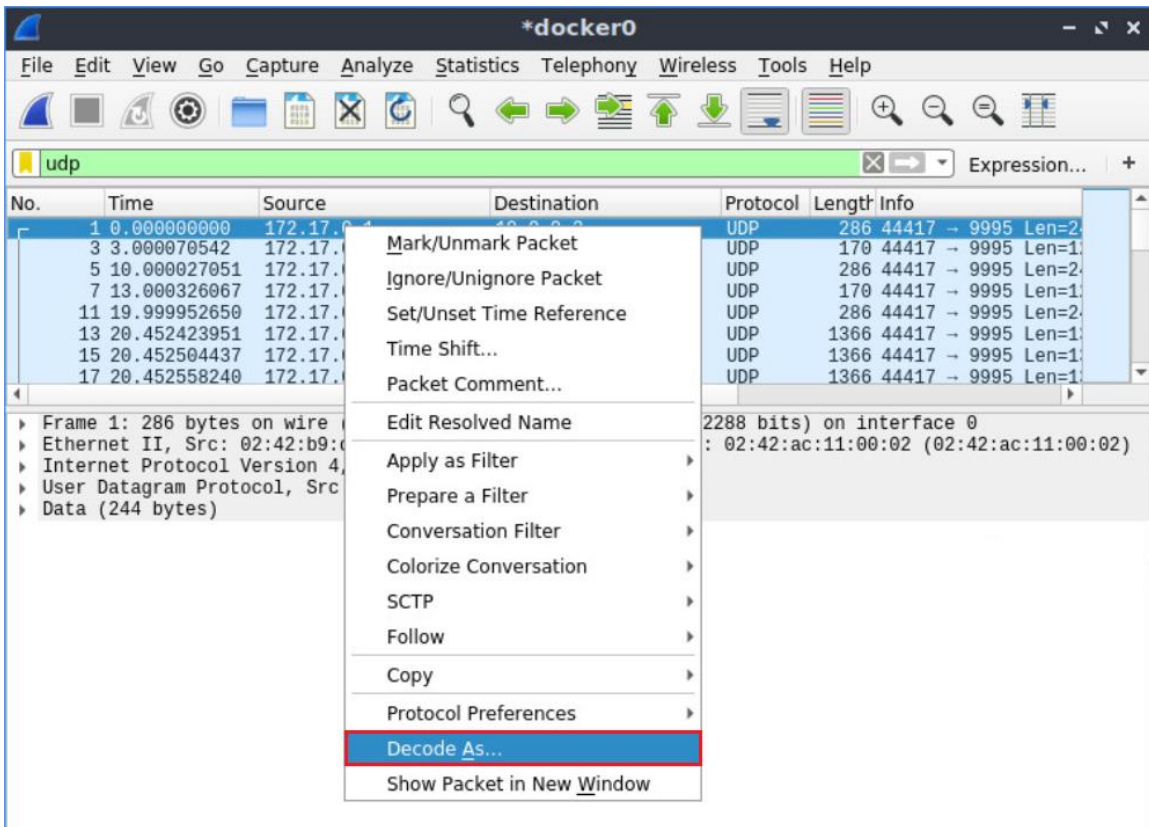


Figure 22. Decoding UDP packet.

Step 3. From the drop-down options, select *sflow* for the current field and click ok.

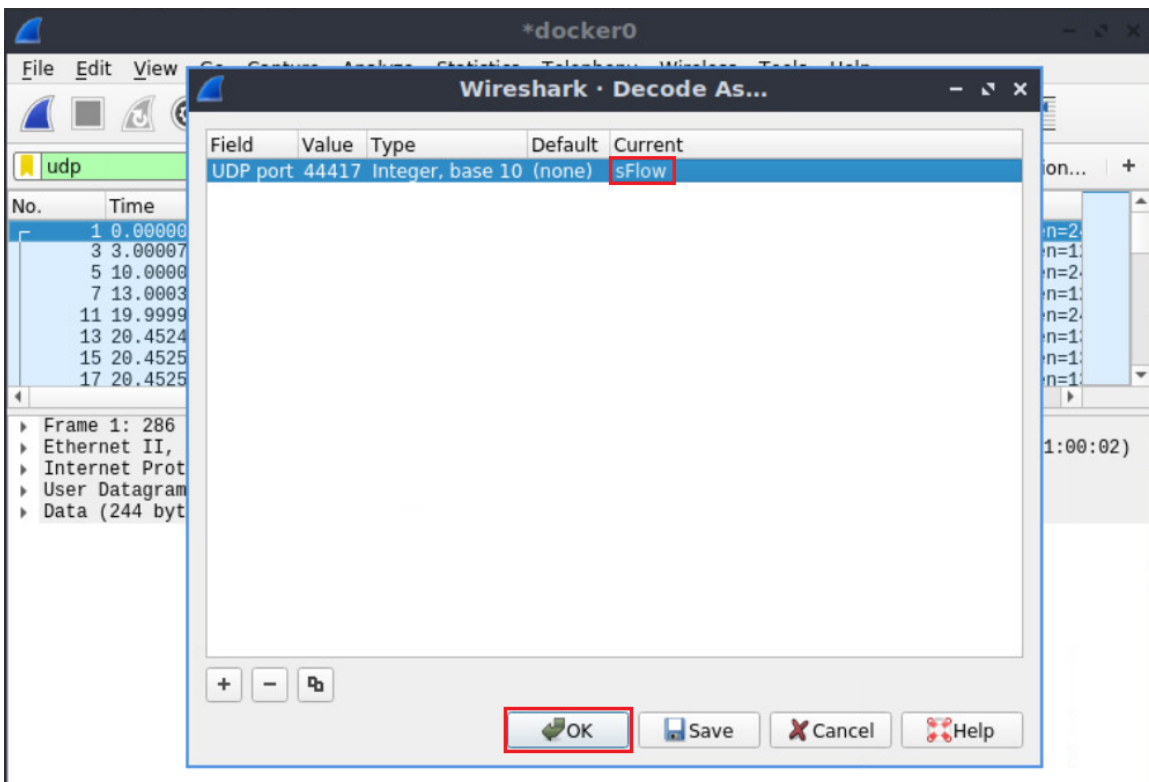


Figure 23. Decoding as sflow.

The decode functionality of Wireshark temporarily diverts the specific protocol dissections. sflow shows all the sFlow information.

Step 4. You will notice a new field called *InMon sFlow* which includes all the information regarding sFlow.

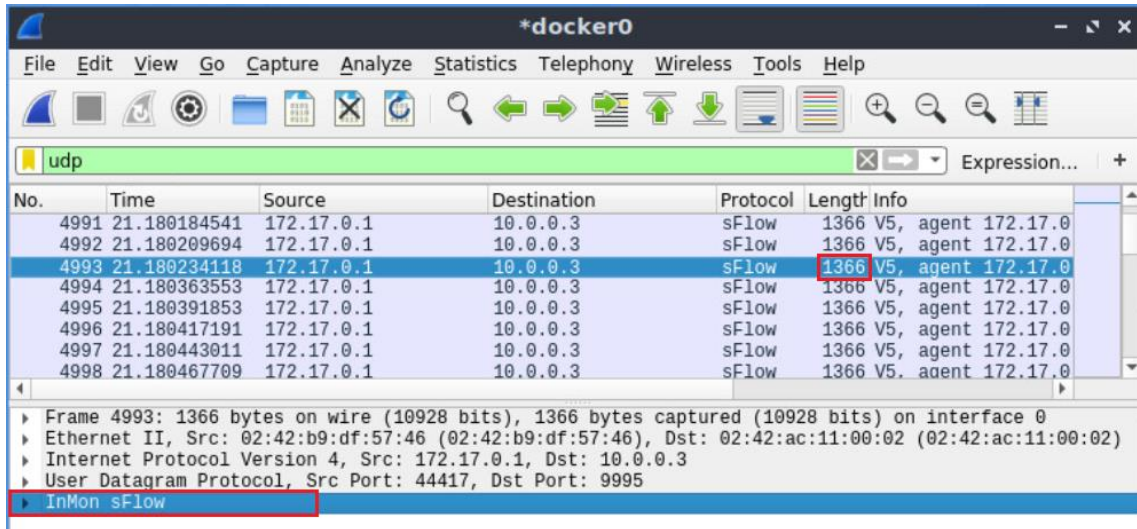


Figure 24. Verifying sFlow information.

Step 5. Click on the arrow located on the leftmost side of the field called *InMon sFlow*. A list will be displayed.

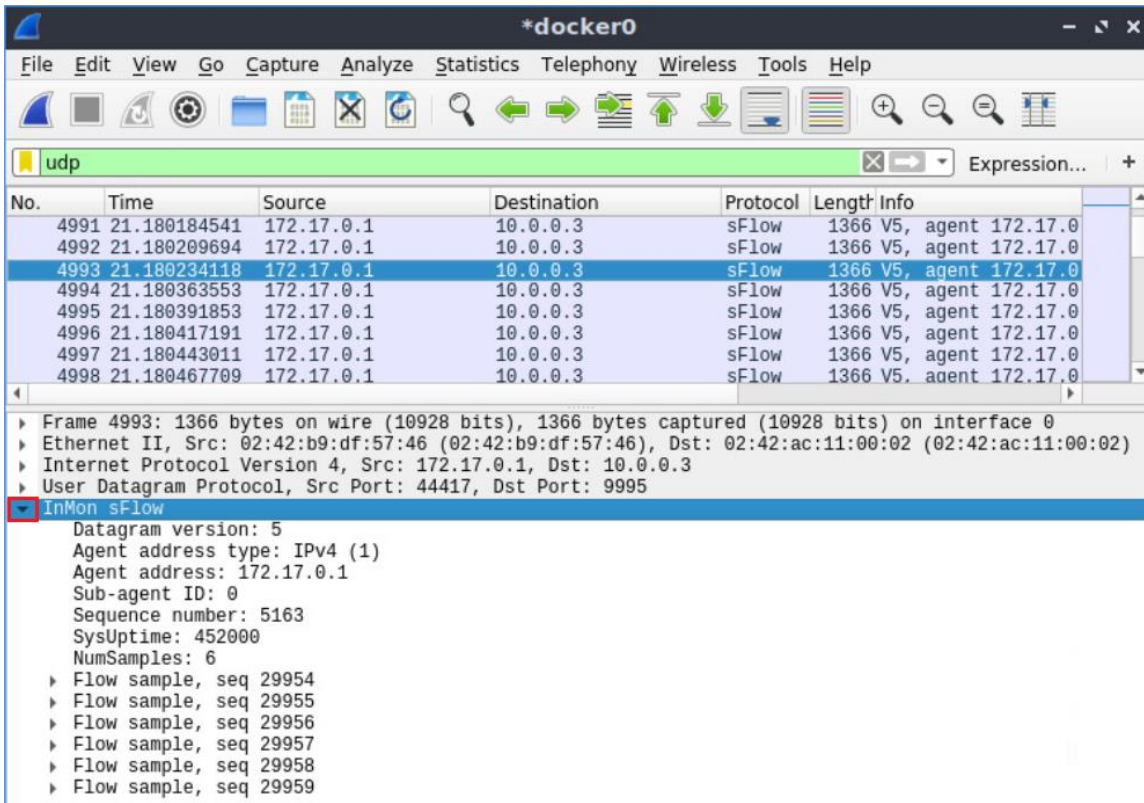


Figure 25. Verifying sFlow header information.

Consider the figure above. You will notice the information about sFlow header which includes sFlow version (Datagram version 5), Agent address 172.17.0.1 which is the switch interface responsible for sending sFlow records to the collector. sFlow v5 supports the switch/router running multiple separate sFlow agent processes. They collect, assemble, and export their sFlow information separately. If the device is running multiple agent processes, each process is given a unique ID. Sub-agent ID is set to 0 here since only a single agent is running at this time. NumSamples refers to the number of sFlow samples contained in the current packet (6). The flow samples are also listed.

Step 6. Click on the arrow located on the leftmost side of the field called *Flow sample*. A list will be displayed.

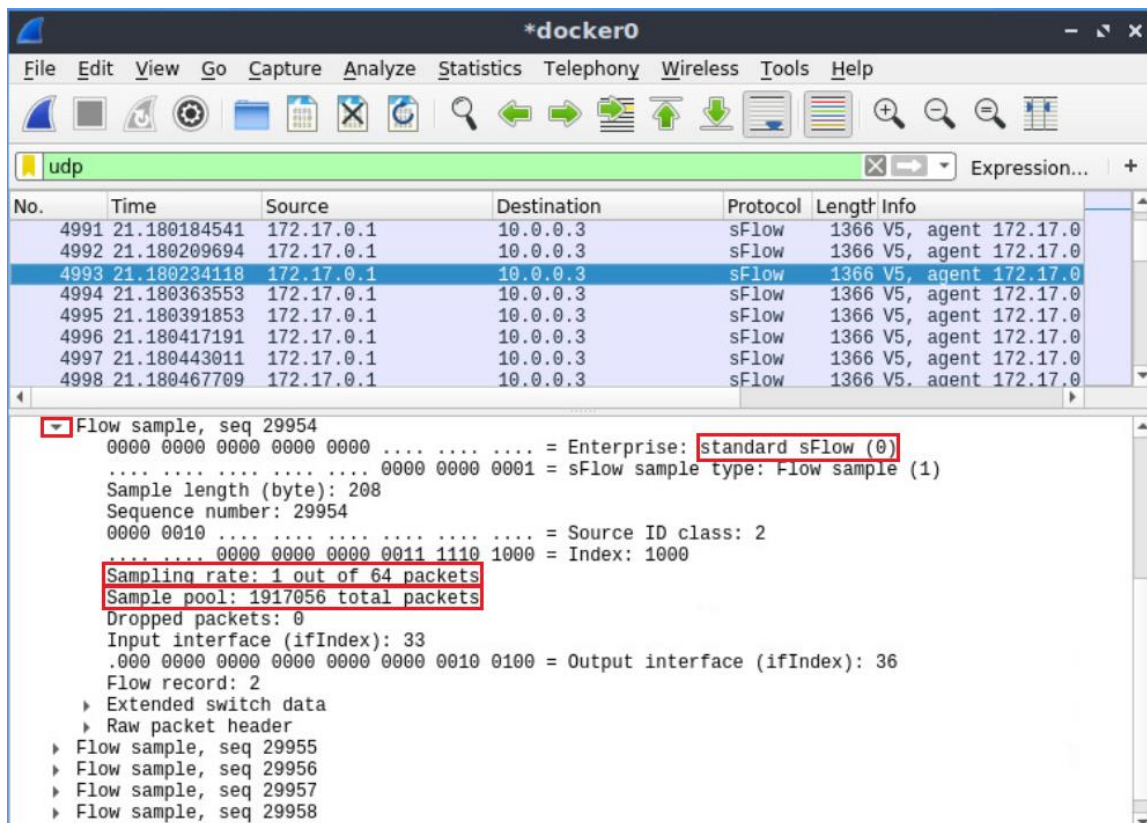


Figure 26. Verifying flow sample record.

Consider the figure above. The Enterprise types allow individual enterprises to define their sample types. This is 0 by default which refers to the standard sFlow type. The sampling rate is 1 out of 64 which means 1 in 64 packets is sampled. Sample pool refers to the total number of packets that could have been sampled.

Step 7. Click on the arrow located on the leftmost side of the field called *Extended switch data*. A list will be displayed.

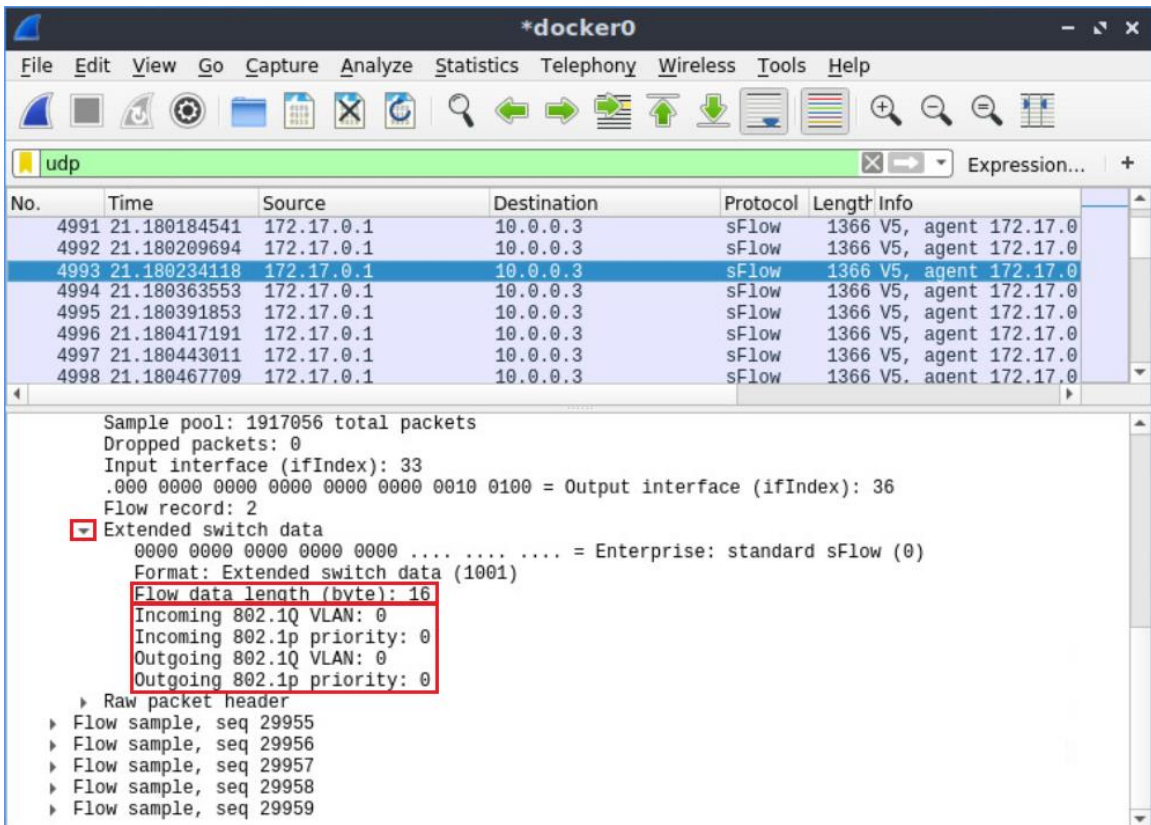


Figure 27. Verifying flow sample record.

Consider the figure above. It shows the extended switch information such as VLAN information.

Step 8. Click on the arrow located on the leftmost side of the field called *Raw packet header*. A list will be displayed.

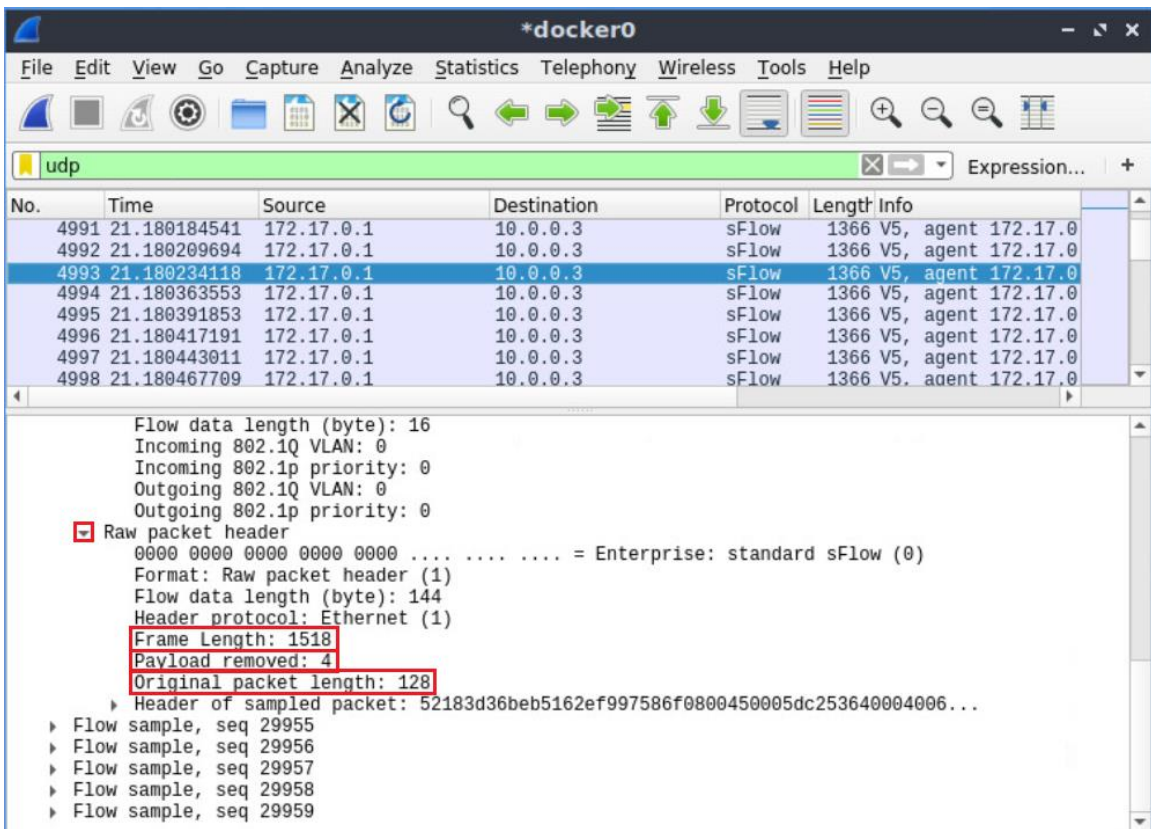


Figure 28. Verifying flow sample record.

Consider the figure above. It shows the information about raw packet header. Header protocol is ethernet, length of frame before sampling is 1518, Payload removed value is 4 which means 4 bytes has been removed by the sampling process. Original packet length is 128 bytes.

Step 9. Click on the arrow located on the leftmost side of the field called *Header of sampled packet*. A list will be displayed.

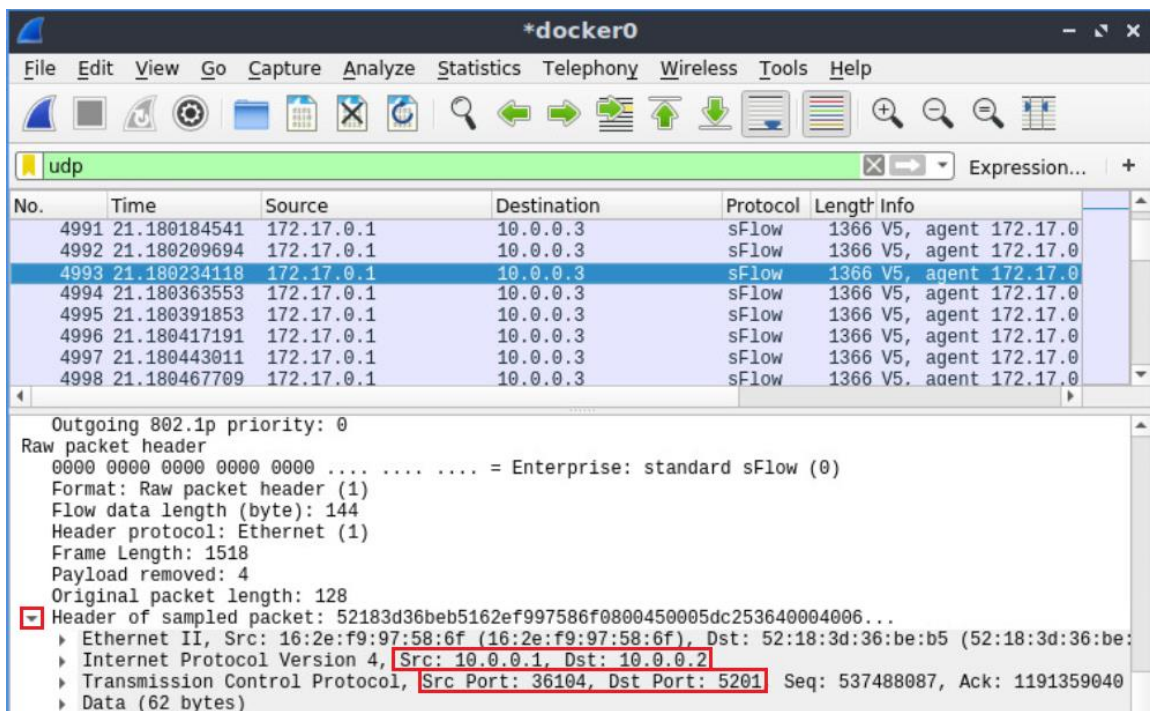


Figure 29. Verifying flow sample record.

Consider the figure above. It shows the source and destination MAC addresses, source IP (10.0.0.1) and destination IP (10.0.0.2), TCP information such as source and destination ports.

This concludes Lab 4. Close Wireshark window, stop the emulation and then exit out of MiniEdit and the Linux terminal.

References

1. InMon, "sFlow", [Online]. Available: <https://inmon.com/technology/>
2. Thwack, "NetFlow vs. sFlow – Differences and Applications!", [Online]. Available: <https://thwack.solarwinds.com/resources/b/geek-speak/posts/netflow-vs-sflow-differences-and-applications>
3. P. Phaal, S. Panchen, N. Mckee, InMon Corp, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks", [Online]. Available: <https://datatracker.ietf.org/doc/html/rfc3176>
4. sFlow, "Traffic monitoring using sFlow", [Online]. Available: <https://sflow.org/sFlowOverview.pdf>



UNIVERSITY OF
SOUTH CAROLINA

NETWORK MANAGEMENT

Lab 5: Collecting and Processing NetFlow, IPFIX and sFlow data using Nfdump

Document Version: **07-08-2022**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Introduction	3
1.1 Introduction to Nfdump.....	4
1.2 Processing data using Nfdump.....	4
2 Lab topology.....	5
2.1 Lab settings.....	5
2.2 Loading a topology	6
3 Launching NetFlow collector and exporter	8
4 Analyzing NetFlow data using Nfdump.....	10
4.1 Performing a connectivity test.....	11
4.2 Visualizing NetFlow data stored by nfcapd.....	12
5 Analyzing IPFIX data using Nfdump	14
5.1 Launching IPFIX collector and exporter	15
5.2 Visualizing IPFIX data stored by nfcapd	17
6 Analyzing sFlow data using Nfdump.....	19
6.1 Launching sFlow collector and agent.....	19
6.2 Visualizing sFlow data stored by sfcapd.....	21
References	23

Overview

This lab introduces Nfdump, a set of tools to collect and process NetFlow, IPFIX and sFlow data. The focus of this lab is to enable NetFlow, IPFIX, sFlow exporters in Open Virtual Switch (Open vSwitch) and analyze data using Nfdump.

Objectives

By the end of this lab, you should be able to:

1. Understand the concept of Nfdump.
2. Enable NetFlow, IPFIX and sFlow in Open vSwitch.
3. Analyze collected data using Nfdump.

Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Launching NetFlow collector and exporter.
4. Section 4: Analyzing NetFlow data using Nfdump.
5. Section 5: Analyzing IPFIX data using Nfdump.
6. Section 6: Analyzing sFlow data using Nfdump.

1 Introduction

NetFlow and IPFIX are designed for the collection of IP traffic information and the monitoring of network traffic. They provide a detailed view of application flows and a popular way of gaining both a broad and detailed picture of what is happening inside the network. With the help of these tools, it is possible to visualize where network traffic ends up, the source of the traffic, and how much traffic is being generated. You can gain

increased visibility into bandwidth allocation, identify, and prevent any further abuse potentially impacting the performance of the network¹.

sFlow uses sampling technology to capture traffic statistics from the device it is monitoring. It captures packet headers and partial packet payload data into sFlow datagrams that are then exported to collectors for analysis². Based on a defined sampling rate, an average of 1 out of N packets is randomly sampled.

1.1 Introduction to Nfdump

Nfdump is a very popular command-line toolset for collecting, storing, and processing NetFlow/SFlow/IPFIX data. This utility can process data very fast. Nfcapd is the NetFlow and IPFIX capture daemon that reads the data from the network and stores the data into files. Sfcapd is the sFlow capture daemon. Nfdump reads the data from the files stored by the daemons. All the data is stored as nfcapd file. Every five minutes, nfcapd rotates and renames the output file with the time stamp nfcapd.YYMMddhhmm of the interval (e.g., nfcapd.202103221140 contains data from March 22nd, 2021, time 11:40 onward). It can save 288 files per day based on five minutes time interval. Nfdump also offers a graphical web-based front-end tool called nfsen-ng which allows processing the NetFlow data within the specified time span, easily navigate through the collected data, set alerts, based on various conditions. It can process a file stored by nfcapd every five minutes³.

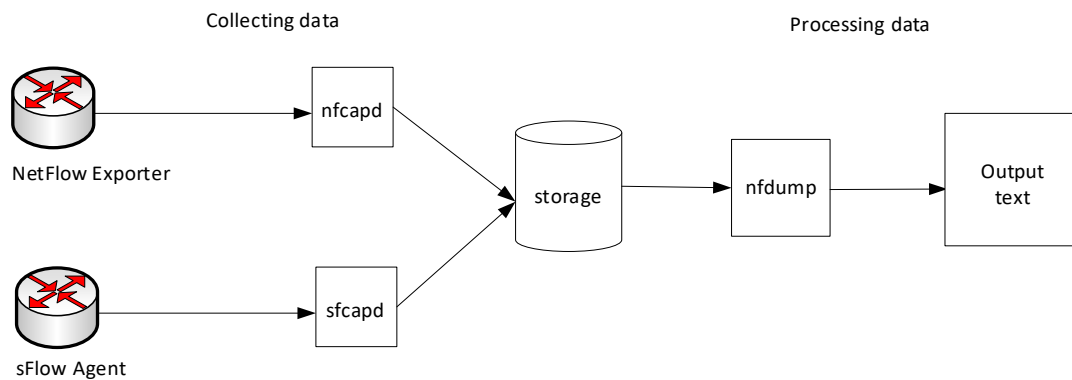


Figure 1. Nfdump architecture.

Consider Figure 1. NetFlow and sFlow daemons are capturing and saving data from routers. Nfdump reads the stored data from the files and delivers as text based on the queries.

1.2 Processing data using Nfdump

Flows can be read either from a single file or from a sequence of files. Nfdump has several output formats (line, long, extended). The output format line is the default output format when no format is specified. It limits the information to the connection details as well as number of packets, bytes, and flows. The output format long is identical to the format line and includes additional information such as TCP flags and Type of Service (ToS). The output format extended is identical to the format long and includes additional computed

information such as packet per second (pps), bytes per second (bps) and bits per packet (bpp)⁴.

Nfdump has a powerful and fast filter engine. All flows are filtered before they are further processed. If no filter is given, any flow will be processed. The filter is either given on the command line as last argument enclosed in ', or in a file³.

2 Lab topology

Consider Figure 2. There are three switches, two end hosts, and a docker container. Switch s3 is acting as an exporter and the docker d1 is the collector.

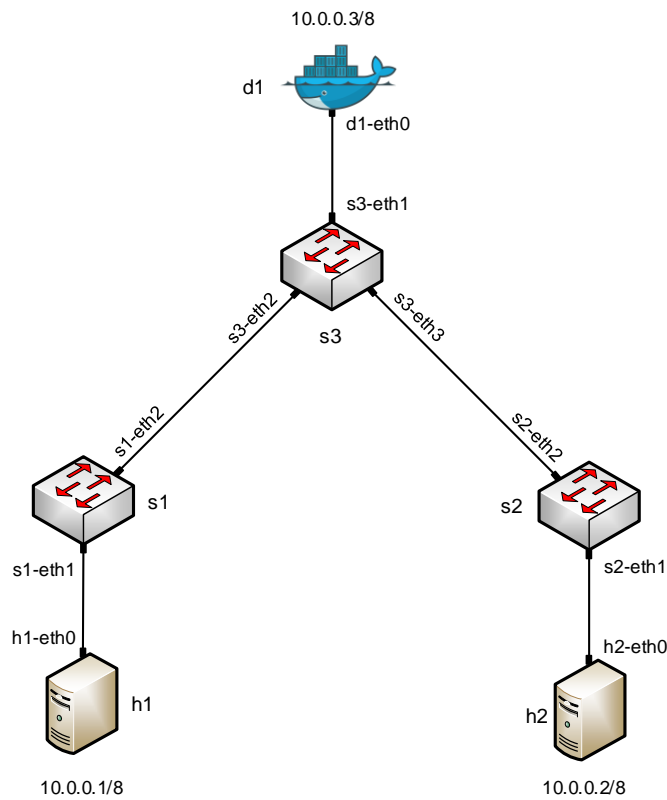


Figure 2. Lab topology.

2.1 Lab settings

The devices should be configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
d1	d1-eth0	10.0.0.3	/8

2.2 Loading a topology

Step 1. Click on the Client tab to access the Client PC.

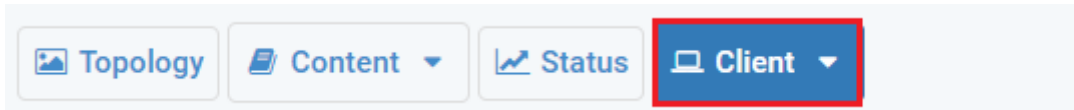


Figure 3. Accessing the Client PC.

Step 2. Start by launching MiniEdit by clicking on desktop's shortcut. When prompted for a password, type `password`.



Figure 4. MiniEdit shortcut.

Step 3. On MiniEdit's menu bar, click on *File* then open to load the lab's topology. Open the directory called *lab5* and select the file *lab5.mn*. Then, click on *Open* to open the topology.

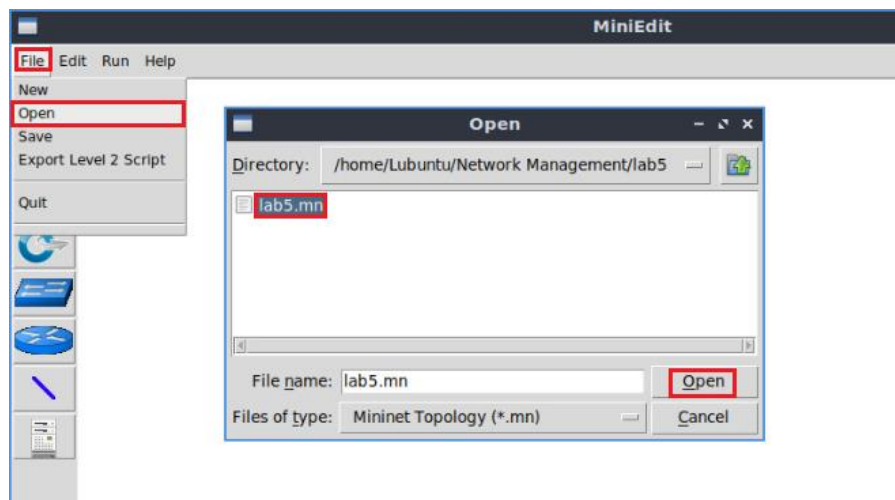


Figure 5. MiniEdit's Open dialog.

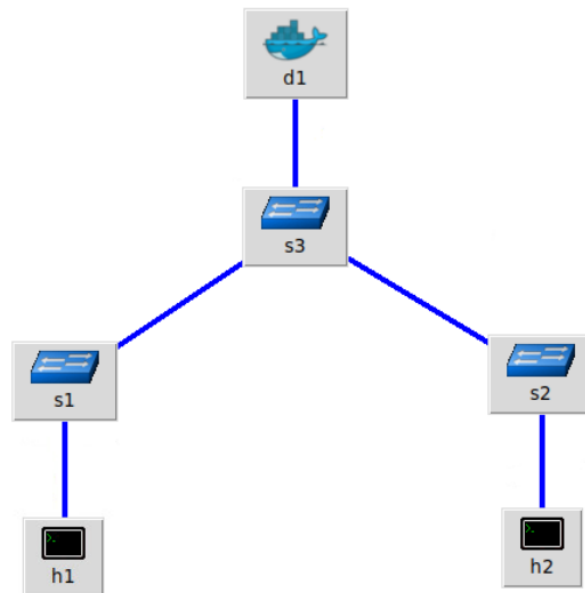


Figure 6. MiniEdit's topology.

Step 4. To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 7. Starting the emulation.

Step 5. Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

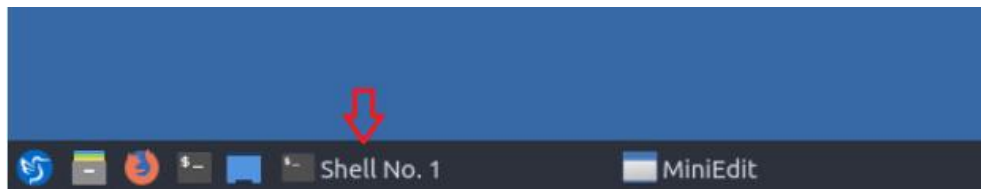
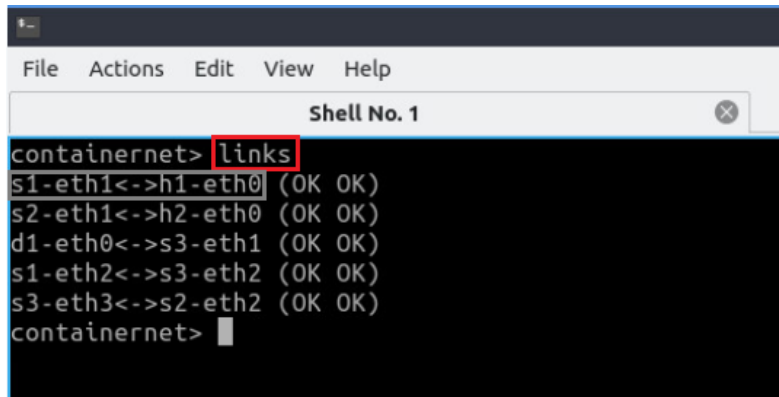


Figure 8. Opening Mininet's terminal.

Step 6. Issue the following command to display the interface names and connections.

```
links
```



```
File Actions Edit View Help
Shell No. 1
containernet> links
s1-eth1<->h1-eth0 (OK OK)
s2-eth1<->h2-eth0 (OK OK)
d1-eth0<->s3-eth1 (OK OK)
s1-eth2<->s3-eth2 (OK OK)
s3-eth3<->s2-eth2 (OK OK)
containernet> █
```

Figure 9. Displaying network interfaces.

In figure 9, the link displayed within the gray box indicates that interface eth0 of host h1 connects to interface eth1 of switch s1 (i.e., *s1-eth1<->h1-eth0*).

3 Launching NetFlow collector and exporter

Step 1. Go back to MiniEdit. Hold right-click on docker d1 and select *Terminal*. This opens the terminal of the docker and allows the execution of commands.

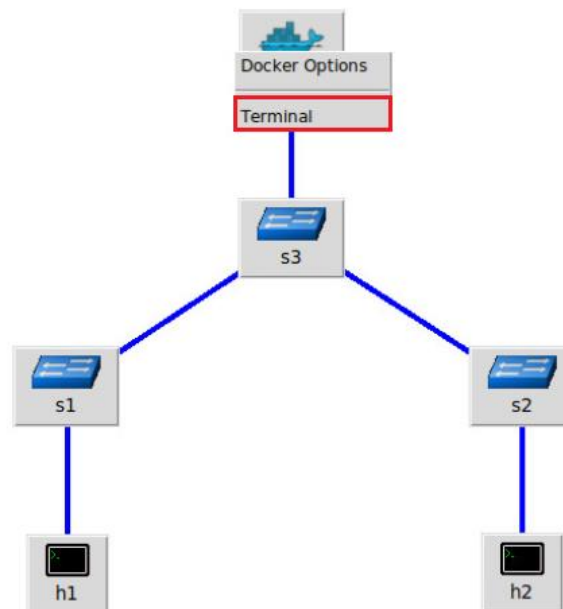
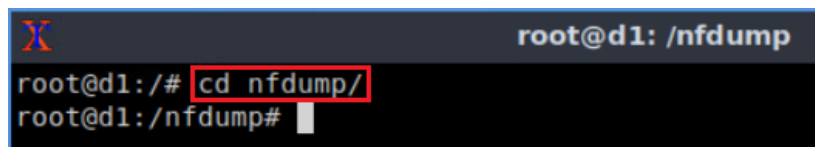


Figure 10. Opening a terminal on docker d1.

Step 2. Navigate into */nfdump* directory by issuing the following command.

```
cd nfdump/
```



```
root@d1: /nfdump
root@d1:/# cd nfdump/
root@d1:/nfdump# █
```

Figure 11. Entering into nfdump directory.

Step 3. Type the following command to start the nfcapd daemon. The daemon will start collecting data and stores in file *netflow_files*.

```
nfcapd -b 10.0.0.3 -p 9995 -l /nfdump/netflow_files/
```

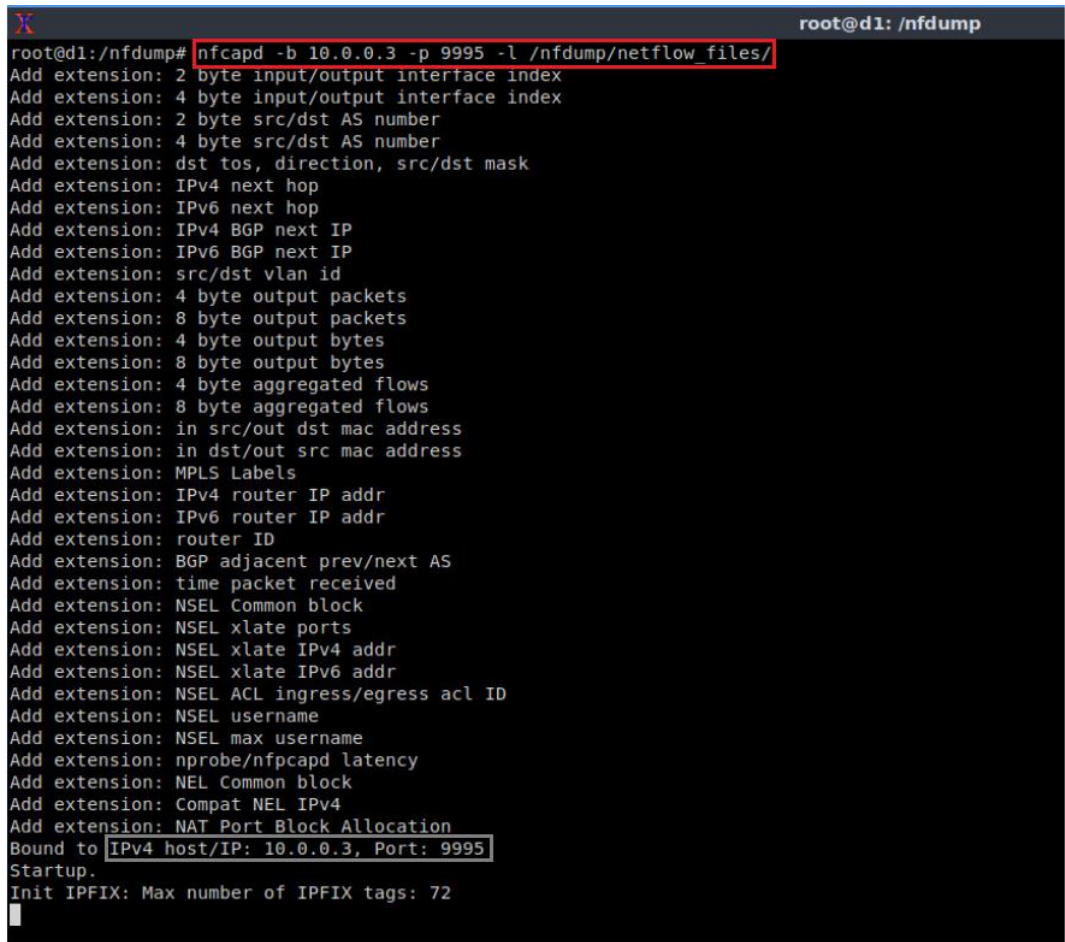


Figure 12. Starting nfcapd daemon.

Consider the figure above. `-b` specifies the host address 10.0.0.3 to bind for listening. `-p` specifies the port number 9995. `-l` is referring to the directory where the file will be saved. The figure shows that the host was bound to the daemon.

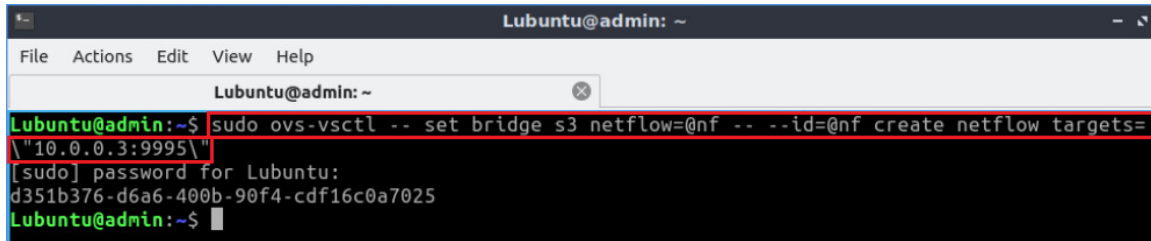
Step 4. Open the Linux terminal.



Figure 13. Opening Linux terminal.

Step 5. Execute the following script to start the NetFlow exporter. If prompted for password, type `password`.

```
sudo ovs-vsctl -- set bridge s3 netflow=@nf -- --id=@nf create netflow targets="10.0.0.3:9995"
```

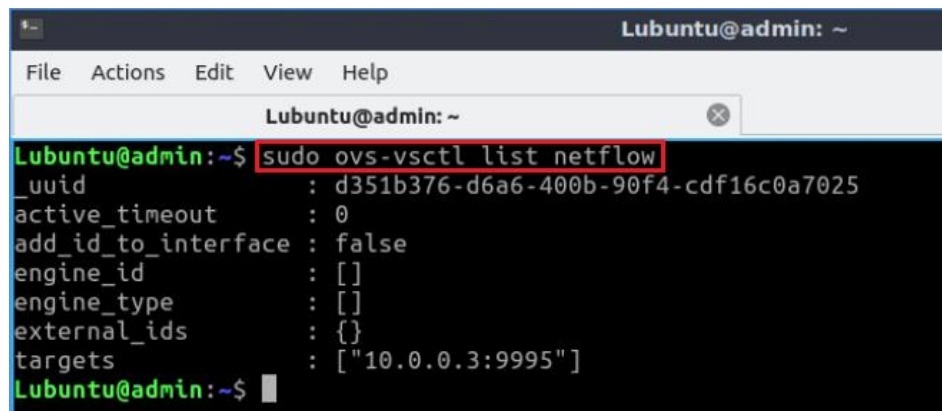
```
Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
Lubuntu@admin:~$ sudo ovs-vsctl -- set bridge s3 netflow=@nf -- --id=@nf create netflow targets=
{"10.0.0.3:9995"}
[sudo] password for Lubuntu:
d351b376-d6a6-400b-90f4-cdf16c0a7025
Lubuntu@admin:~$
```

Figure 14. Starting NetFlow exporter.

Consider the figure above. The command creates a NetFlow ID and is attached to switch s3. Switch s3 is acting as an exporter and transmits data to the collector. 10.0.0.3 is the collector IP and the port is the default UDP port 9995.

Step 6. Type the following command to verify NetFlow configuration.

```
sudo ovs-vsctl list netflow
```



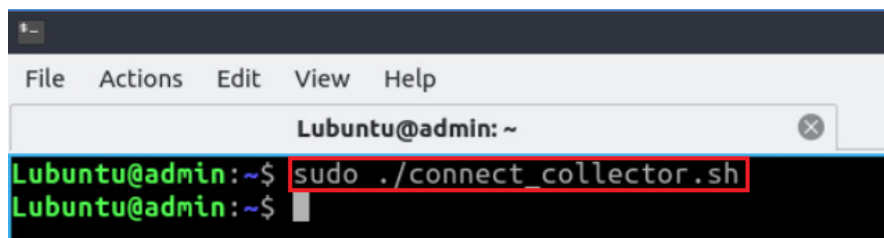
```
Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
Lubuntu@admin:~$ sudo ovs-vsctl list netflow
_uuid          : d351b376-d6a6-400b-90f4-cdf16c0a7025
active_timeout : 0
add_id_to_interface : false
engine_id      : []
engine_type    : []
external_ids   : {}
targets        : ["10.0.0.3:9995"]
Lubuntu@admin:~$
```

Figure 15. Verifying NetFlow configuration.

Consider the figure above. Exporter s3 is running with a Universally unique identifier (UUID) which is also known as NetFlow ID, target IP address is 10.0.0.3 and the port is 9995.

Step 7. Type the following command to execute a script so that the exporter can send flows to the collector. If prompted for password, type `password`.

```
sudo ./connect_collector.sh
```



```
Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
Lubuntu@admin:~$ sudo ./connect_collector.sh
Lubuntu@admin:~$
```

Figure 16. Connecting collector to the exporter.

4 Analyzing NetFlow data using Nfdump

In this section, you will analyze NetFlow data.

4.1 Performing a connectivity test

Step 1. Hold right-click on host h2 and select *Terminal*. This opens the terminal of host h2 and allows the execution of commands on that host.

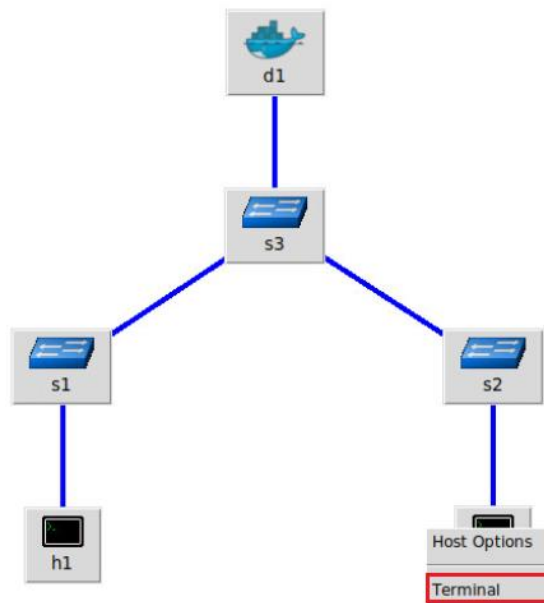


Figure 17. Opening a terminal on host h2.

Step 2. In host h2 terminal, type the following command to run the host in server mode.

```
iperf3 -s
```

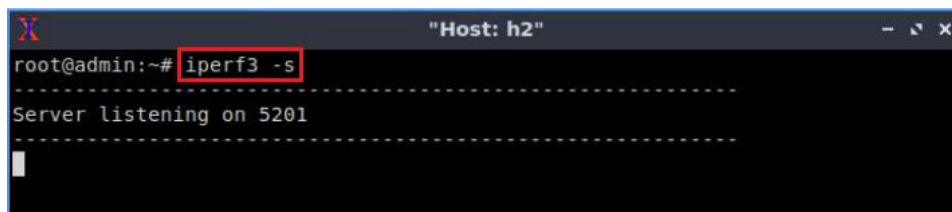


Figure 18. Running host h2 in server mode.

Consider the figure above. The figure shows that host h2 is acting as a server and listening to port 5201.

Step 3. Open host h1 terminal and type the following command to run the host in client mode.

```
iperf3 -c 10.0.0.2
```

```

X "Host: h1"
root@admin:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 7] local 10.0.0.1 port 41764 connected to 10.0.0.2 port 5201
[ ID] Interval      Transfer      Bitrate      Retr  Cwnd
[ 7]  0.00-1.00    sec  4.68 GBytes  40.2 Gbits/sec    0   1.13 MBytes
[ 7]  1.00-2.00    sec  4.80 GBytes  41.3 Gbits/sec    0   2.58 MBytes
[ 7]  2.00-3.00    sec  4.87 GBytes  41.8 Gbits/sec  202   2.19 MBytes
[ 7]  3.00-4.00    sec  4.80 GBytes  41.2 Gbits/sec    0   2.85 MBytes
[ 7]  4.00-5.00    sec  4.81 GBytes  41.3 Gbits/sec  108   2.00 MBytes
[ 7]  5.00-6.00    sec  4.84 GBytes  41.5 Gbits/sec    0   2.00 MBytes
[ 7]  6.00-7.00    sec  4.89 GBytes  42.0 Gbits/sec  235   1.25 MBytes
[ 7]  7.00-8.00    sec  4.82 GBytes  41.4 Gbits/sec    0   1.25 MBytes
[ 7]  8.00-9.00    sec  4.80 GBytes  41.3 Gbits/sec    0   1.25 MBytes
[ 7]  9.00-10.00   sec  4.82 GBytes  41.4 Gbits/sec    0   1.34 MBytes
-----
[ ID] Interval      Transfer      Bitrate      Retr
[ 7]  0.00-10.00   sec  48.1 GBytes  41.3 Gbits/sec  545
[ 7]  0.00-10.04   sec  48.1 GBytes  41.2 Gbits/sec
sender
receiver
iperf Done.
root@admin:~#

```

Figure 19. Running host h1 in client mode.

Consider the figure above. The test runs for ten seconds with an interval of one second.

4.2 Visualizing NetFlow data stored by nfcapd

Step 1. Go to docker d1 terminal to verify that the exporter was detected to examine the packets. Press `Ctrl+c` to stop the nfcapd daemon.

```

root@d1:/nfdump# nfcapd -b 10.0.0.3 -p 9995 -l /nfdump/netflow_files/
Add extension: 2 byte input/output interface index
Add extension: 4 byte input/output interface index
Add extension: 2 byte src/dst AS number
Add extension: 4 byte src/dst AS number
Add extension: dst tos, direction, src/dst mask
Add extension: IPv4 next hop
Add extension: IPv6 next hop
Add extension: IPv4 BGP next IP
Add extension: IPv6 BGP next IP
Add extension: src/dst vlan id
Add extension: 4 byte output packets
Add extension: 8 byte output packets
Add extension: 4 byte output bytes
Add extension: 8 byte output bytes
Add extension: 4 byte aggregated flows
Add extension: 8 byte aggregated flows
Add extension: in src/out dst mac address
Add extension: in dst/out src mac address
Add extension: MPLS Labels
Add extension: IPv4 router IP addr
Add extension: IPv6 router IP addr
Add extension: router ID
Add extension: BGP adjacent prev/next AS
Add extension: time packet received
Add extension: NSEL Common block
Add extension: NSEL xlate ports
Add extension: NSEL xlate IPv4 addr
Add extension: NSEL xlate IPv6 addr
Add extension: NSEL ACL ingress/egress acl ID
Add extension: NSEL username
Add extension: NSEL max username
Add extension: nprobe/nfpcapd latency
Add extension: NEL Common block
Add extension: Compat NEL IPv4
Add extension: NAT Port Block Allocation
Bound to IPv4 host/IP: 10.0.0.3, Port: 9995
Startup.
Init IPFIX: Max number of IPFIX tags: 72
Process_v5: New exporter: SysID: 1 engine id 17, type 17, IP: 10.173.78.66, Sampling Mode: 0, Sampling Interval: 1

^CIdent: 'none' Flows: 16, Packets: 2267125, Bytes: 51329316845, Sequence Errors: 1, Bad Packets: 0
Total ignored packets: 0
Terminating nfcapd.
root@d1:/nfdump#
    
```

Figure 20. Verifying the exporter connectivity.

Consider the figure above. It shows that the new exporter was detected and includes information about the exporter such as total flows (16), number of packets (2267125) and bytes (51329316845).

If total flow number is 0, run the collector and perform the test again. You might get different number of flows and bytes.

Step 2. Type the following command to verify the nfcapd file stored into *netflow_files*.

```
ls /nfdump/netflow_files | tail -1 > tmp ; cat tmp
```

```

root@d1:/nfdump# ls /nfdump/netflow_files | tail -1 > tmp ; cat tmp
nfcapd.202207062210
root@d1:/nfdump#
    
```

Figure 21. Verifying nfcapd files.

Consider the output in the figure above. It shows a new file added named nfcapd.202207062205 (Year 2022, Month 07, Date 06, Time 22:05).

You will see a file that is named based on the time you are running the test.

Nfcapd stores file for every 5 minutes (e.g., one file from 15:45 to 15:50). You might get 2 files if the exporter is running for more than 5 minutes (e.g., 5:45 to 15:52).

Step 3. Type the following command to read the file that was currently stored. Note the symbols enclosing the command `cat tmp` are backticks ``` note single quotations.

```
nfdump -r /nfdump/netflow_files/`cat tmp`
```

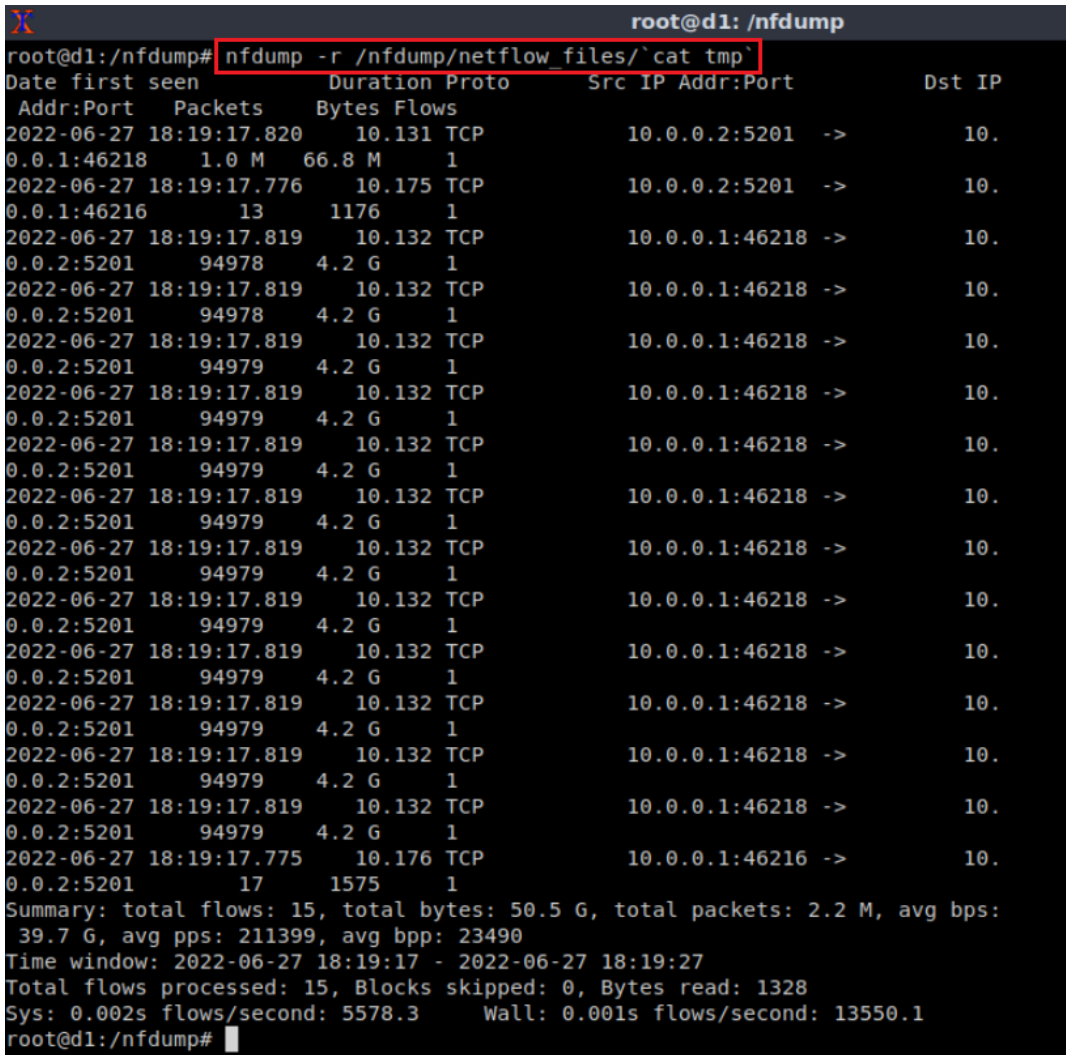


Figure 22. Reading the recorded file.

Consider the figure above. The output includes the date, time, duration, protocol, source and destination IP addresses, ports, packets, bytes, and flows. It also provides a summary of the test.

You can read any file if you get more than 1 file stored in the folder.

5 Analyzing IPFIX data using Nfdump

In this section, you will enable IPFIX agent and collector. You will also perform a connectivity test between hosts. Finally, you will verify the stored file using nfdump.

5.1 Launching IPFIX collector and exporter

Step 1. Type the following command to start the nfcapd daemon. The daemon will start reading data from the exporter and stores in a file.

```
nfcapd -b 10.0.0.3 -p 9995 -l /nfdump/ipfix_files/
```

```

root@d1:/nfdump# nfcapd -b 10.0.0.3 -p 9995 -l /nfdump/ipfix_files/
Add extension: 2 byte input/output interface index
Add extension: 4 byte input/output interface index
Add extension: 2 byte src/dst AS number
Add extension: 4 byte src/dst AS number
Add extension: dst tos, direction, src/dst mask
Add extension: IPv4 next hop
Add extension: IPv6 next hop
Add extension: IPv4 BGP next IP
Add extension: IPv6 BGP next IP
Add extension: src/dst vlan id
Add extension: 4 byte output packets
Add extension: 8 byte output packets
Add extension: 4 byte output bytes
Add extension: 8 byte output bytes
Add extension: 4 byte aggregated flows
Add extension: 8 byte aggregated flows
Add extension: in src/out dst mac address
Add extension: in dst/out src mac address
Add extension: MPLS Labels
Add extension: IPv4 router IP addr
Add extension: IPv6 router IP addr
Add extension: router ID
Add extension: BGP adjacent prev/next AS
Add extension: time packet received
Add extension: NSEL Common block
Add extension: NSEL xlate ports
Add extension: NSEL xlate IPv4 addr
Add extension: NSEL xlate IPv6 addr
Add extension: NSEL ACL ingress/egress acl ID
Add extension: NSEL username
Add extension: NSEL max username
Add extension: nprobe/nfcapd latency
Add extension: NEL Common block
Add extension: Compat NEL IPv4
Add extension: NAT Port Block Allocation
Bound to IPv4 host/IP: 10.0.0.3, Port: 9995
Startup.
Init IPFIX: Max number of IPFIX tags: 72

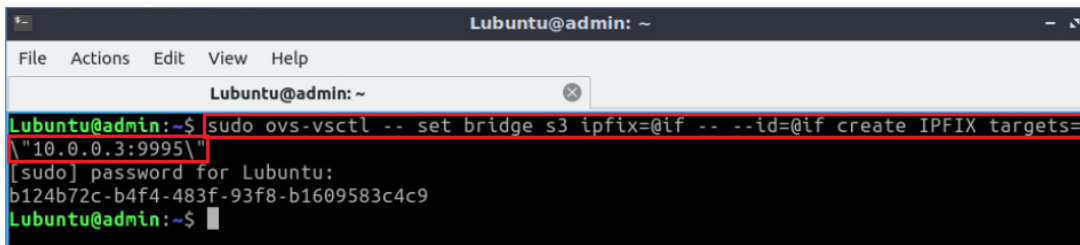
```

Figure 23. Starting nfcapd daemon.

Consider the figure above. `-b` specifies the host address 10.0.0.3 to bind for listening. `-p` specifies the port number 9995. `-l` is referring to the directory where the file will be saved. The figure shows that the host was bound to the daemon.

Step 2. Go back to the Linux terminal and execute the following script to start the IPFIX exporter. If prompted for password, type `password`.

```
sudo ovs-vsctl -- set bridge s3 ipfix=@if -- --id=@if create IPFIX
targets=\"10.0.0.3:9995\"
```

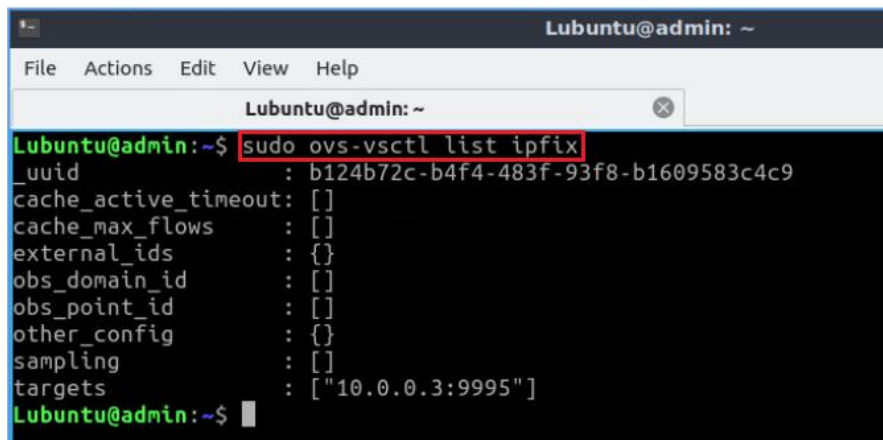
```
Lubuntu@admin: ~  
File Actions Edit View Help  
Lubuntu@admin: ~  
Lubuntu@admin:~$ sudo ovs-vsctl -- set bridge s3 ipfix=@if -- --id=@if create IPFIX targets=  
"10.0.0.3:9995\  
[sudo] password for Lubuntu:  
b124b72c-b4f4-483f-93f8-b1609583c4c9  
Lubuntu@admin:~$
```

Figure 24. Starting IPFIX exporter.

Consider the figure above. The command creates an IPFIX ID and is attached to switch s3. Switch s3 is acting as an exporter and transmits data to the collector. 10.0.0.3 is the collector IP and the port is the default UDP port 9995.

Step 3. Type the following command to verify IPFIX configuration.

```
sudo ovs-vsctl list ipfix
```



```
Lubuntu@admin: ~  
File Actions Edit View Help  
Lubuntu@admin: ~  
Lubuntu@admin:~$ sudo ovs-vsctl list ipfix  
_uuid : b124b72c-b4f4-483f-93f8-b1609583c4c9  
cache_active_timeout: []  
cache_max_flows : []  
external_ids : {}  
obs_domain_id : []  
obs_point_id : []  
other_config : {}  
sampling : []  
targets : ["10.0.0.3:9995"]  
Lubuntu@admin:~$
```

Figure 25. Verifying IPFIX configuration.

Consider the figure above. Exporter s3 is running, target IP address is 10.0.0.3 and the port is 9995.

Step 4. Open host h1 terminal and type the following command to perform a test.

```
iperf3 -c 10.0.0.2
```

```

Host: h1
root@admin:~# iperf3 -c 10.0.0.2
Connecting to host 10.0.0.2, port 5201
[ 7] local 10.0.0.1 port 41792 connected to 10.0.0.2 port 5201
[ ID] Interval      Transfer    Bitrate     Retr  Cwnd
[ 7]  0.00-1.00    sec  4.05 GBytes 34.8 Gbits/sec  0   5.96 MBytes
[ 7]  1.00-2.00    sec  4.17 GBytes 35.8 Gbits/sec  0   7.25 MBytes
[ 7]  2.00-3.00    sec  4.26 GBytes 36.6 Gbits/sec  0   7.61 MBytes
[ 7]  3.00-4.00    sec  4.16 GBytes 35.7 Gbits/sec 135  5.88 MBytes
[ 7]  4.00-5.00    sec  4.42 GBytes 37.9 Gbits/sec  0   6.08 MBytes
[ 7]  5.00-6.00    sec  4.45 GBytes 38.2 Gbits/sec  0   6.08 MBytes
[ 7]  6.00-7.00    sec  4.45 GBytes 38.2 Gbits/sec  0   6.08 MBytes
[ 7]  7.00-8.00    sec  4.47 GBytes 38.4 Gbits/sec  0   7.34 MBytes
[ 7]  8.00-9.00    sec  4.27 GBytes 36.7 Gbits/sec  0   7.48 MBytes
[ 7]  9.00-10.00   sec  4.43 GBytes 38.1 Gbits/sec  0   7.69 MBytes

-----
[ ID] Interval      Transfer    Bitrate     Retr
[ 7]  0.00-10.00   sec  43.1 GBytes 37.0 Gbits/sec 135
[ 7]  0.00-10.04   sec  43.1 GBytes 36.9 Gbits/sec
sender
receive

iperf Done.
root@admin:~#
  
```

Figure 26. Performing iperf3 test.

Consider the figure above. The test runs for ten seconds with an interval of one second.

5.2 Visualizing IPFIX data stored by nfcapd

Step 1. Go to docker d1 terminal to verify the exporter was detected to examine the packets. Press `Ctrl+c` to stop the nfcapd daemon.


```

root@d1: /nfdump
Process_ipfix: [0] Add template 336
Process_ipfix: [0] Add template 337
Process_ipfix: [0] Add template 338
Process_ipfix: [0] Add template 339
Process_ipfix: [0] Add template 340
Process_ipfix: [0] Add template 341
Process_ipfix: [0] Add template 342
Process_ipfix: [0] Add template 343
Process_ipfix: [0] Add template 344
Process_ipfix: [0] Add template 345
Process_ipfix: [0] Add template 346
Process_ipfix: [0] Add template 347
Process_ipfix: [0] Add template 348
Process_ipfix: [0] Add template 349
Process_ipfix: [0] Add template 350
Process_ipfix: [0] Add template 351
Process_ipfix: [0] Add template 352
Process_ipfix: [0] Add template 353
Process_ipfix: [0] Add template 354
Process_ipfix: [0] Add template 355
Process_ipfix: [0] Add template 356
Process_ipfix: [0] Add template 357
Process_ipfix: [0] Add template 358
Process_ipfix: [0] Add template 359
Process_ipfix: [0] Add template 360
Process_ipfix: [0] Add template 361
Process_ipfix: [0] Add template 362
Process_ipfix: [0] Add template 363
Process_ipfix: [0] Add template 364
Process_ipfix: [0] Add template 365
Process_ipfix: [0] Add template 366
Process_ipfix: [0] Add template 367
Process_ipfix: [0] Add template 368
Process_ipfix: [0] Add template 369
Process_ipfix: [0] Add template 370
Process_ipfix: [0] Add template 371
Process_ipfix: [0] Add template 372
Process_ipfix: [0] Add template 373
Process_ipfix: [0] Add template 374
Process_ipfix: [0] Add template 375
Process_v5: New exporter: SysID: 2, engine id 17, type 17, IP: 172.17.0.1, Sampling Mode: 0, Sampling Interval: 1
^CIdent: 'none' Flows: 161869, Packets: 16164936857708, Bytes: 23712250012432262 Sequence Errors: 2, Bad Packets: 0
Total ignored packets: 0
Terminating nfcapd.
root@d1: /nfdump#

```

Figure 27. Verifying the exporter connectivity.

Consider the figure above. It shows that the new exporter was detected and includes information about the exporter such as total flows, number of packets and bytes.

If total flow number is 0, run the collector and perform the test again.

Step 2. Type the following command to verify the nfcapd file stored into *ipfix_files*.

```
ls /nfdump/ipfix_files | tail -1 > tmp ; cat tmp
```

```

root@d1: /nfdump
root@d1:/nfdump# ls /nfdump/ipfix_files | tail -1 > tmp ; cat tmp
nfcapd.202207062220
root@d1: /nfdump#

```

Figure 28. Verifying nfcapd files.

Consider the figure above. It shows a new file added named nfcapd.202207062220 (Year 2022, Month 07, Date 06, Time 22:20).

You will see a file that is named based on the time you are running the test.

Step 3. Type the following command to read the file that was currently stored. Since the total flow number is 161869, it will take time to load all the flows. For simplicity, you will

use `-c 15` to show top 15 flows from the list. Note the symbols enclosing the command `cat tmp` are backticks ``` note single quotations.

```
nfdump -r /nfdump/ipfix_files/`cat tmp` -c 15
```

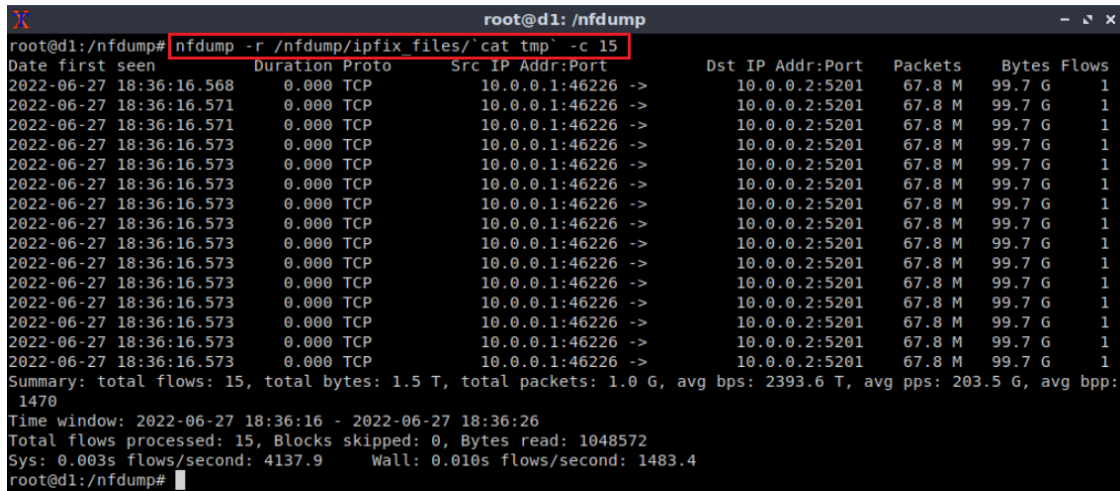


Figure 29. Reading the recorded file.

Consider the figure above. The output includes the date, time, duration, protocol, source and destination IP addresses, ports, packets, bytes, and flows. It also provides a summary of the test. Since nfdump uses a fixed output format, the result is similar to netflow data.

6 Analyzing sFlow data using Nfdump

In this section, you will enable sFlow agent and collector. You will also perform a connectivity test between hosts. Finally, you will verify the stored file using nfdump.

6.1 Launching sFlow collector and agent

Step 1. Type the following command to start the sfcapd daemon. The daemon will start reading data from the agent and stores in a file.

```
sfcapd -b 10.0.0.3 -p 9995 -l /nfdump/sflow_files/
```

```

root@d1:/nfdump# sfcapd -b 10.0.0.3 -p 9995 -l /nfdump/sflow files/
Add extension: 2 byte input/output interface index
Add extension: 4 byte input/output interface index
Add extension: 2 byte src/dst AS number
Add extension: 4 byte src/dst AS number
Add extension: dst tos, direction, src/dst mask
Add extension: IPv4 next hop
Add extension: IPv6 next hop
Add extension: IPv4 BGP next IP
Add extension: IPv6 BGP next IP
Add extension: src/dst vlan id
Add extension: 4 byte output packets
Add extension: 8 byte output packets
Add extension: 4 byte output bytes
Add extension: 8 byte output bytes
Add extension: 4 byte aggregated flows
Add extension: 8 byte aggregated flows
Add extension: in src/out dst mac address
Add extension: in dst/out src mac address
Add extension: MPLS Labels
Add extension: IPv4 router IP addr
Add extension: IPv6 router IP addr
Add extension: router ID
Add extension: BGP adjacent prev/next AS
Add extension: time packet received
Add extension: NSEL Common block
Add extension: NSEL xlate ports
Add extension: NSEL xlate IPv4 addr
Add extension: NSEL xlate IPv6 addr
Add extension: NSEL ACL ingress/egress acl ID
Add extension: NSEL username
Add extension: NSEL max username
Add extension: nprobe/nfpcapd latency
Add extension: NEL Common block
Add extension: Compat NEL IPv4
Add extension: NAT Port Block Allocation
Bound to IPv4 host/IP: 10.0.0.3, Port: 9995
Startup.
    
```

Figure 30. Starting sfcapd daemon.

Consider the figure above. `-b` specifies the host address 10.0.0.3 to bind for listening. `-p` specifies the port number 9995. `-l` is referring to the directory where the file will be saved. The figure shows that the host was bound to the daemon.

Step 2. Go back to the Linux terminal and execute the following script to enable a sFlow agent. If prompted for password, type `password`.

```

sudo ovs-vsctl -- --id=@s create sflow agent=s3-eth1 target="\10.0.0.3:9995\"
sampling=64 polling=10 -- set bridge s3 sflow=@s
    
```

```

Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~
Lubuntu@admin:~$ sudo ovs-vsctl -- --id=@s create sflow agent=s3-eth1 target="\10.0.0.3:9995\"
\" sampling=64 polling=10 -- set bridge s3 sflow=@s
[sudo] password for Lubuntu:
5c716827-7544-44ab-b68d-2093aab52c87
Lubuntu@admin:~$
    
```

Figure 31. Enabling sFlow agent.

Consider the figure above. The command creates a sFlow ID and is attached to switch s3. Switch s3 is acting as an agent and transmits data to the collector. 10.0.0.3 is the collector IP and the port is the default UDP port 9995.

Step 3. Type the following command to verify sFlow configuration.

```
sudo ovs-vsctl list sflow
```

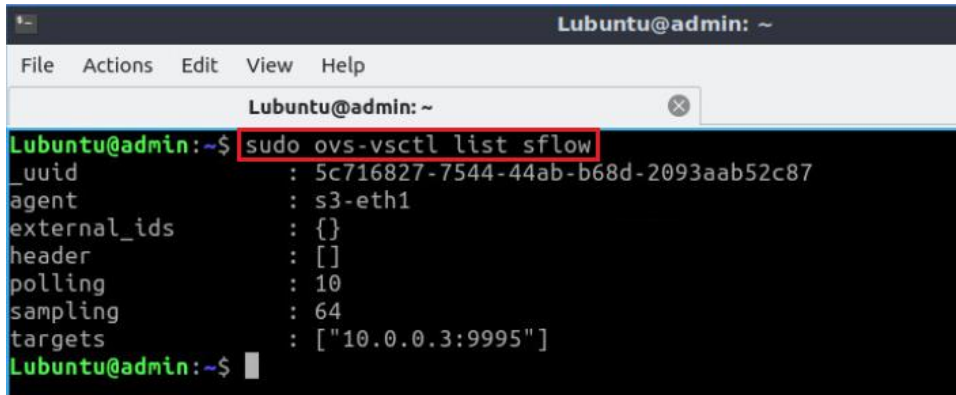


Figure 32. Verifying sFlow configuration.

Consider the figure above. s3 agent is running, target IP address is 10.0.0.3 and the port is 9995.

Step 4. Open host h1 terminal and type the following command to perform iperf3 test.

```
iperf3 -c 10.0.0.2
```

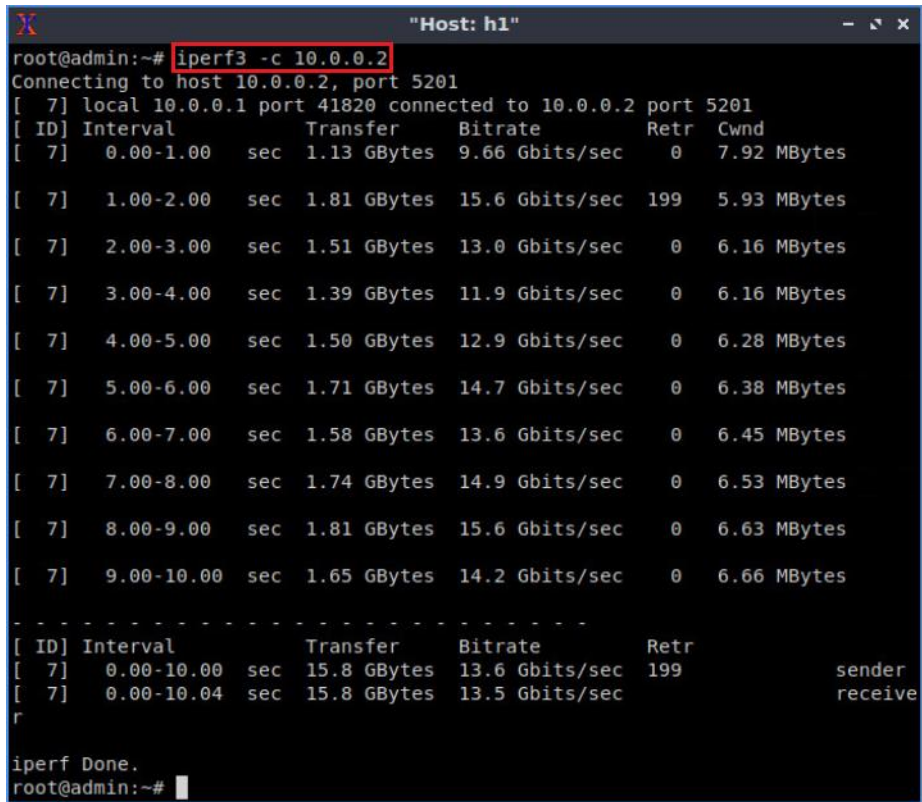


Figure 33. Performing iperf3 test.

Consider the figure above. The test runs for ten seconds with an interval of one second.

6.2 Visualizing sFlow data stored by sfcapd

Step 1. Go to docker d1 terminal to verify that the exporter was detected to examine the packets. Press `Ctrl+c` to stop the sfcapd daemon.

```

root@d1: /nfdump
Add extension: dst tos, direction, src/dst mask
Add extension: IPv4 next hop
Add extension: IPv6 next hop
Add extension: IPv4 BGP next IP
Add extension: IPv6 BGP next IP
Add extension: src/dst vlan id
Add extension: 4 byte output packets
Add extension: 8 byte output packets
Add extension: 4 byte output bytes
Add extension: 8 byte output bytes
Add extension: 4 byte aggregated flows
Add extension: 8 byte aggregated flows
Add extension: in src/out dst mac address
Add extension: in dst/out src mac address
Add extension: MPLS Labels
Add extension: IPv4 router IP addr
Add extension: IPv6 router IP addr
Add extension: router ID
Add extension: BGP adjacent prev/next AS
Add extension: time packet received
Add extension: NSEL Common block
Add extension: NSEL xlate ports
Add extension: NSEL xlate IPv4 addr
Add extension: NSEL xlate IPv6 addr
Add extension: NSEL ACL ingress/egress acl ID
Add extension: NSEL username
Add extension: NSEL max username
Add extension: nprobe/nfpcapd latency
Add extension: NEL Common block
Add extension: Compat NEL IPv4
Add extension: NAT Port Block Allocation
Bound to IPv4 host/IP: 10.0.0.3, Port: 9995
Startup.
SFLOW: New exporter
SFLOW: New exporter: SysID: 1, agentSubId: 0, MeanSkipCount: 64, IP: 10.173.7
8.66
SFLOW: setup extension map: 0
SFLOW: setup extension map 0
Extension size: 100
Extension map size: 32
New extension map id: 0
SFLOW: setup extension map: 0 done
^CIdent: 'none' Flows: 438382, Packets: 28056448, Bytes: 41838903552, Sequence Errors: 0, Bad Packets: 0
Total ignored packets: 0
Terminating sfcapd.
root@d1:/nfdump#
    
```

Figure 34. Verifying the exporter connectivity.

Consider the figure above. It shows that the new exporter was detected and includes information about the exporter such as total flows, number of packets and bytes.

If total flow number is 0, run the collector and perform the test again.

Step 2. Type the following command to verify the nfcapd file stored into *sflow_files*.

```
ls /nfdump/sflow_files | tail -1 > tmp ; cat tmp
```

```

root@d1: /nfdump
root@d1:/nfdump# ls /nfdump/sflow_files/ | tail -1 > tmp ; cat tmp
nfcapd.202207062227
root@d1:/nfdump#
    
```

Figure 35. Verifying nfcapd files.

Nfdump stores sFlow files as nfcapd file though the daemon for sFlow is sfcapd. Consider the figure above. It shows a new file added named nfcapd.202207062227 (Year 2022, Month 07, Date 06, Time 22:27).

You will see a file that is named based on the time you are running the test.

Step 3. Type the following command to read the file that was currently stored. Since the total flow number is 438382, it will take time to load all the flows. For simplicity, you will use `-c 15` to show top 15 flows from the list.

```
nfdump -r /nfdump/sflow_files/`cat tmp` -c 15
```

```

root@d1: /nfdump
root@d1:/nfdump# nfdump -r /nfdump/sflow_files/`cat tmp` -c 15
Date first seen      Duration Proto      Src IP Addr:Port    Dst IP Addr:Port    Packets  Bytes  Flows
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.2:5201 -> 10.0.0.1:46230     64      4480   1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.2:5201 -> 10.0.0.1:46230     64      4480   1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.740  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.741  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.741  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
2022-06-27 18:47:27.741  0.000 TCP        10.0.0.1:46230 -> 10.0.0.2:5201      64      97152  1
Summary: total flows: 15, total bytes: 1.3 M, total packets: 960, avg bps: 10.2 G, avg pps: 960000, avg bpp: 1324
Time window: 2022-06-27 18:47:27 - 2022-06-27 18:47:38
Total flows processed: 15, Blocks skipped: 0, Bytes read: 1048524
Sys: 0.004s flows/second: 3542.7    Wall: 0.006s flows/second: 2293.6
root@d1:/nfdump#

```

Figure 36. Reading the recorded file.

Consider the figure above. The output includes the date, time, duration, protocol, source and destination IP addresses, ports, packets, bytes, and flows. It also provides a summary of the test.

This concludes Lab 5. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

References

1. DNSstuff, “Best NetFlow Analyzer and Collectors”, [Online].
<https://www.dnsstuff.com/netflow-analyzer-software#best-netflow-analyzers-and-collectors-list>
2. P. Phaal, S. Panchen, N. Mckee, InMon Corp, “InMon Corporation’s sFlow: A Method for Monitoring Traffic in Switched and Routed Networks”, [Online].
<https://datatracker.ietf.org/doc/html/rfc3176>
3. Sourceforge, “Nfdump”, [Online]. <http://nfdump.sourceforge.net/>
4. Omar Santos, “Network security with NetFlow and IPFIX”, (2016).
5. sFlow, “Traffic monitoring using sFlow”, [Online].
<https://sflow.org/sFlowOverview.pdf>
6. IBM, “IPFIX”, [Online].
<https://www.ibm.com/docs/en/qradar-on-cloud?topic=sources-ipfix>
7. Cisco, “Introduction to Cisco IOS NetFlow – A technical overview”, [Online].
https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

8. B. Claise, "*RFC 3954: Cisco systems NetFlow services export version 9*", (2004).
<https://tools.ietf.org/html/rfc3954>



UNIVERSITY OF
SOUTH CAROLINA

NETWORK MANAGEMENT

Lab 6: Filtering and Formatting Data using Nfdump

Document Version: **07-08-2022**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Introduction	3
1.1 Introduction to Nfdump.....	4
1.2 Processing data using Nfdump.....	4
2 Lab topology.....	5
2.1 Lab settings.....	5
2.2 Loading a topology	5
3 Exploring Nfdump features.....	8
3.1 Nfdump formatting features.....	9
3.2 Nfdump filtering features.....	12
References	16

Overview

This lab introduces Nfdump, a set of tools to analyze NetFlow, sFlow and IPFIX data as well as to track traffic patterns continuously. The focus of this lab is to show formatting and filtering features of Nfdump.

Objectives

By the end of this lab, you should be able to:

1. Understand the concept of Nfdump.
2. Analyze NetFlow data using Nfdump.
3. Explore Nfdump formatting and filtering features.

Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Launching NetFlow collector and exporter.
4. Section 4: Analyzing NetFlow data using Nfdump.

1 Introduction

NetFlow is designed for the collection of IP traffic information and the monitoring of network traffic. It provides a detailed view of application flows and a popular way of gaining both a broad and detailed picture of what is happening inside the network. With the help of NetFlow collector and NetFlow analyzer tools, it is possible to visualize where network traffic ends up, the source of the traffic, and how much traffic is being generated. You can gain increased visibility into bandwidth allocation, identify, and prevent any further abuse potentially impacting the performance of the network³.

1.1 Introduction to Nfdump

Nfdump is a very popular command-line toolset for collecting, storing, and processing NetFlow/SFlow/IPFIX data. This utility can process data very fast. Nfcapd is the NetFlow and IPFIX capture daemon that reads the data from the network and stores the data into files. Sfcapd is the sFlow capture daemon. Nfdump reads the data from the files stored by the daemons. All the data is stored as nfcapd file. Every five minutes, nfcapd rotates and renames the output file with the time stamp nfcapd.YYMMddhhmm of the interval (e.g., nfcapd.202103221140 contains data from March 22nd, 2021, time 11:40 onward). It can save 288 files per day based on five minutes time interval. Nfdump also offers a graphical web-based front-end tool called nfsen-ng which allows processing the NetFlow data within the specified time span, easily navigate through the collected data, set alerts, based on various conditions. It can process a file stored by nfcapd every five minutes³.

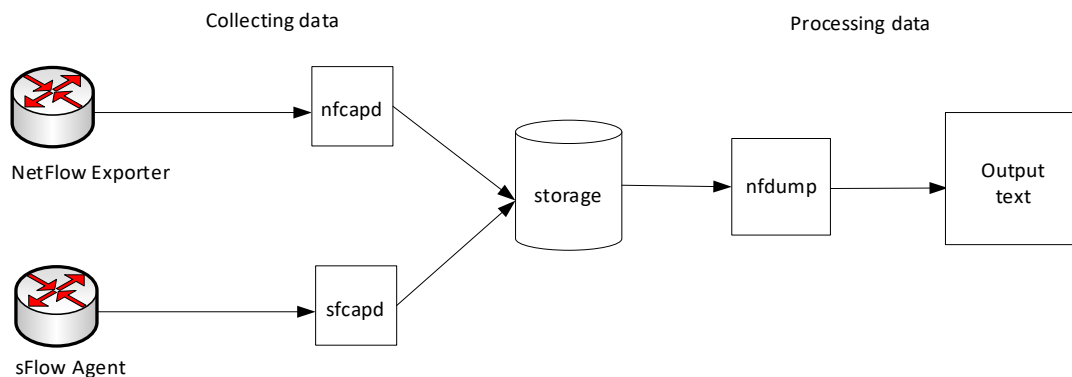


Figure 1. Nfdump architecture.

Consider Figure 1. NetFlow and sFlow daemons are capturing and saving data from routers. Nfdump reads the stored data from the files and delivers as text based on the queries.

1.2 Processing data using Nfdump

Flows can be read either from single file or from a sequence of files. Nfdump has several output formats (line, long, extended). The output format line is the default output format when no format is specified. It limits the information to the connection details as well as number of packets, bytes, and flows. The output format long is identical to the format line and includes additional information such as TCP flags and Type of Service (Tos). The output format extended is identical to the format long and includes additional computed information such as packet per second (pps), bytes per second (bps) and bits per packet (bpp)⁴.

Nfdump has a powerful and fast filter engine. All flows are filtered before they are further processed. If no filter is given, any flow will be processed. The filter is either given on the command line as last argument enclosed in ', or in a file³.

2 Lab topology

Consider Figure 2. There are three switches, four end hosts, and a docker container. Switch s3 is acting as a NetFlow exporter and the docker d1 is the collector.

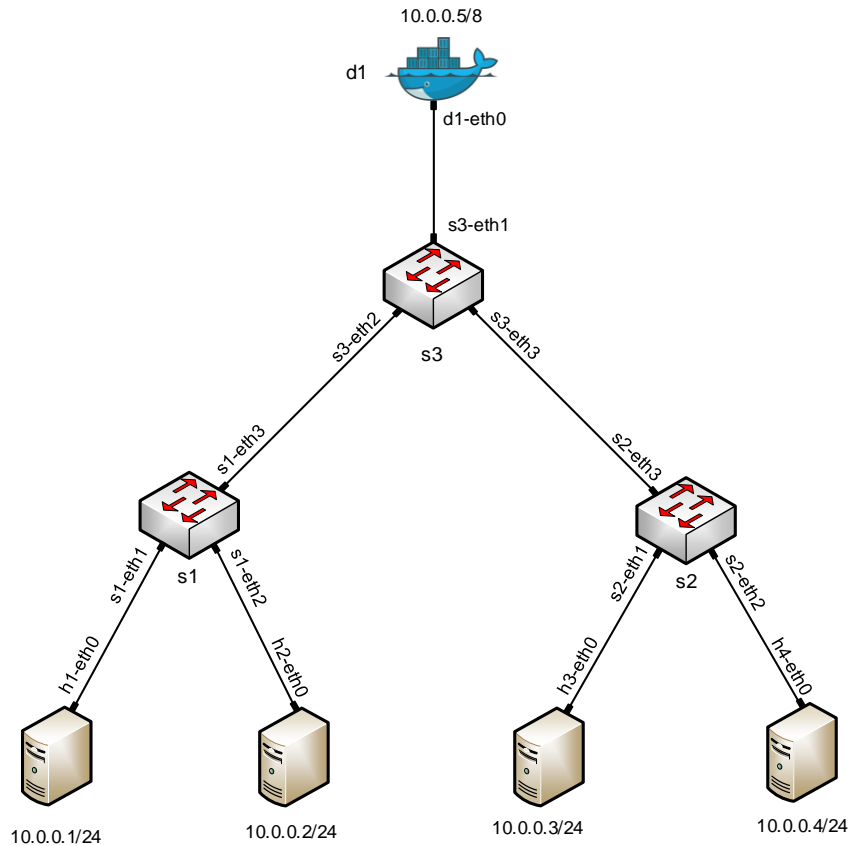


Figure 2. Lab topology.

2.1 Lab settings

The devices should be configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8
h3	h3-eth0	10.0.0.3	/8
h4	h4-eth0	10.0.0.4	/8
d1	d1-eth0	10.0.0.5	/8

2.2 Loading a topology

Step 1. Click on the Client tab to access the Client PC.

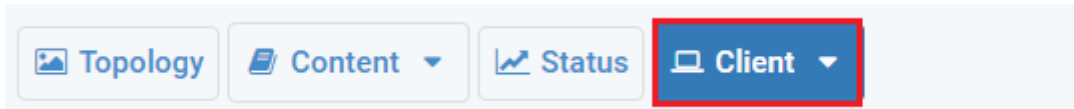


Figure 3. Accessing the Client PC.

Step 2. Start by launching MiniEdit by clicking on desktop's shortcut. When prompted for a password, type `password`.

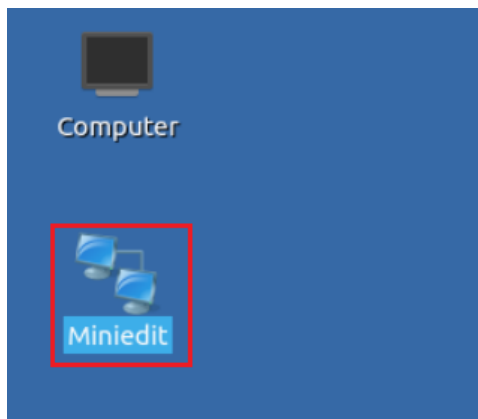


Figure 4. MiniEdit shortcut.

Step 3. On MiniEdit's menu bar, click on *File* then open to load the lab's topology. Open the directory called *lab6* and select the file *lab6.mn*. Then, click on *Open* to open the topology.

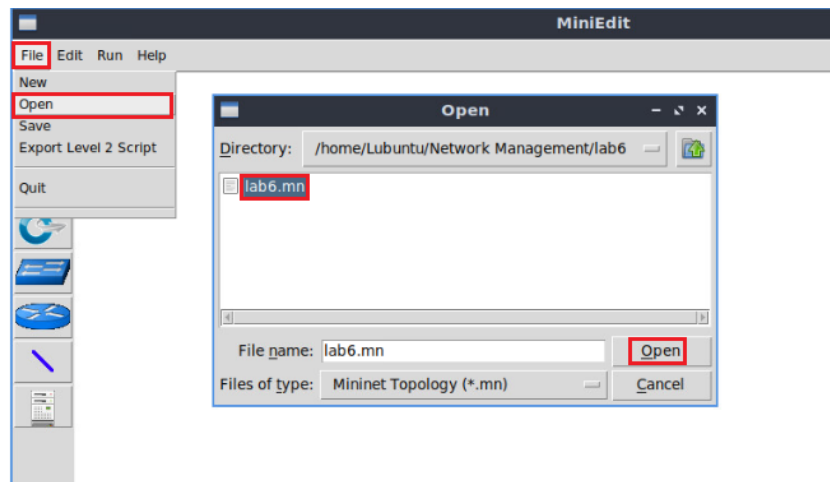


Figure 5. MiniEdit's Open dialog.

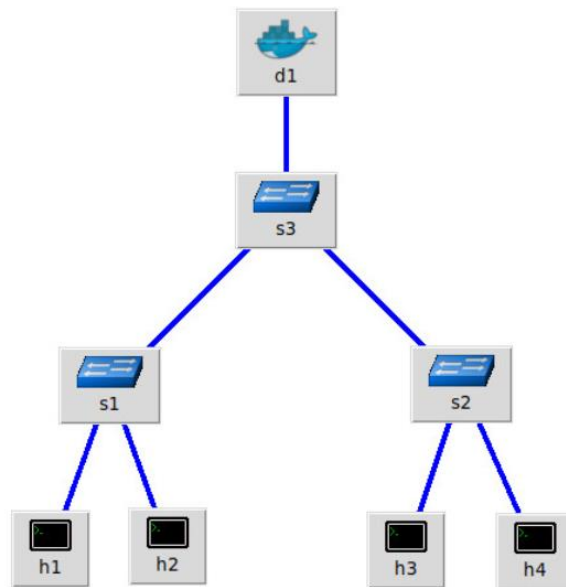


Figure 6. MiniEdit's topology.

Step 4. To proceed with the emulation, click on the *Run* button located on the lower left-hand side.

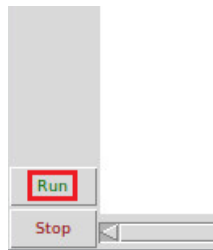


Figure 7. Starting the emulation.

Step 5. Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

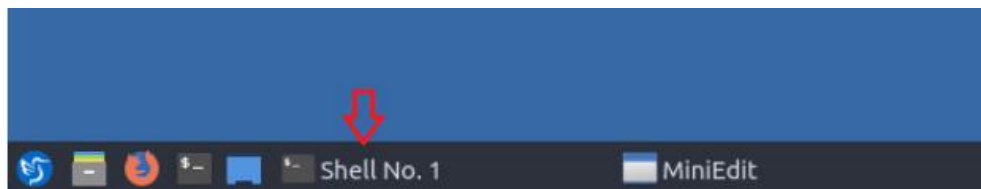


Figure 8. Opening Mininet's terminal.

Step 6. Issue the following command to display the interface names and connections.

```
links
```

```

File  Actions  Edit  View  Help
Shell No. 1
containernet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
h3-eth0<->s2-eth1 (OK OK)
h4-eth0<->s2-eth2 (OK OK)
d1-eth0<->s3-eth1 (OK OK)
s1-eth3<->s3-eth2 (OK OK)
s3-eth3<->s2-eth3 (OK OK)
containernet>
    
```

Figure 9. Displaying network interfaces.

In figure 9, the link displayed within the gray box indicates that interface eth0 of host h1 connects to interface eth1 of switch s1 (i.e., *h1-eth0<->s1-eth1*).

3 Exploring Nfdump features

Step 1. Go back to MiniEdit. Hold right-click on docker d1 and select *Terminal*. This opens the terminal of the docker and allows the execution of commands.

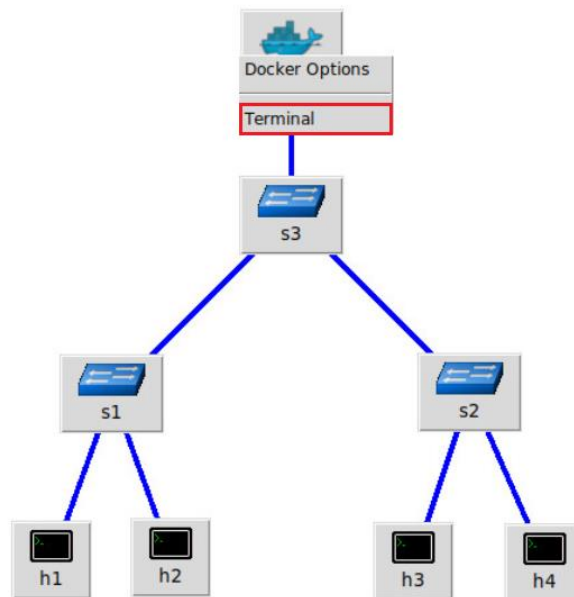


Figure 10. Opening a terminal on docker d1.

Step 2. Navigate into */nfdump/nfcapd_files/* directory by issuing the following command.

```
cd nfdump/nfcapd_files/
```

```

root@d1: /nfdump/nfcapd_files
root@d1:/# cd nfdump/nfcapd_files/
root@d1:/nfdump/nfcapd_files#
    
```

Figure 11. Entering into nfdump directory.

Step 3. Type the following command to show the list of files that are stored already for exploring nfdump features.

```
ls
```

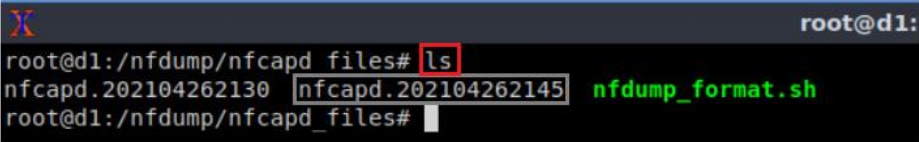
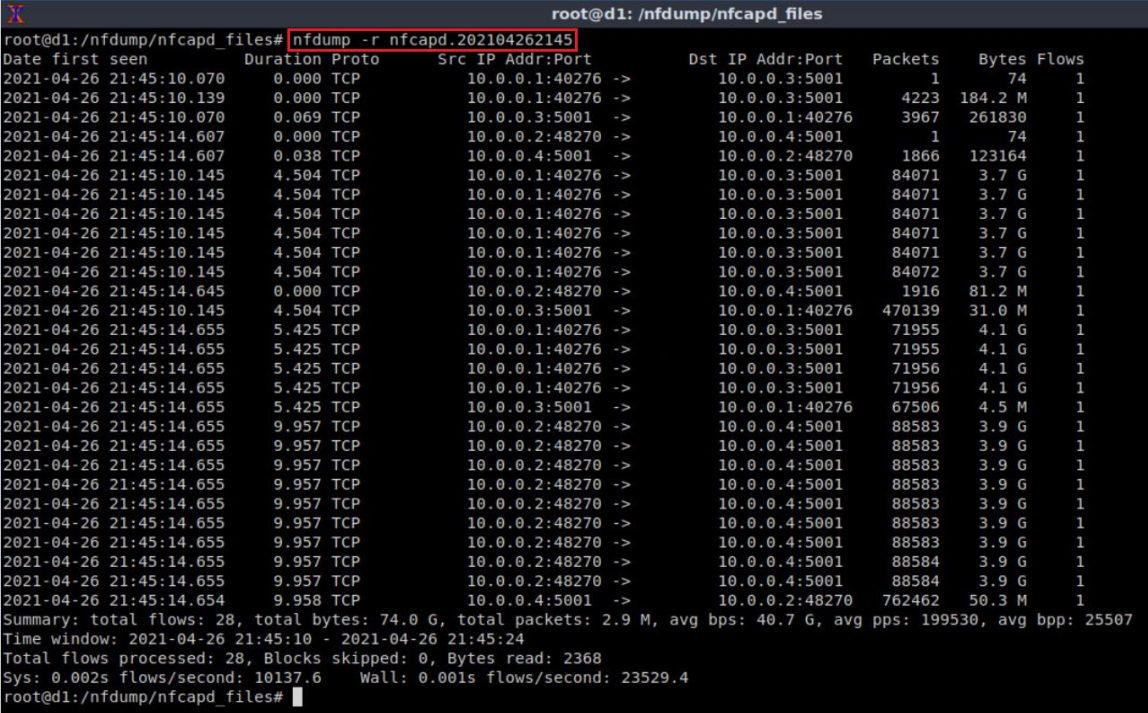


Figure 12. Verifying the recorded file.

Consider the figure above. It shows two files (e.g., nfcapd.202104262145 which refers to Year 2021, Month 04, Date 26, Time 21:45).

Step 4. Type the following command to read file *nfcapd.202104262145*.

```
nfdump -r nfcapd.202104262145
```



Date first seen	Duration	Proto	Src IP Addr:Port	Dst IP Addr:Port	Packets	Bytes	Flows
2021-04-26 21:45:10.070	0.000	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	1	74	1
2021-04-26 21:45:10.139	0.000	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	4223	184.2 M	1
2021-04-26 21:45:10.070	0.069	TCP	10.0.0.3:5001 ->	10.0.0.1:40276	3967	261830	1
2021-04-26 21:45:14.607	0.000	TCP	10.0.0.2:48270 ->	10.0.0.4:5001	1	74	1
2021-04-26 21:45:14.607	0.038	TCP	10.0.0.4:5001 ->	10.0.0.2:48270	1866	123164	1
2021-04-26 21:45:10.145	4.504	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	84071	3.7 G	1
2021-04-26 21:45:10.145	4.504	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	84071	3.7 G	1
2021-04-26 21:45:10.145	4.504	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	84071	3.7 G	1
2021-04-26 21:45:10.145	4.504	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	84071	3.7 G	1
2021-04-26 21:45:10.145	4.504	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	84071	3.7 G	1
2021-04-26 21:45:10.145	4.504	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	84072	3.7 G	1
2021-04-26 21:45:14.645	0.000	TCP	10.0.0.2:48270 ->	10.0.0.4:5001	1916	81.2 M	1
2021-04-26 21:45:10.145	4.504	TCP	10.0.0.3:5001 ->	10.0.0.1:40276	470139	31.0 M	1
2021-04-26 21:45:14.655	5.425	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	71955	4.1 G	1
2021-04-26 21:45:14.655	5.425	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	71955	4.1 G	1
2021-04-26 21:45:14.655	5.425	TCP	10.0.0.1:40276 ->	10.0.0.3:5001	71956	4.1 G	1
2021-04-26 21:45:14.655	5.425	TCP	10.0.0.2:48270 ->	10.0.0.3:5001	71956	4.1 G	1
2021-04-26 21:45:14.655	5.425	TCP	10.0.0.3:5001 ->	10.0.0.1:40276	67506	4.5 M	1
2021-04-26 21:45:14.655	9.957	TCP	10.0.0.2:48270 ->	10.0.0.4:5001	88583	3.9 G	1
2021-04-26 21:45:14.655	9.957	TCP	10.0.0.2:48270 ->	10.0.0.4:5001	88583	3.9 G	1
2021-04-26 21:45:14.655	9.957	TCP	10.0.0.2:48270 ->	10.0.0.4:5001	88583	3.9 G	1
2021-04-26 21:45:14.655	9.957	TCP	10.0.0.2:48270 ->	10.0.0.4:5001	88583	3.9 G	1
2021-04-26 21:45:14.655	9.957	TCP	10.0.0.2:48270 ->	10.0.0.4:5001	88583	3.9 G	1
2021-04-26 21:45:14.655	9.957	TCP	10.0.0.2:48270 ->	10.0.0.4:5001	88584	3.9 G	1
2021-04-26 21:45:14.655	9.957	TCP	10.0.0.2:48270 ->	10.0.0.4:5001	88584	3.9 G	1
2021-04-26 21:45:14.654	9.958	TCP	10.0.0.4:5001 ->	10.0.0.2:48270	762462	50.3 M	1

Summary: total flows: 28, total bytes: 74.0 G, total packets: 2.9 M, avg bps: 40.7 G, avg pps: 199530, avg bpp: 25507
 Time window: 2021-04-26 21:45:10 - 2021-04-26 21:45:24
 Total flows processed: 28, Blocks skipped: 0, Bytes read: 2368
 Sys: 0.002s flows/second: 10137.6 Wall: 0.001s flows/second: 23529.4

Figure 13. Reading the recorded file.

Consider the figure above. The output includes the date, time, duration, protocol, source and destination IP addresses, ports, packets, bytes, and flows. It also provides a summary of the test.

3.1 Nfdump formatting features

Step 1. Type the following command to see the long format output.

```
nfdump -r nfcapd.202104262145 -o long
```



```
./nfdump_format.sh
```

```

root@d1:~/nfdump/nfcapd_files# ./nfdump_format.sh
Date first seen      Date last seen      Src IP Addr:Port      Dst IP Addr:Port      Packets      Bytes
2021-04-26 21:45:10.070 2021-04-26 21:45:10.070 10.0.0.1:40276 -> 10.0.0.3:5001 1 74
2021-04-26 21:45:10.139 2021-04-26 21:45:10.139 10.0.0.1:40276 -> 10.0.0.3:5001 4223 184.2 M
2021-04-26 21:45:10.070 2021-04-26 21:45:10.139 10.0.0.3:5001 -> 10.0.0.1:40276 3967 261830
2021-04-26 21:45:14.607 2021-04-26 21:45:14.607 10.0.0.2:48270 -> 10.0.0.4:5001 1 74
2021-04-26 21:45:14.607 2021-04-26 21:45:14.645 10.0.0.4:5001 -> 10.0.0.2:48270 1866 123164
2021-04-26 21:45:10.145 2021-04-26 21:45:14.649 10.0.0.1:40276 -> 10.0.0.3:5001 84071 3.7 G
2021-04-26 21:45:10.145 2021-04-26 21:45:14.649 10.0.0.1:40276 -> 10.0.0.3:5001 84071 3.7 G
2021-04-26 21:45:10.145 2021-04-26 21:45:14.649 10.0.0.1:40276 -> 10.0.0.3:5001 84071 3.7 G
2021-04-26 21:45:10.145 2021-04-26 21:45:14.649 10.0.0.1:40276 -> 10.0.0.3:5001 84071 3.7 G
2021-04-26 21:45:10.145 2021-04-26 21:45:14.649 10.0.0.1:40276 -> 10.0.0.3:5001 84072 3.7 G
2021-04-26 21:45:14.645 2021-04-26 21:45:14.645 10.0.0.2:48270 -> 10.0.0.4:5001 1916 81.2 M
2021-04-26 21:45:10.145 2021-04-26 21:45:14.649 10.0.0.3:5001 -> 10.0.0.1:40276 470139 31.0 M
2021-04-26 21:45:14.655 2021-04-26 21:45:20.080 10.0.0.1:40276 -> 10.0.0.3:5001 71955 4.1 G
2021-04-26 21:45:14.655 2021-04-26 21:45:20.080 10.0.0.1:40276 -> 10.0.0.3:5001 71955 4.1 G
2021-04-26 21:45:14.655 2021-04-26 21:45:20.080 10.0.0.1:40276 -> 10.0.0.3:5001 71956 4.1 G
2021-04-26 21:45:14.655 2021-04-26 21:45:20.080 10.0.0.1:40276 -> 10.0.0.3:5001 71956 4.1 G
2021-04-26 21:45:14.655 2021-04-26 21:45:20.080 10.0.0.3:5001 -> 10.0.0.1:40276 67506 4.5 M
2021-04-26 21:45:14.655 2021-04-26 21:45:24.612 10.0.0.2:48270 -> 10.0.0.4:5001 88583 3.9 G
2021-04-26 21:45:14.655 2021-04-26 21:45:24.612 10.0.0.2:48270 -> 10.0.0.4:5001 88583 3.9 G
2021-04-26 21:45:14.655 2021-04-26 21:45:24.612 10.0.0.2:48270 -> 10.0.0.4:5001 88583 3.9 G
2021-04-26 21:45:14.655 2021-04-26 21:45:24.612 10.0.0.2:48270 -> 10.0.0.4:5001 88583 3.9 G
2021-04-26 21:45:14.655 2021-04-26 21:45:24.612 10.0.0.2:48270 -> 10.0.0.4:5001 88583 3.9 G
2021-04-26 21:45:14.655 2021-04-26 21:45:24.612 10.0.0.2:48270 -> 10.0.0.4:5001 88584 3.9 G
2021-04-26 21:45:14.655 2021-04-26 21:45:24.612 10.0.0.2:48270 -> 10.0.0.4:5001 88584 3.9 G
2021-04-26 21:45:14.654 2021-04-26 21:45:24.612 10.0.0.4:5001 -> 10.0.0.2:48270 762462 50.3 M
Summary: total flows: 28, total bytes: 74.0 G, total packets: 2.9 M, avg bps: 40.7 G, avg pps: 199530, avg bpp: 25507
Time window: 2021-04-26 21:45:10 - 2021-04-26 21:45:24
Total flows processed: 28, Blocks skipped: 0, Bytes read: 2368
Sys: 0.003s flows/second: 8303.7 Wall: 0.006s flows/second: 4569.2
root@d1:~/nfdump/nfcapd_files#

```

Figure 16. Reading a file in customized format.

The following command was executed in the script.

```
nfdump -r /nfcapd.202104262145 -o "fmt:%ts %te %sap -> %dap %pkt %byt"
```

You will see all the available options in the nfdump manual (Reference 4). Followings are the ones used in the script.

- %ts: Start time – First seen
- %te: End time – last seen
- %sap: Source address:port
- %dap: Destination address:port
- %pkt: Packets
- %byt: Bytes

Step 4. Type the following command to aggregate the data.

```
nfdump -r nfcapd.202104262145 -a
```

```

root@d1:~/nfdump/nfcapd_files# nfdump -r nfcapd.202104262145 -a
Date first seen      Duration Proto      Src IP Addr:Port      Dst IP Addr:Port      Packets      Bytes      Flows
2021-04-26 21:45:10.070 10.010 TCP      10.0.0.3:5001 -> 10.0.0.1:40276 541612 35.7 M 3
2021-04-26 21:45:10.070 10.010 TCP      10.0.0.1:40276 -> 10.0.0.3:5001 796473 38.8 G 12
2021-04-26 21:45:14.607 10.005 TCP      10.0.0.2:48270 -> 10.0.0.4:5001 799166 35.1 G 11
2021-04-26 21:45:14.607 10.005 TCP      10.0.0.4:5001 -> 10.0.0.2:48270 764328 50.4 M 2
Summary: total flows: 28, total bytes: 74.0 G, total packets: 2.9 M, avg bps: 40.7 G, avg pps: 199530, avg bpp: 25507
Time window: 2021-04-26 21:45:10 - 2021-04-26 21:45:24
Total flows processed: 28, Blocks skipped: 0, Bytes read: 2368
Sys: 0.004s flows/second: 6474.0 Wall: 0.000s flows/second: 49122.8
root@d1:~/nfdump/nfcapd_files#

```


Figure 17. Aggregating data from a file.

Consider the first entry of the figure above. Total 3 flows are detected where source and destination IP addresses are 10.0.0.3 and 10.0.0.1, respectively. Total number of packets are 541612, total bytes 35.7 M.

3.2 Nfdump filtering features

Step 1. Type the following command to show top five flows from the selected file.

```
nfdump -r nfcapd.202104262145 -c 5
```

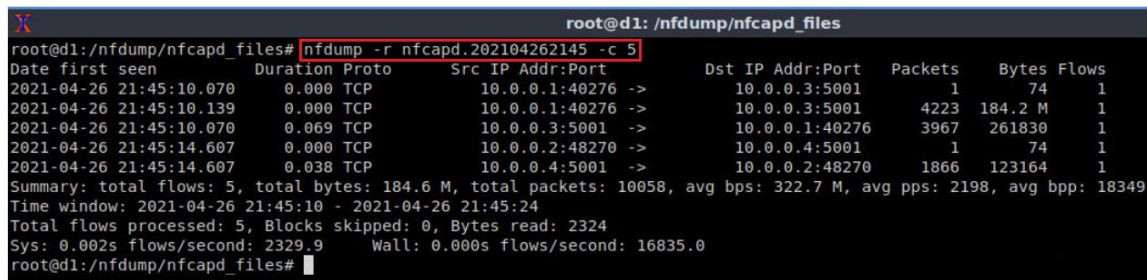


Figure 18. Filtering top five flows from a file.

Consider the figure above. The figure shows top five flows from the file.

Step 2. Type the following command to filter output based on the protocol.

```
nfdump -r nfcapd.202104262145 'proto tcp'
```

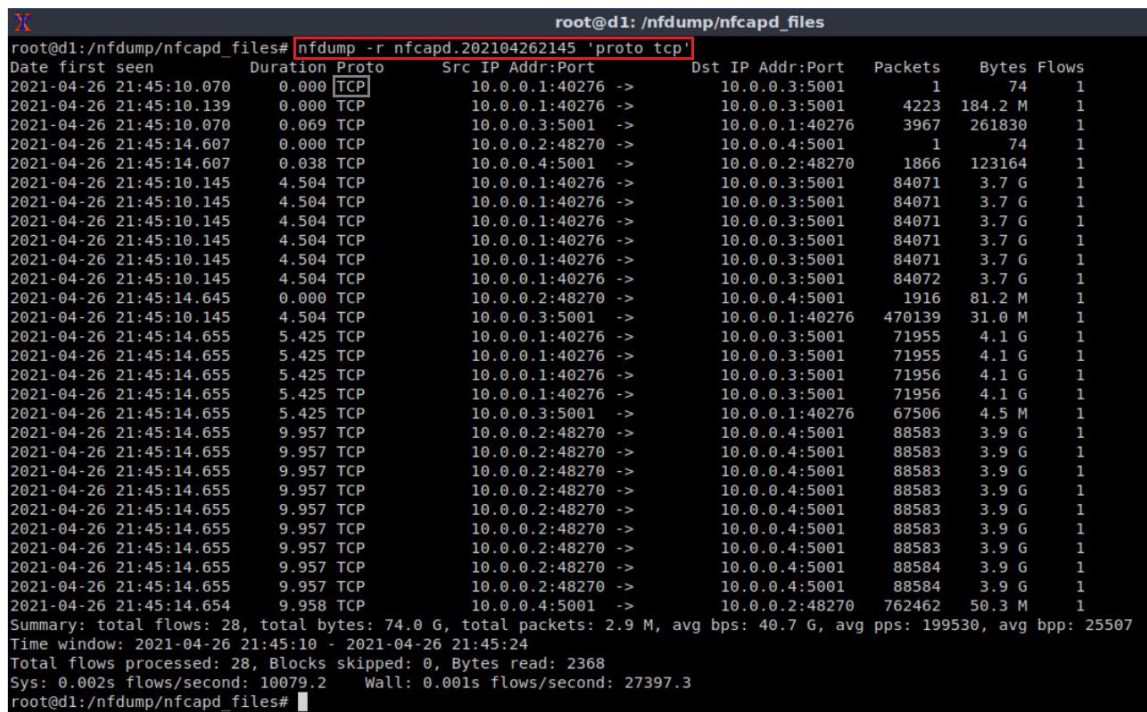


Figure 19. Filtering NetFlow data based on TCP protocol.

Consider the figure above. The figure shows the output for TCP protocol.

Step 3. Type the following command to filter output ordered by bytes.

```
nfdump -r nfcapd.202104262145 -O bytes
```

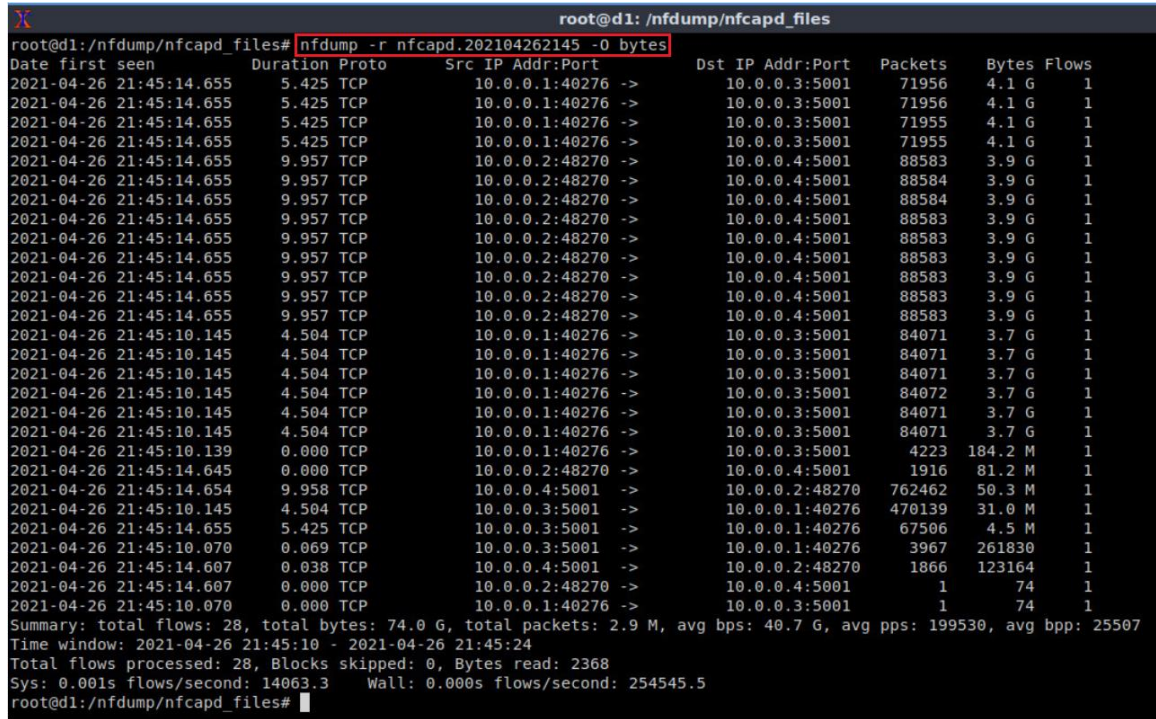


Figure 20. Filtering NetFlow data ordered by bytes.

Consider the figure above. `-O` is referring to order and the figure shows that the bytes field is sorted by ascending order.

Step 4. Type the following command to filter output by source IP address.

```
nfdump -r nfcapd.202104262145 'src ip 10.0.0.1'
```

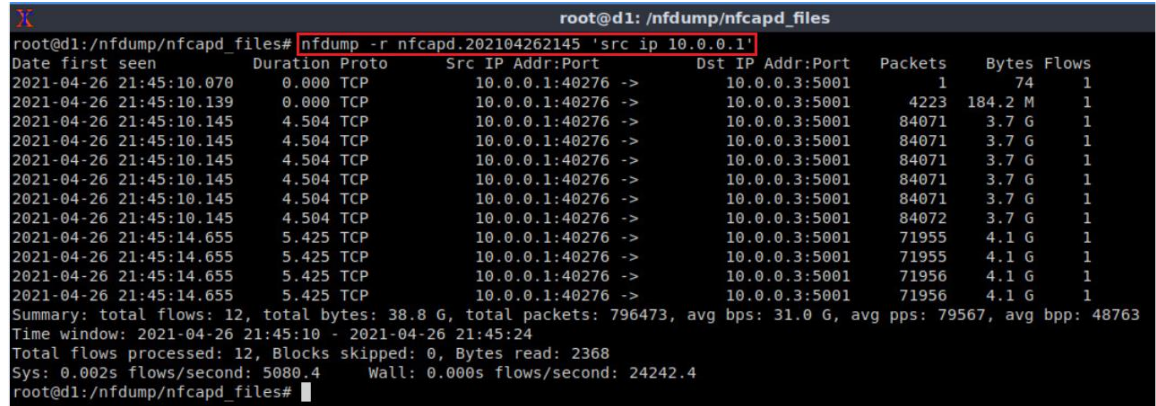


Figure 21. Filtering NetFlow data based on IP address.

Consider the figure above. The figure shows all the flows containing source IP address 10.0.0.1.

Step 5. Type the following command to filter output by network address.

```
nfdump -r nfcapd.202104262145 'net 10.0.0.0/8'
```

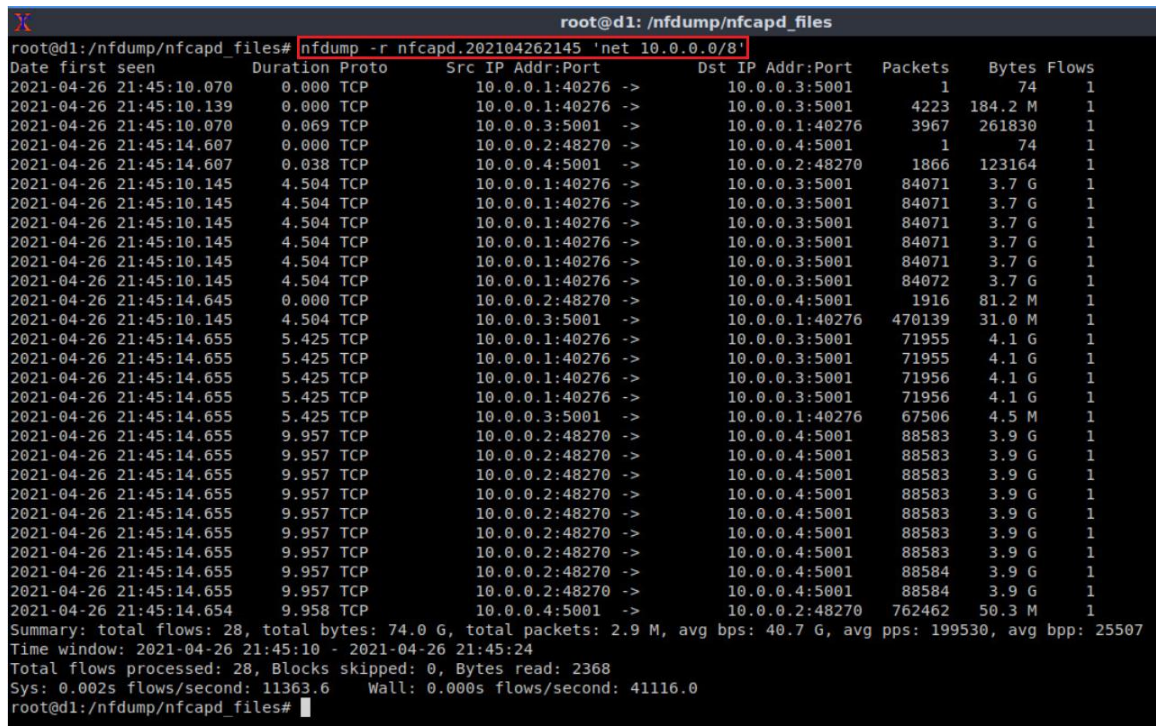


Figure 22. Filtering NetFlow data based on network address.

Consider the figure above. The figure shows all the flows belong to the network 10.0.0.0/8.

Step 6. Nfdump provides a number of statistics. Type the following command to show the source IP statistics.

```
nfdump -r nfcapd.202104262145 -s srcip
```

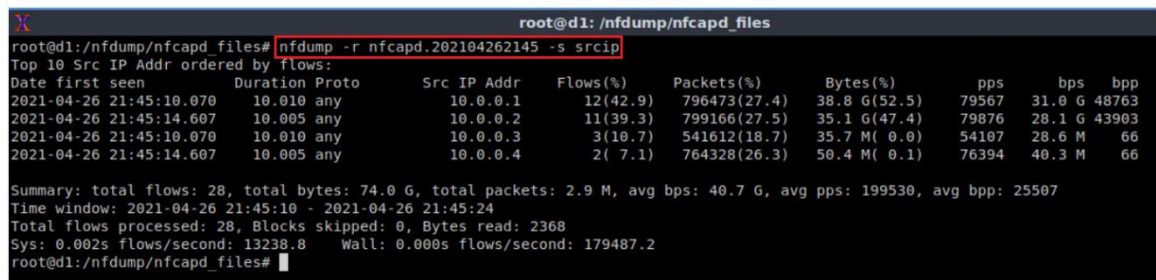


Figure 23. Displaying top source IP address statistics.

Consider the figure above. The figure shows top source IP addresses ordered by flows. It includes the percentage for flows, packets and bytes used by the source IP addresses.

Step 7. You can also use multiple nfdump filters at the same time. Type the following command to filter output by multiple expressions.

```
nfdump -r nfcapd.202104262145 -o long 'net 10.0.0.0/8 and not host 10.0.0.1 and packets > 70000'
```


This concludes Lab 6. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

References

1. Cisco, "*Introduction to Cisco IOS NetFlow – A technical overview*", [Online]. https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html
2. B. Claise, "*RFC 3954: Cisco systems NetFlow services export version 9*", (2004). <https://tools.ietf.org/html/rfc3954>
3. DNSstuff, "*Best NetFlow Analyzer and Collectors*", [Online]. <https://www.dnsstuff.com/netflow-analyzer-software#best-netflow-analyzers-and-collectors-list>
4. Sourceforge, "*Nfdump*", [Online]. <http://nfdump.sourceforge.net/>
5. Omar Santos, "*Network security with NetFlow and IPFIX*", (2016).
6. Plixer, "*Open vSwitch NetFlow*", [Online]. <https://www.plixer.com/blog/open-vswitch-netflow/>



UNIVERSITY OF
SOUTH CAROLINA

NETWORK MANAGEMENT

Lab 7: Collecting and Visualizing sFlow data using GoFlow and Grafana

Document Version: **07-08-2022**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Introduction	3
1.1 Introduction to GoFlow and Grafana	4
2 Lab topology.....	4
2.1 Lab settings.....	5
2.2 Loading a topology	5
3 Launching sFlow collector and agent.....	7
4 Analyzing sFlow sampling records using Grafana.....	11
4.1 Launching Grafana	11
4.2 Creating dashboard in Grafana	13
4.3 Performing a connectivity test.....	17
4.4 Exploring Grafana dashboard.....	19
4.5 Creating an alert in Grafana	29
References	32

Overview

This lab introduces Grafana which is a multi-platform open-source analytics and interactive visualization web application. The focus of this lab is to enable sFlow agent in Open Virtual Switch (Open vSwitch) and analyze the collected flows using Grafana dashboard.

Objectives

By the end of this lab, you should be able to:

1. Understand the concept of GoFlow and Grafana.
2. Enable sFlow in Open vSwitch.
3. Analyze sFlow sampling records using Grafana.

Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Launching sFlow collector and agent.
4. Section 4: Analyzing sFlow sampling records using Grafana.

1 Introduction

Data visualization transforms abstract data into a visual context, such as a charts, plots, or graph, to make data easier for the human brain to understand. We can visualize large volumes of data in an understandable and coherent way, which in turn helps us comprehend the information and draw conclusions and insights¹.

Before Graphical User Interface (GUI) systems became popular, command line interface (CLI) systems were the norm. On these systems, users had to input commands using lines

of coded text. The commands ranged from simple instructions for accessing files or directories to far more complicated commands that required many lines of code. Apparently, GUI systems have made computers far more user-friendly than CLI systems².

1.1 Introduction to GoFlow and Grafana

Grafana is an open and composable observability and data visualization platform. The purpose of Grafana dashboards is to bring data together in a way that is both efficient and organized. It allows users to better understand the metrics of their data through queries, informative visualizations and alerts³. Users can also share the dashboards you create with other team members, allowing you to explore the data together.

A Grafana dashboard is a powerful open source analytical and visualization tool that consists of multiple individual panels arranged in a grid. The panels interact with configured data sources, including Microsoft SQL server, Prometheus, MySQL, InfluxDB, PostgreSQL and many others. Each panel is connected to a data source. Since the Grafana dashboards support multiple panels in a single grid, users can visualize results from multiple data sources simultaneously.

GoFlow is an application by Cloudflare to collect Netflow/IPFIX/sFlow data. Flow-pipeline is a repository which contains GoFlow collector, kafka (provides a framework for storing, reading, and analyzing streaming data), a database and an inserter (to insert the flows in a database)⁴.

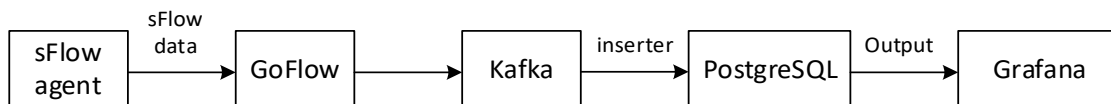


Figure 1. GoFlow and Grafana integration.

Consider Figure 1. GoFlow collector will collect sFlow data from the sFlow agent. The data is sent to Kafka. Apache Kafka is a distributed data store optimized for ingesting and processing streaming data in real-time. Streaming data is the data that is continuously generated by thousands of data sources, which typically send the data records simultaneously. A streaming platform needs to handle this constant influx of data and process the data sequentially and incrementally⁵. An inserter is responsible for inserting the flows in the PostgreSQL database from kafka. Grafana is connected to the database to visualize the data based on the user requirements⁴.

2 Lab topology

Consider Figure 2. There are three switches and two end hosts. Switch s3 is acting as a sFlow agent. Localhost 127.0.0.1 (loopback address of the Virtual Machine) is the collector. The collector is not visible in Mininet topology since it is a part of the operating system.

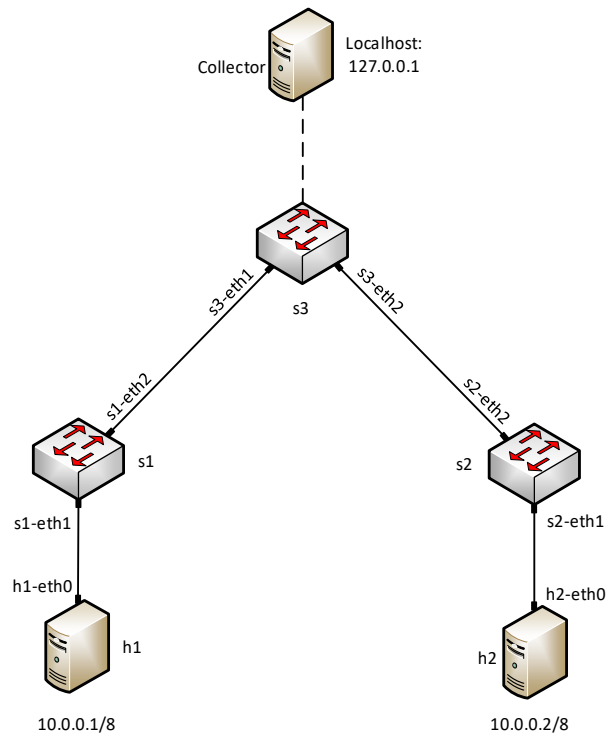


Figure 2. Lab topology.

2.1 Lab settings

The devices should be configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8

2.2 Loading a topology

Step 1. Click on the Client tab to access the Client PC.

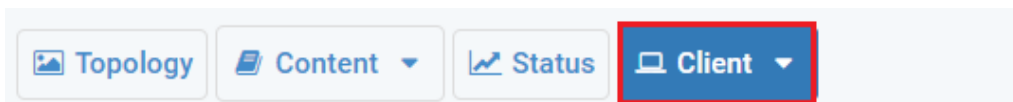


Figure 3. Accessing the Client PC.

Step 2. Start by launching MiniEdit by clicking on desktop's shortcut. When prompted for a password, type `password`.

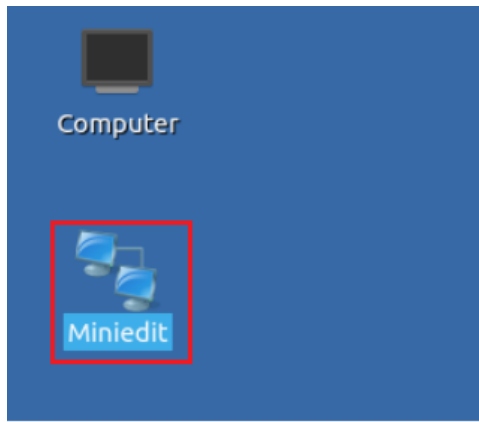


Figure 4. MiniEdit shortcut.

Step 3. On MiniEdit's menu bar, click on *File* then open to load the lab's topology. Open the directory called *lab7* and select the file *lab7.mn*. Then, click on *Open* to open the topology.

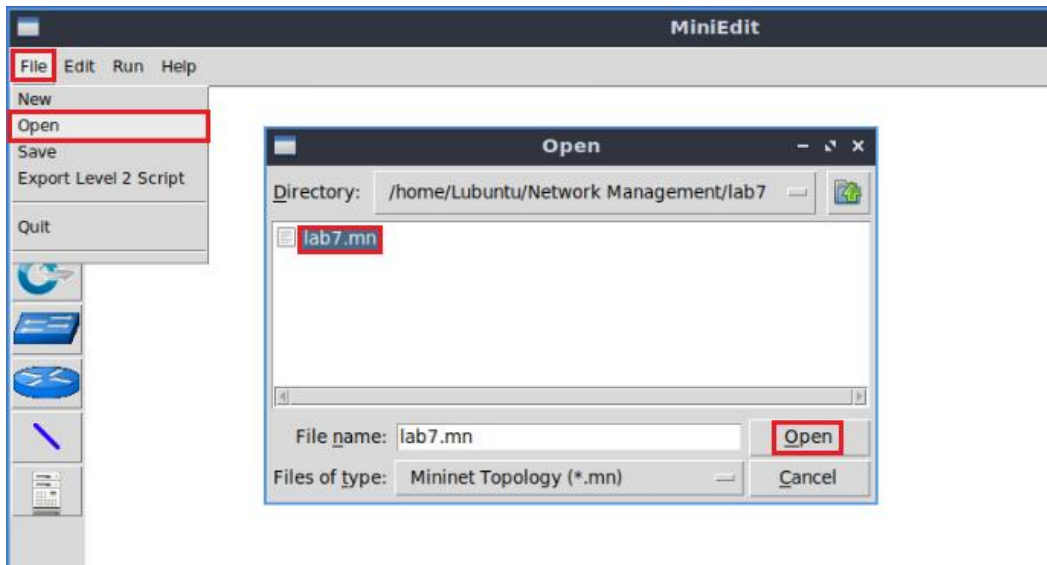


Figure 5. MiniEdit's Open dialog.

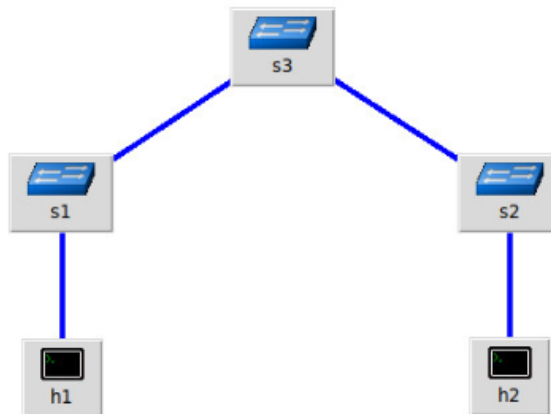


Figure 6. MiniEdit's topology.

Step 4. To proceed with the emulation, click on the *Run* button located on the lower left-hand side.

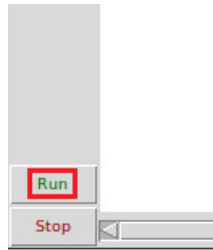


Figure 7. Starting the emulation.

Step 5. Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

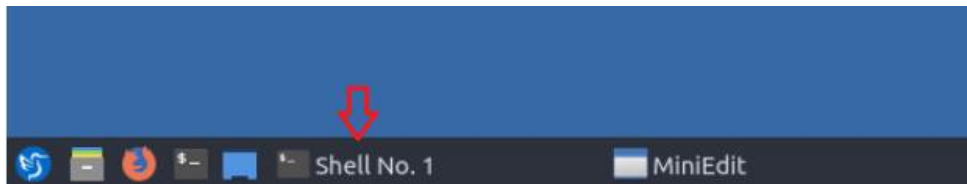


Figure 8. Opening Mininet's terminal.

Step 6. Issue the following command to display the interface names and connections.

```
links
```

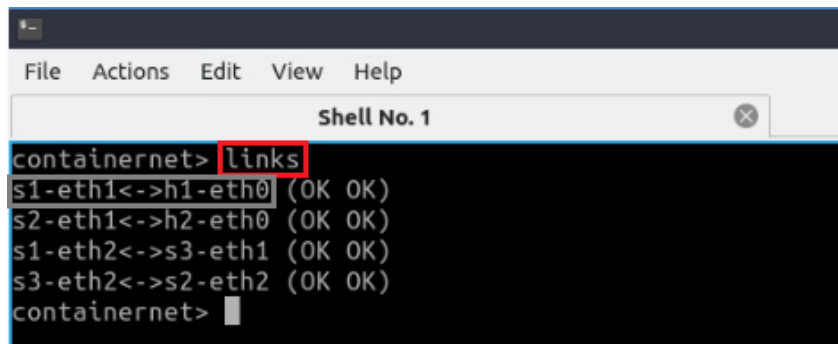


Figure 9. Displaying network interfaces.

In figure 9, the link displayed within the gray box indicates that interface eth1 of host s1 connects to interface eth0 of switch h1 (i.e., *s1-eth1<-> h1-eth0*).

3 Launching sFlow collector and agent

Step 1. Open the Linux terminal.



Figure 10. Opening Linux terminal.

Step 2. Type the following command to assign a valid IP address to all interfaces. If prompted for password, type `password`.

```
sudo dhclient
```

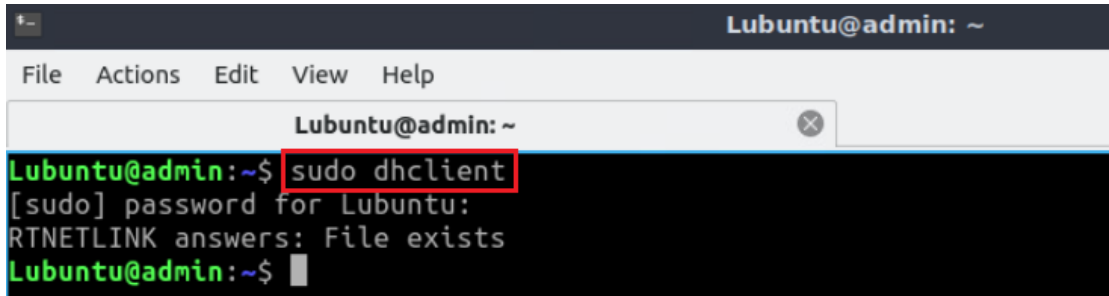


Figure 11. Assigning valid IP addresses to all interfaces.

Step 3. Navigate into `flow-pipeline/compose` directory by issuing the following command. The folder contains a startup file that includes GoFlow, Kafka, PostgreSQL, and an inserter.

```
cd flow-pipeline/compose/
```

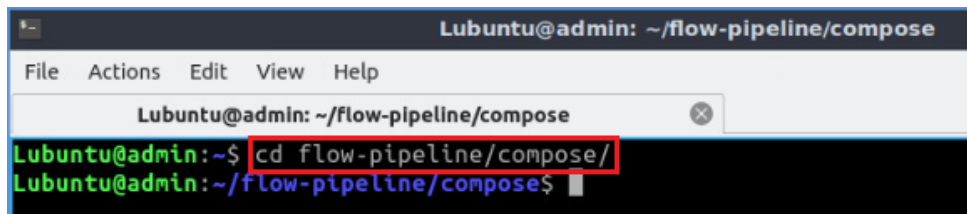


Figure 12. Navigating into flow-pipeline/compose directory.

Step 4. Type the following command to run the startup file.

```
sudo docker-compose -f docker-compose-postgres-collect.yml up
```

```

Lubuntu@admin: ~/flow-pipeline/compose
Lubuntu@admin:~/flow-pipeline/compose$ sudo docker-compose -f docker-compose-postgres-collect.yml up
Pulling goflow (cloudflare/goflow:latest)...
latest: Pulling from cloudflare/goflow
c9b1b535fdd9: Pull complete
13ecdff041c73: Pull complete
f778a00b494eb: Pull complete
Digest: sha256:dc78fad655a60d2a46ff772d3db38d6d8af4817c2244b1671ac4fe7e0302b6f
Status: Downloaded newer image for cloudflare/goflow:latest
Pulling postgres (postgres:latest)...
latest: Pulling from library/postgres
7d63c13d9b9b: Extracting [====>] 2.949MB/31.36MB
cad0f9d5f5fe: Download complete
ff74a7a559cb: Download complete
c43dfd845683: Download complete
kafka_1 | [2021-11-01 17:40:16,456] INFO [GroupMetadataManager brokerId=1001] Finished loading offsets and group metadata from __consu
insserter_1 | time="2021-11-01T17:40:20Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:40:25Z" level=info msg="Processed 0 records in the last iteration."
prometheus_1 | level=info ts=2021-11-01T17:40:26.062Z caller=compact.go:509 component=tsdb msg="write block resulted in empty block" mint=1
635717600000 maxt=1635724800000 duration=22.041249ms
prometheus_1 | level=info ts=2021-11-01T17:40:26.064Z caller=head.go:798 component=tsdb msg="Head GC completed" duration=1.278927ms
prometheus_1 | level=info ts=2021-11-01T17:40:26.064Z caller=checkpoint.go:97 component=tsdb msg="Creating checkpoint" from_segment=5 to_se
gment=7 mint=1635724800000
prometheus_1 | level=info ts=2021-11-01T17:40:26.067Z caller=head.go:967 component=tsdb msg="WAL checkpoint complete" first=5 last=7 durati
on=2.871342ms
goflow_1 | time="2021-11-01T17:40:28Z" level=info msg="Starting GoFlow"
goflow_1 | time="2021-11-01T17:40:28Z" level=info msg="Listening on UDP :2056" Type=NetFlowLegacy
goflow_1 | time="2021-11-01T17:40:28Z" level=info msg="Listening on UDP :6343" Type=sFlow
goflow_1 | time="2021-11-01T17:40:28Z" level=info msg="Listening on UDP :2055" Type=NetFlow
initializer_1 | Exception in thread "main" java.lang.UnrecognizedOptionException: zookeeper is not a recognized option
initializer_1 | at joptsimple.OptionException.unrecognizedOption(OptionException.java:108)
initializer_1 | at joptsimple.OptionParser.handleLongOptionToken(OptionParser.java:510)
initializer_1 | at joptsimple.OptionParserState$2.handleArgument(OptionParserState.java:56)
initializer_1 | at joptsimple.OptionParser.parse(OptionParser.java:396)
initializer_1 | at kafka.admin.TopicCommand$TopicCommandOptions.<init>(TopicCommand.scala:517)
initializer_1 | at kafka.admin.TopicCommand$.main(TopicCommand.scala:47)
initializer_1 | at kafka.admin.TopicCommand.main(TopicCommand.scala)
compose_initializer_1 exited with code 1
insserter_1 | time="2021-11-01T17:40:30Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:40:35Z" level=info msg="Processed 0 records in the last iteration."
    
```

Figure 13. Enabling the collector.

Consider the figure above. It may take some time to start the collector. The highlighted part shows that the collector *GoFlow* is starting. UDP port 6343 is for sFlow, 2056 is for Netflow and port 2055 is for IPFIX. Once the collector is running, you will see the number of records processed by the inserter.

Once you run a test between hosts h1 and h2, you will notice number of records increasing.

Step 5. Click on the *File* option and select *+ New Tab* or press `Ctrl+Shift+T`.

```

Lubuntu@admin: ~/flow-pipeline/compose
File Actions Edit View Help
+ New Tab Ctrl+Shift+T
New Tab From Preset
Close Tab Ctrl+Shift+W
New Window Ctrl+Shift+N
Preferences...
Quit
insserter_1 | time="2021-11-01T17:49:20Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:49:25Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:49:30Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:49:35Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:49:40Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:49:45Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:49:50Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:49:55Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:50:00Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:50:05Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:50:10Z" level=info msg="Processed 0 records in the last iteration."
insserter_1 | time="2021-11-01T17:50:15Z" level=info msg="Processed 0 records in the last iteration."
    
```

Figure 14. Opening a new terminal tab.

Step 6. Type the following command to enable a sFlow agent. If prompted for password, type `password`.

```

sudo ovs-vsctl -- --id=@s create sflow agent=s3-eth1 target="\127.0.0.1:6343\"
sampling=64 -- set bridge s3 sflow=@s
    
```



```

Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~/flow-pipeline/compose
Lubuntu@admin: ~
Lubuntu@admin:~$ sudo ovs-vsctl -- --id=@s create sflow agent=s3-eth1 target=["127.0.0.1:6343"] sampling=64 -- set bridge s3 sflow=@s
[sudo] password for Lubuntu:
e07da18a-f9b7-40b8-ade7-603c12fd1a82
Lubuntu@admin:~$
    
```

Figure 15. Enabling sFlow agent.

Consider the figure above. The command creates a sFlow ID and is attached to switch s3. Switch s3 is acting as an agent and transmits data to the collector. 127.0.0.1 is the collector IP and the port is 6343. Sampling rate is 64 which means 1 in 64 packets will be sampled.

Step 7. Type the following command to verify sFlow configuration.

```

sudo ovs-vsctl list sflow
    
```

```

Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~/flow-pipeline/compose
Lubuntu@admin:~$ sudo ovs-vsctl list sflow
_uuid          : e07da18a-f9b7-40b8-ade7-603c12fd1a82
agent          : s3-eth1
external_ids   : {}
header         : []
polling        : []
sampling       : 64
targets        : ["127.0.0.1:6343"]
Lubuntu@admin:~$
    
```

Figure 16. Verifying sFlow configuration.

Consider the figure above. One sFlow agent is running, target collector IP is 127.0.0.1 and the port is 6343.

Step 8. Follow step 5 to open another terminal.

Step 9. Navigate into *flow-pipeline/compose/postgres* directory by issuing the following command.

```

cd flow-pipeline/compose/postgres/
    
```

```

Lubuntu@admin: ~
File Actions Edit View Help
Lubuntu@admin: ~/flow-pipeline/compose
Lubuntu@admin: ~
Lubuntu@admin:~$ cd flow-pipeline/compose/postgres/
Lubuntu@admin:~/flow-pipeline/compose/postgres$
    
```

Figure 17. Navigating into *flow-pipeline/compose/postgres* directory.

Step 10. Type the following command to show the content of the script *create.sh* located inside the directory.

```

Lubuntu@admin: ~/flow-pipeline/compose/postgres
File Actions Edit View Help
Lubuntu@admin: ~/flow-pipeline/compose
Lubuntu@admin: ~
Lubuntu@admin:~/flow-pipeline/compose/postgres$ cat create.sh
#!/bin/bash
set -e

psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_DB" <<-EOSQL
CREATE TABLE IF NOT EXISTS flows (
    id bigserial PRIMARY KEY,
    date_inserted timestamp default NULL,

    time_flow timestamp default NULL,
    type integer,
    sampling_rate integer,
    src_as bigint,
    dst_as bigint,
    src_ip inet,
    dst_ip inet,

    bytes bigint,
    packets bigint,

    etype integer,
    proto integer,
    src_port integer,
    dst_port integer
);
EOSQL
Lubuntu@admin:~/flow-pipeline/compose/postgres$

```

Figure 18. Displaying a file located inside the directory.

Consider the figure above. If any data is stored in the postgres database, a table *flows* will be created. The table has different columns such as `src ip`, `dst ip`, bytes and packets. You will explore more about the database when you analyze records in Grafana dashboard.

4 Analyzing sFlow sampling records using Grafana

In this section, you will analyze sFlow data. You will use graphical interface of Grafana to view the results. You will create a folder and a dashboard which can be saved and used later.

4.1 Launching Grafana

Step 1. Open the browser.

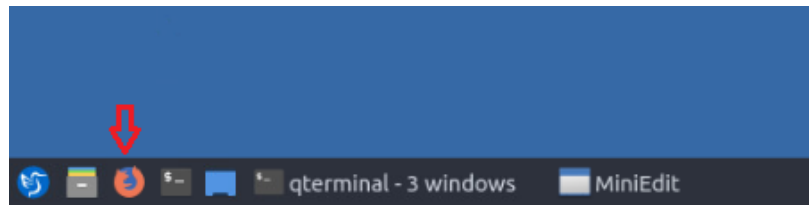


Figure 19. Opening a browser.

Step 2. In the search engine, type `localhost:3000/login`. The username is `admin` and the password is `admin`. The, click on *Login*.

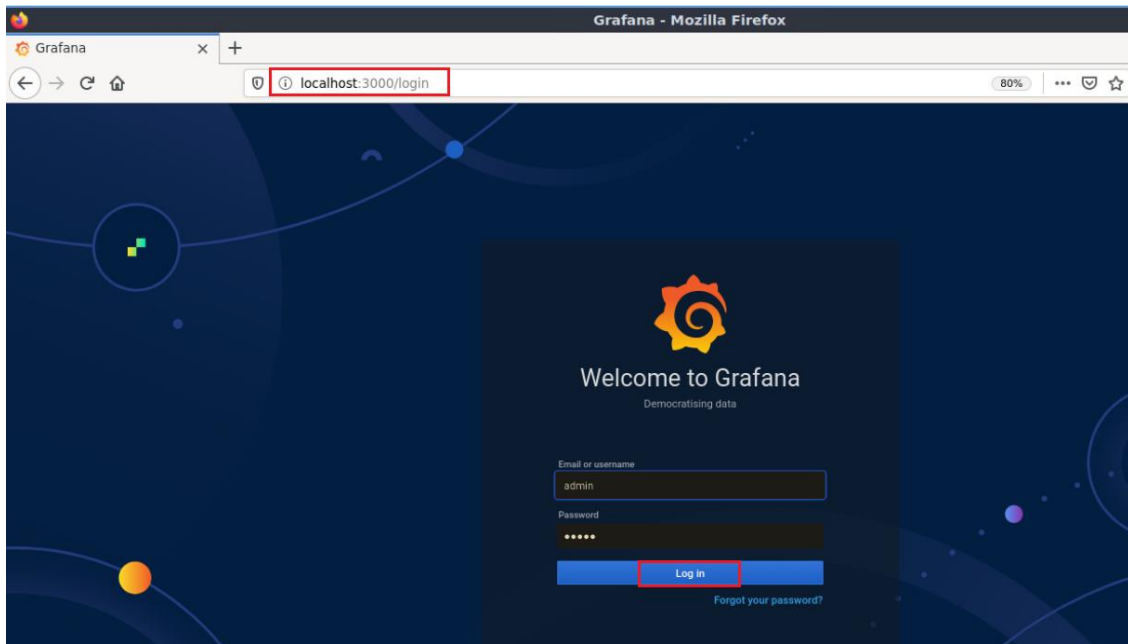


Figure 20. Login to Grafana.

Step 3. There is an option to change the password. You can also click on *skip* if you do not want to change the password.

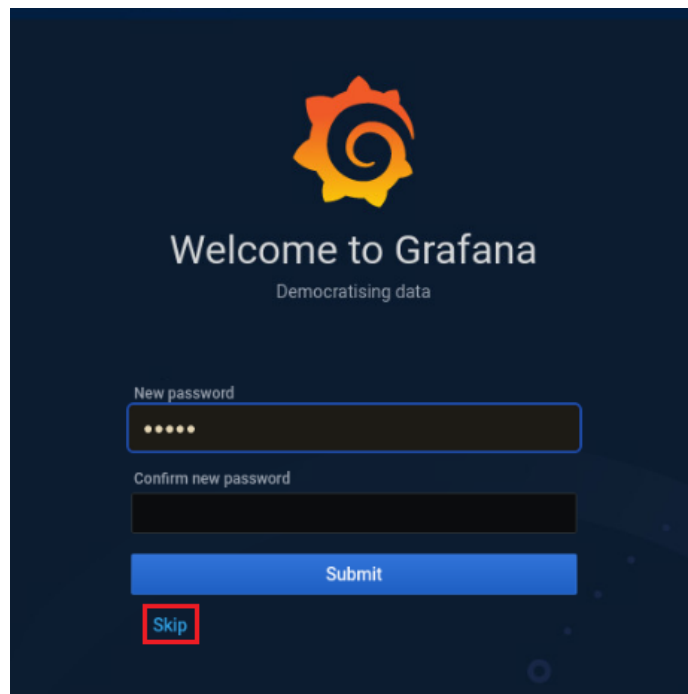


Figure 21. Login to Grafana.

Step 4. Once you login, you will be directed to the homepage of Grafana.

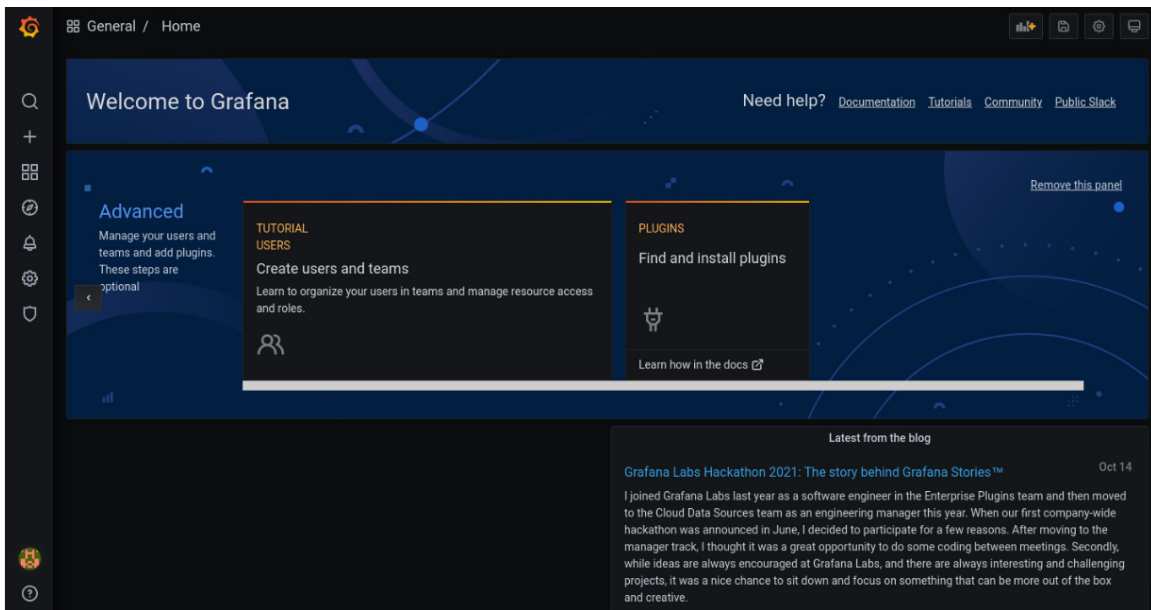



Figure 22. Visualizing Grafana homepage.

4.2 Creating dashboard in Grafana

In this section, you will create a Grafana dashboard.

Step 1. In this step, you will create a folder for sFlow. You will be able to add dashboards in that particular folder. Click on the  sign located on the left side. Select Create -> Folder.

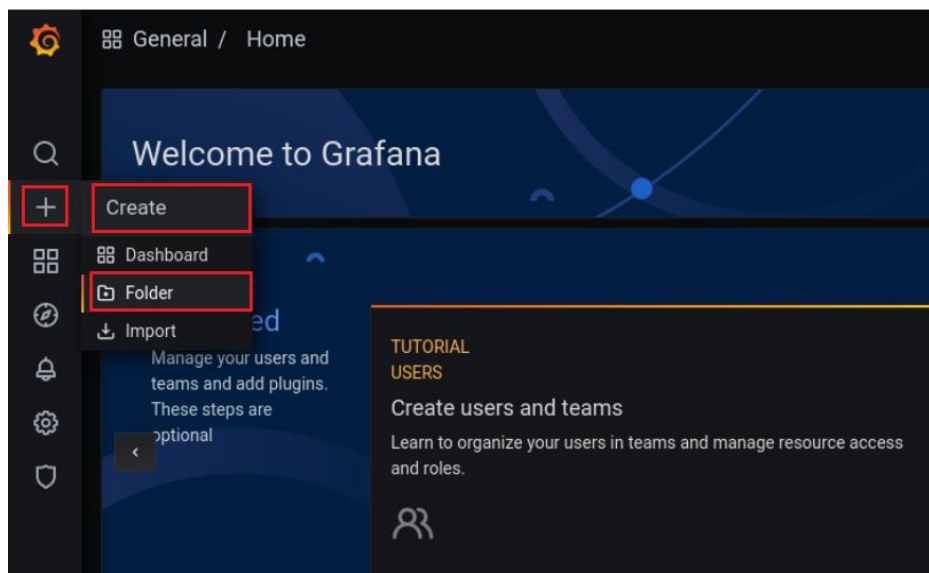


Figure 23. Creating a folder.

Step 2. Name the folder as *sFlow* and select *Create*.

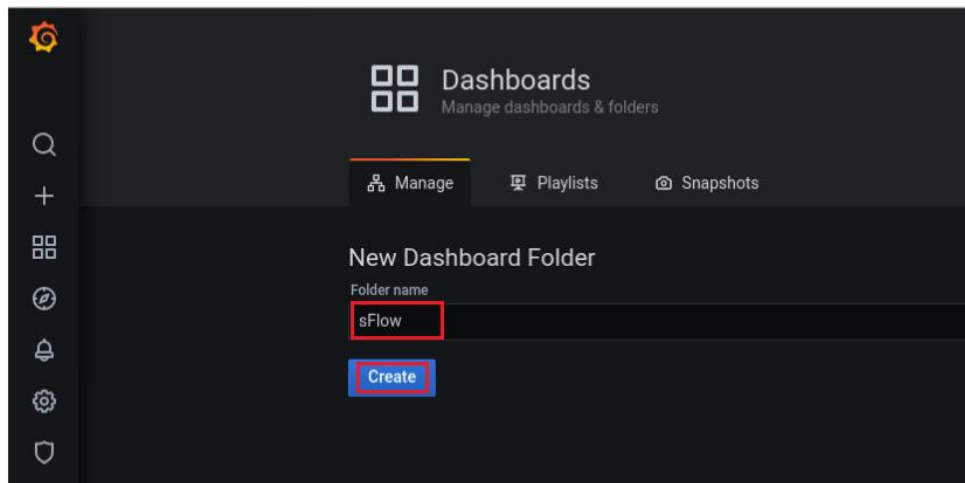


Figure 24. Creating a folder.

Step 3. Once you create a folder, you will see an option to create dashboard. Click on *Create Dashboard*. It will create a panel. You can create multiple panels to make a dashboard.

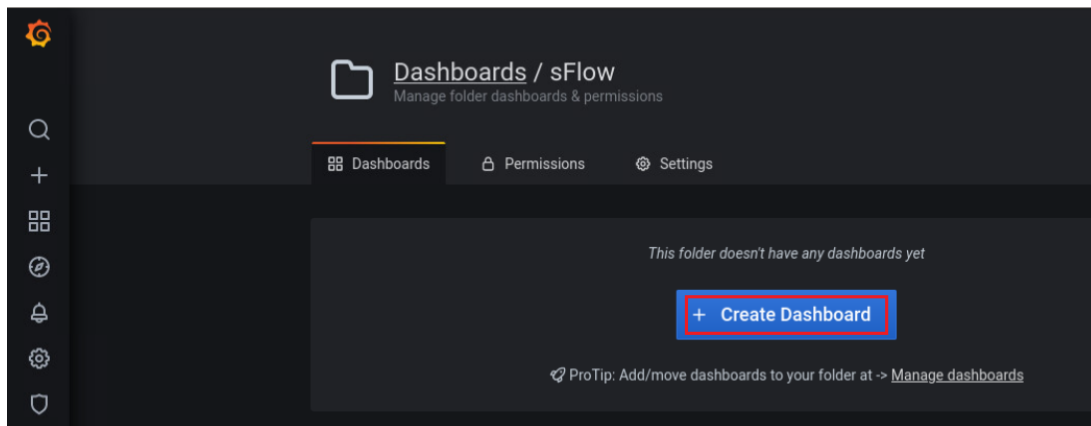


Figure 25. Creating a dashboard.

Step 4. Select *Add an empty panel*.

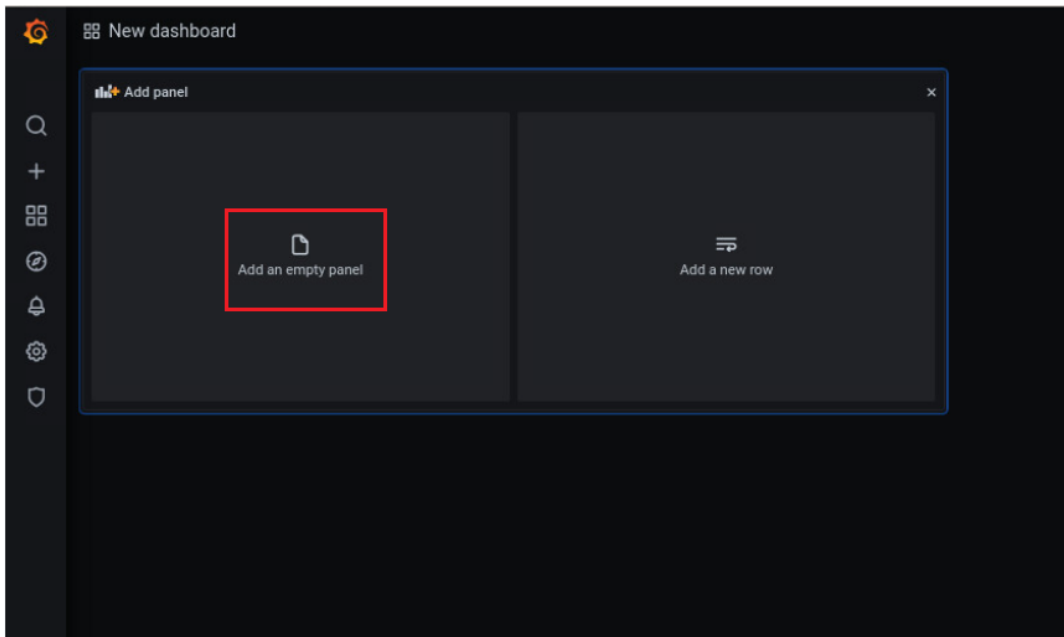


Figure 26. Adding an empty panel into Grafana dashboard.

Step 5. By default, it shows a demo graph. Select the source, *PostgreSQL* from the dropdown box showed in the following figure.

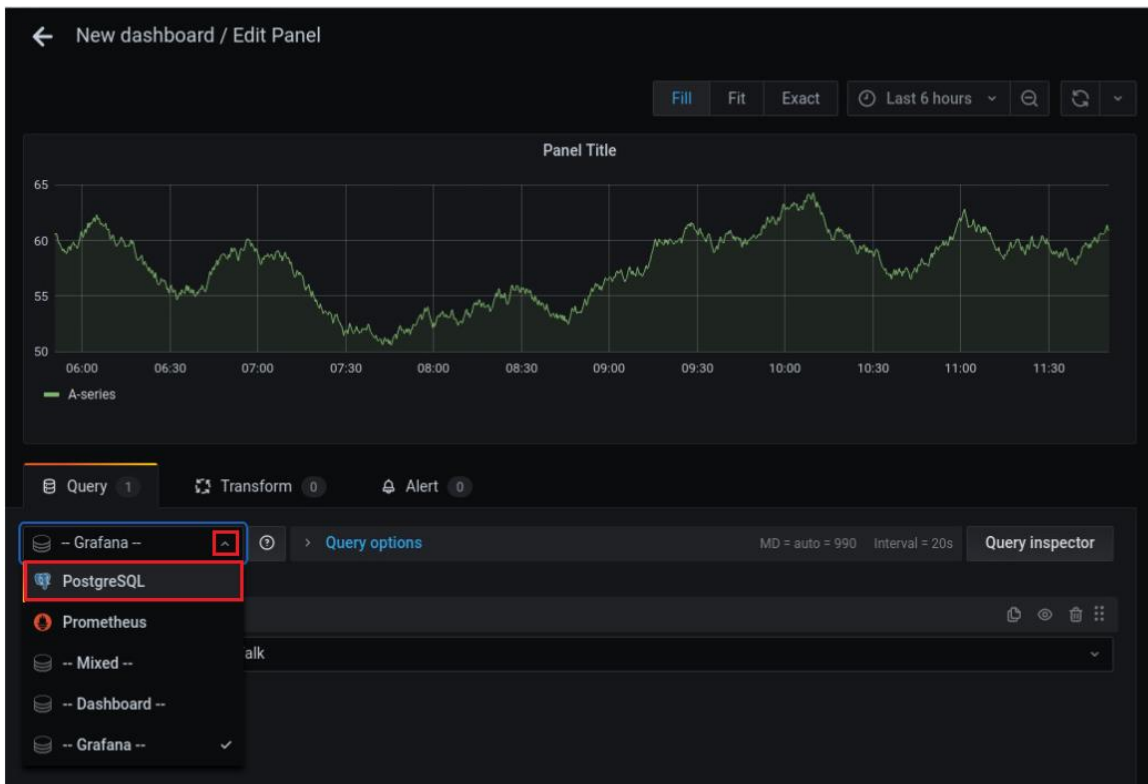


Figure 27. Customizing a panel.

Step 6. Grafana uses queries to communicate with data sources to get data for the visualization. A query is a question written in the query language used by the data source. Data sources have different query languages and syntaxes to ask for the data. Grafana provides a query editor which helps you to write queries. Depending on your data source, the query editor might provide auto-completion, metric names, or variable

suggestion. In this lab, you will write queries manually. Select *Edit SQL* so that you can write queries.

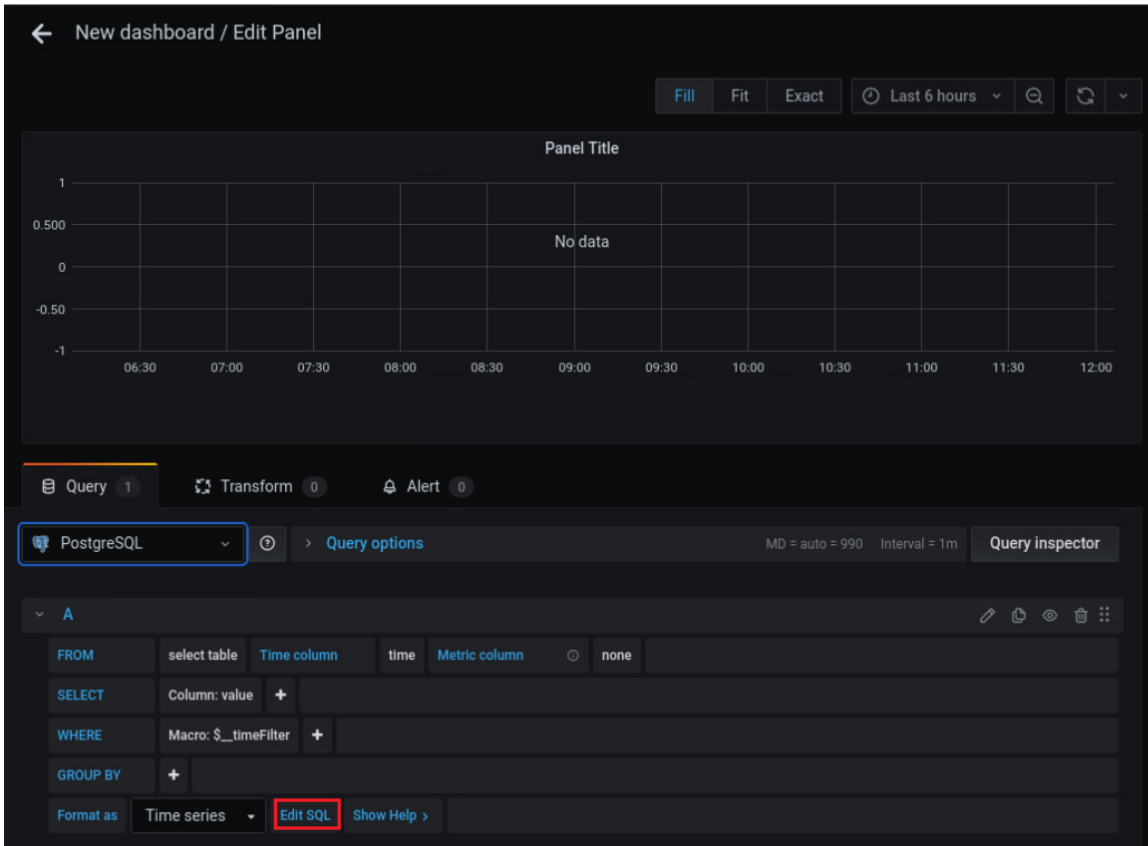


Figure 28. Customizing a panel.

Step 7. Type the query as shown in the following figure to create a graph. The query will extract data from the database, and you will visualize a graph for time vs flows in real-time.

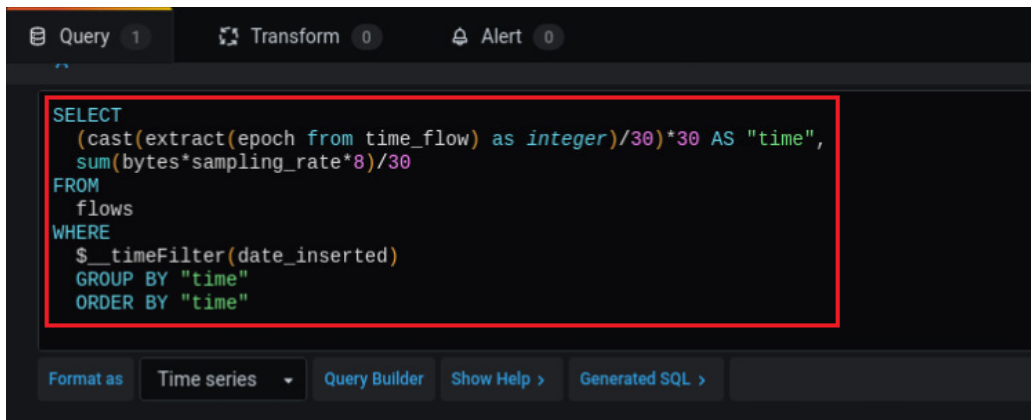


Figure 29. Customizing a panel.

Step 8. Select *Panel* located on the top right of the panel. From the dropdown box of Axes, select *Left Y-> Unit -> Data rate -> bytes/sec(SI)*.

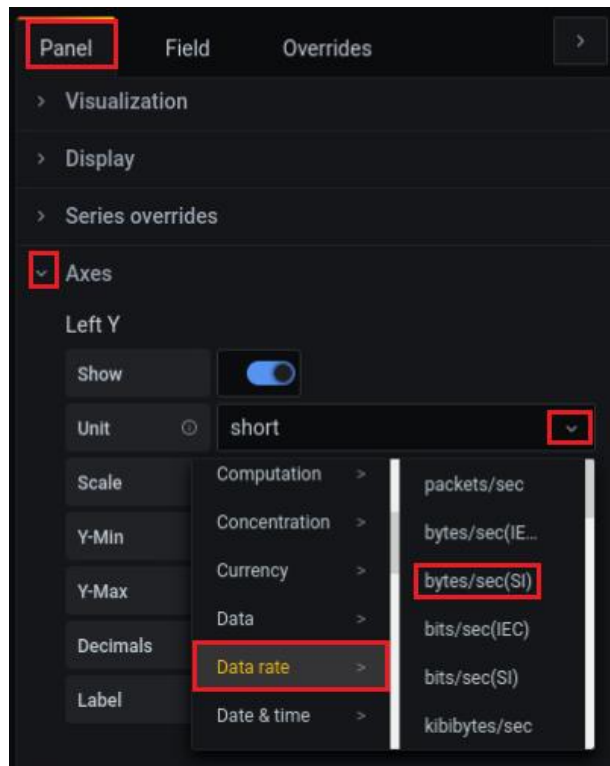


Figure 30. Customizing a panel.

Consider the figure above. This will change the unit for Y-axis.

4.3 Performing a connectivity test

Step 1. Hold right-click on host h2 and select *Terminal*. This opens the terminal of host h2 and allows the execution of commands on that host.

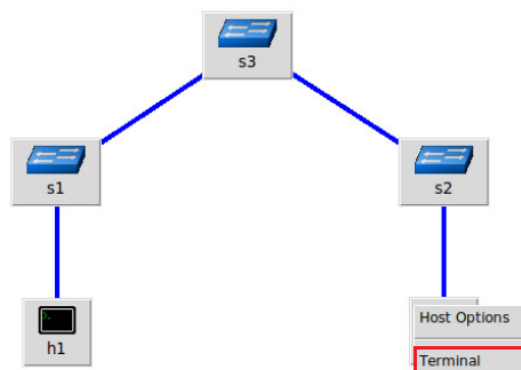


Figure 31. Opening a terminal on host h2.

Step 2. In host h2 terminal, type the following command to run the host in server mode.

```
iperf3 -s
```



```
Host: h2
root@admin:~# iperf3 -s
-----
Server listening on 5201
-----
```

Figure 32. Running host h2 in server mode.

Consider the figure above. The figure shows that host h2 is acting as a server and listening to port 5201.

Step 3. Open host h1 terminal and type the following command to run the host in client mode.

```
iperf3 -c 10.0.0.2 -t 200
```

```
Host: h1
root@admin:~# iperf3 -c 10.0.0.2 -t 200
Connecting to host 10.0.0.2, port 5201
[ 7] local 10.0.0.1 port 55138 connected to 10.0.0.2 port 5201
[ ID] Interval      Transfer    Bitrate    Retr  Cwnd
[ 7]  0.00-1.00    sec  1.74 GBytes 15.0 Gbits/sec  0  8.10 MBytes
[ 7]  1.00-2.00    sec  1.75 GBytes 15.0 Gbits/sec  0  8.10 MBytes
[ 7]  2.00-3.00    sec  1.60 GBytes 13.7 Gbits/sec  0  8.10 MBytes
[ 7]  3.00-4.00    sec  1.40 GBytes 12.1 Gbits/sec  0  8.10 MBytes
[ 7]  4.00-5.01    sec  1.52 GBytes 12.9 Gbits/sec  0  8.10 MBytes
[ 7]  5.01-6.00    sec  1.10 GBytes  9.55 Gbits/sec  0  8.10 MBytes
[ 7]  6.00-7.00    sec  1.21 GBytes 10.4 Gbits/sec  0  8.10 MBytes
[ 7] 199.00-200.00 sec  1.65 GBytes 14.1 Gbits/sec  0  2.85 MBytes
-----
[ ID] Interval      Transfer    Bitrate    Retr  Cwnd
[ 7]  0.00-200.00  sec  316 GBytes 13.6 Gbits/sec 1689
[ 7]  0.00-200.04  sec  316 GBytes 13.6 Gbits/sec
-----
iperf Done.
root@admin:~#
```

Figure 33. Running host h1 in client mode.

Consider the figure above. The test runs for 200 seconds.

Step 4. Go to the flow-pipeline terminal. You will notice number of records increasing.

The figure above shows the traffic for the test running between hosts h1 and h2.

Step 3. In this step, you will save the panel. Go to the *panel* option, select *settings*, and change the panel title to *sFlow traffic*. Click on *apply* to save the panel in the dashboard.

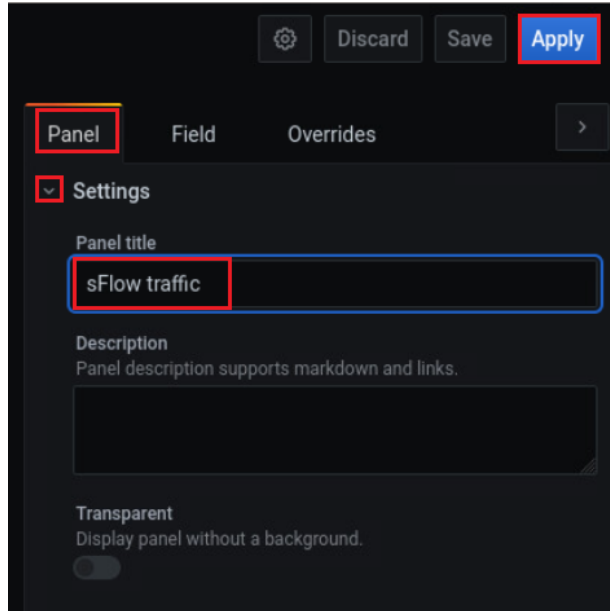


Figure 37. Saving a panel.

Step 4. The dashboard will look like the following figure. Click on the *save* option located on the top right side.



Figure 38. Saving a dashboard.

The time range of the data is set to 15 minutes. You can also change it to any time range (30 minutes, 1 hour) as required. You might get a different graph depending on the flows.

Step 5. Change the dashboard name to *sFlow Dashboard* and click on *save*.

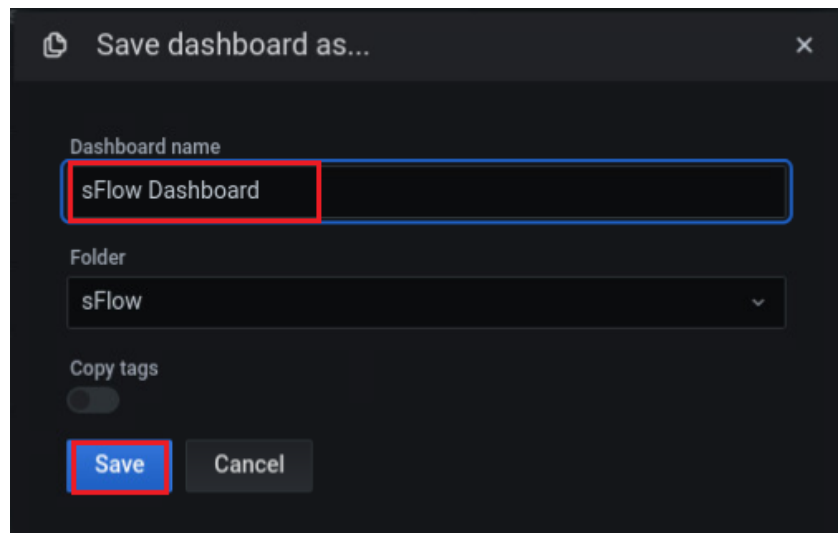


Figure 39. Saving a dashboard.

Step 6. Click on the *add panel* option located on the top right (showed in the following figure).

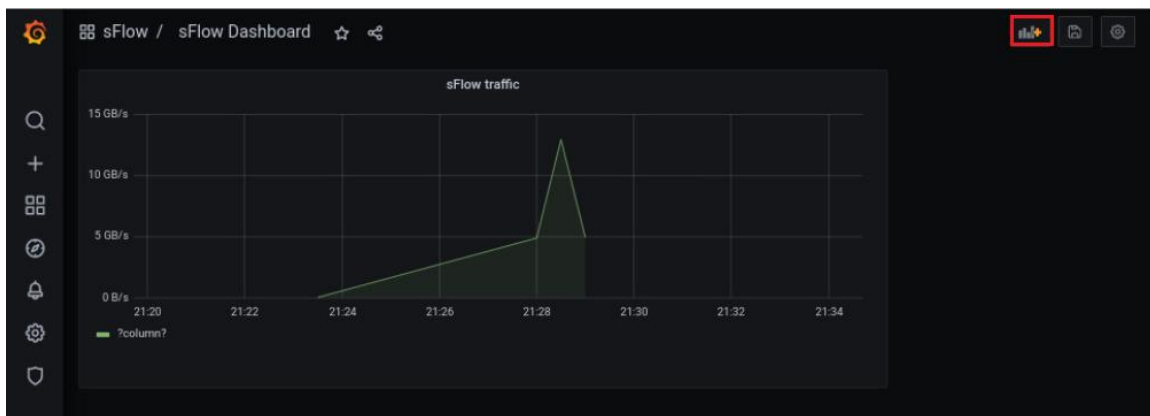


Figure 40. Adding a panel.

Step 7. Select *Add an empty panel*.

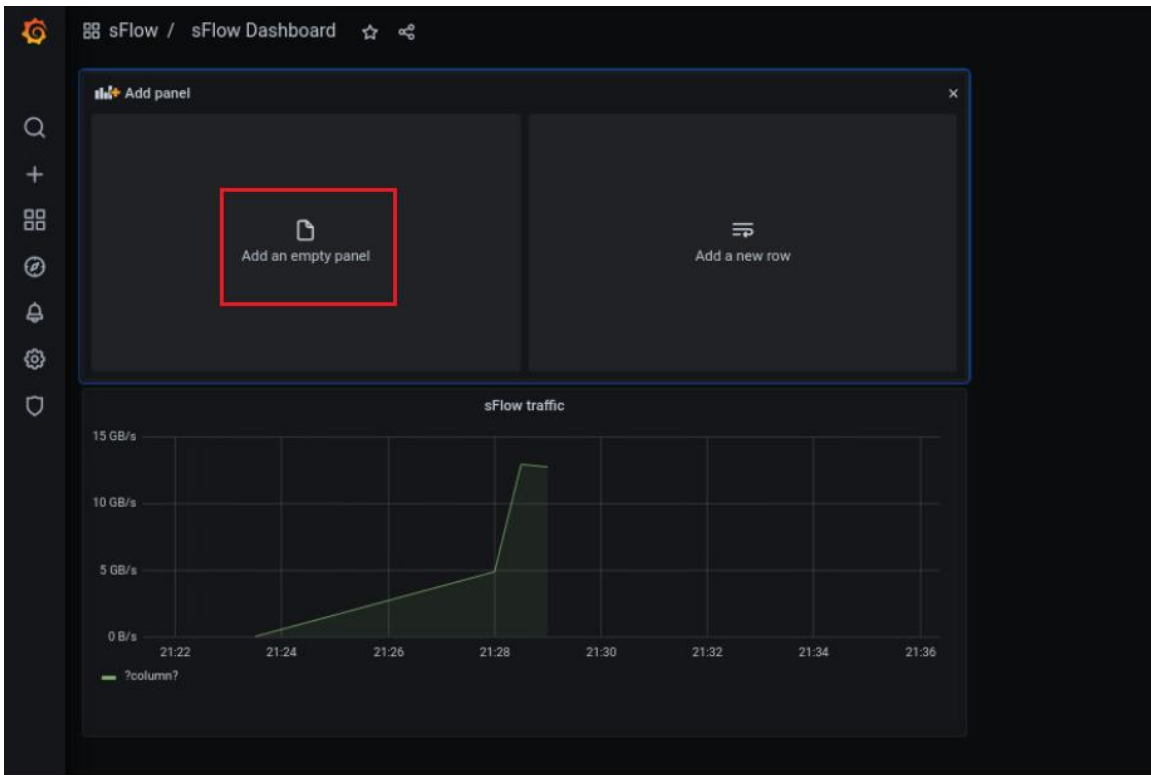


Figure 41. Adding a panel.

Step 8. Change the panel title to *sFlow sampling rate* from the panel setting.

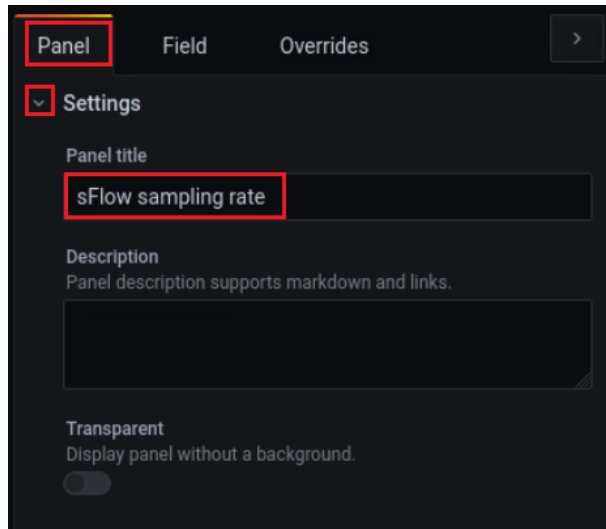


Figure 42. Changing panel title.

Step 9. From panel visualization option, select *Stat*.



Figure 43. Customizing a panel.

Step 10. Select *PostgreSQL* as the data source, select the format as *Table*, and click on *Edit SQL* to insert a custom query. All the steps are highlighted in the following figure.

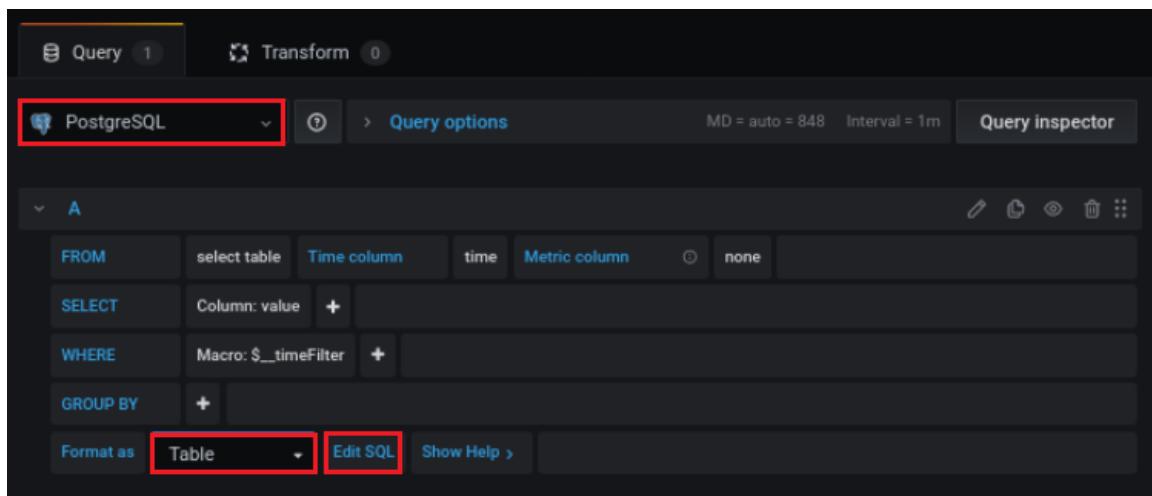


Figure 44. Enabling an SQL custom query.

Step 11. Insert the SQL query as shown in the following figure.

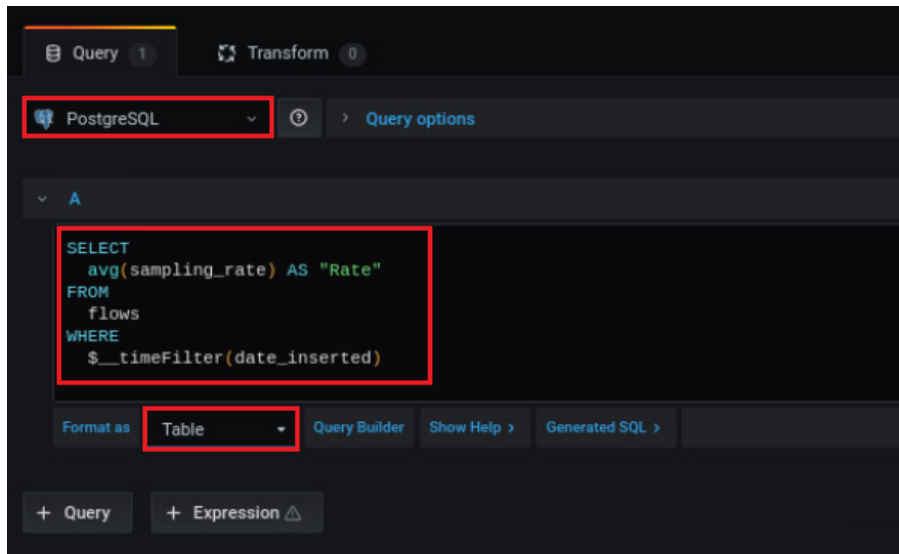


Figure 45. Customizing a panel.

Step 12. You will notice the sampling rate will change from *No Data* to *64* in the panel. Click on *Apply* to save the panel.

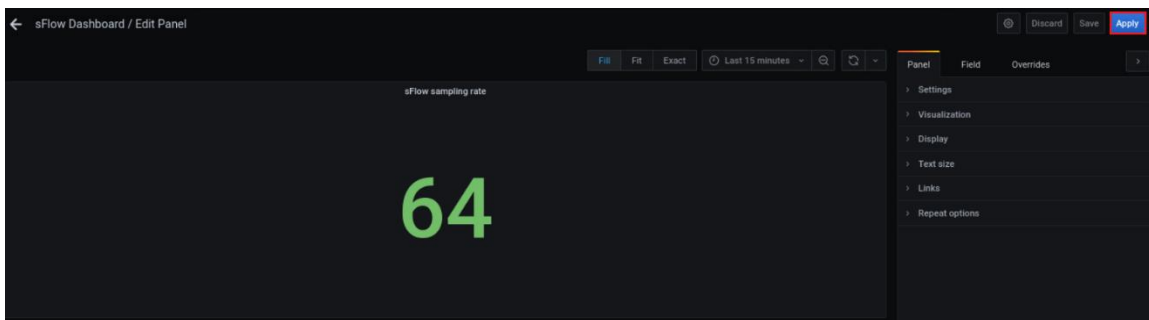


Figure 46. Saving a panel.

Step 13. The sFlow dashboard will look like the following figure.

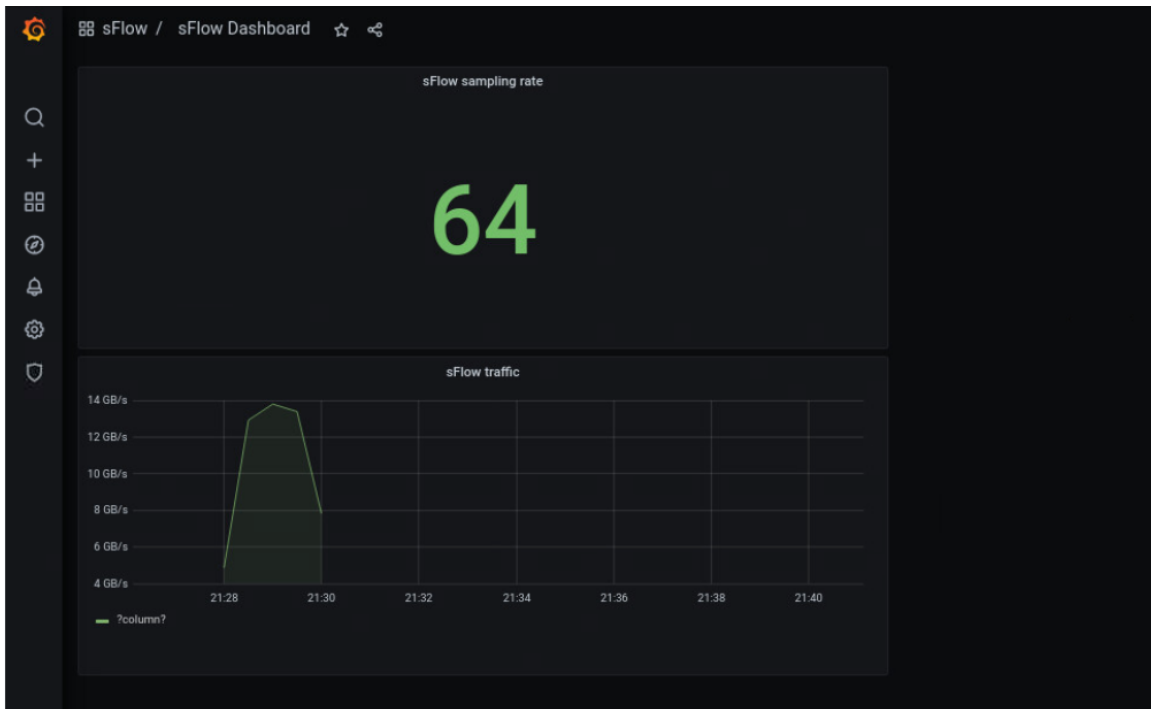


Figure 47. Visualizing the dashboard.

Step 14. Repeat steps 6 and 7 to create a new panel. Change the panel title to *sFlow table*.

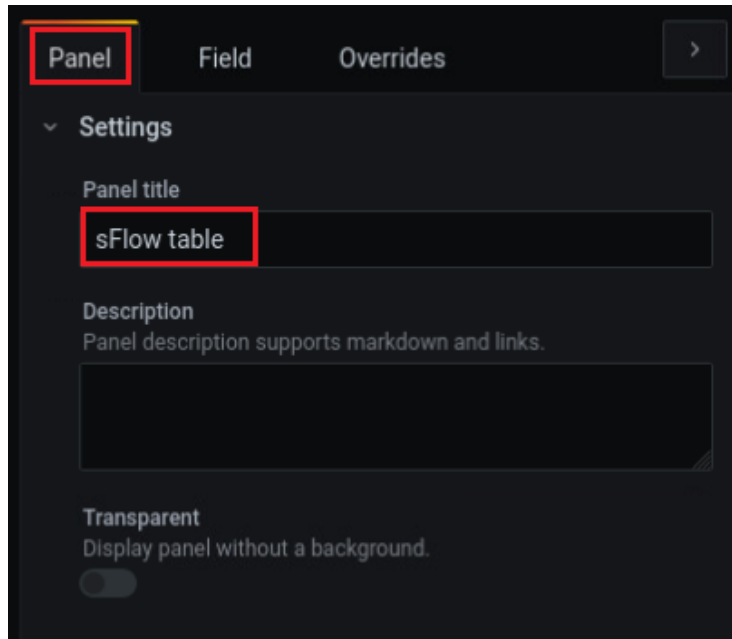


Figure 48. Changing panel title.

Step 15. From panel visualization option, select *Table*.



Figure 49. Customizing a panel.

Step 16. Select *PostgreSQL* as the data source, select the format as *Table*, and click on *Edit SQL* to insert a custom query. All the steps are highlighted in the following figure.

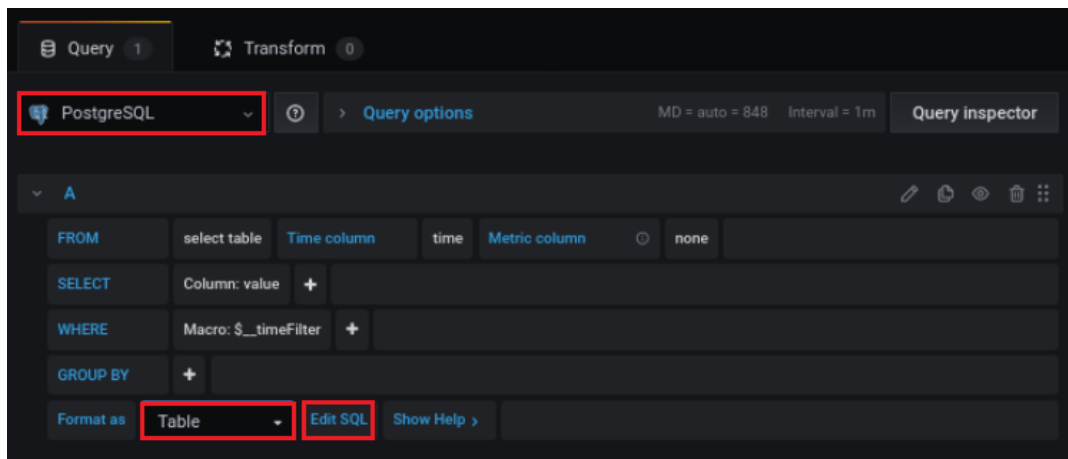


Figure 50. Enabling an SQL custom query.

Step 17. Select *PostgreSQL* as the data source, write the queries and select the format as *Table*. All the steps are highlighted in the following figure.

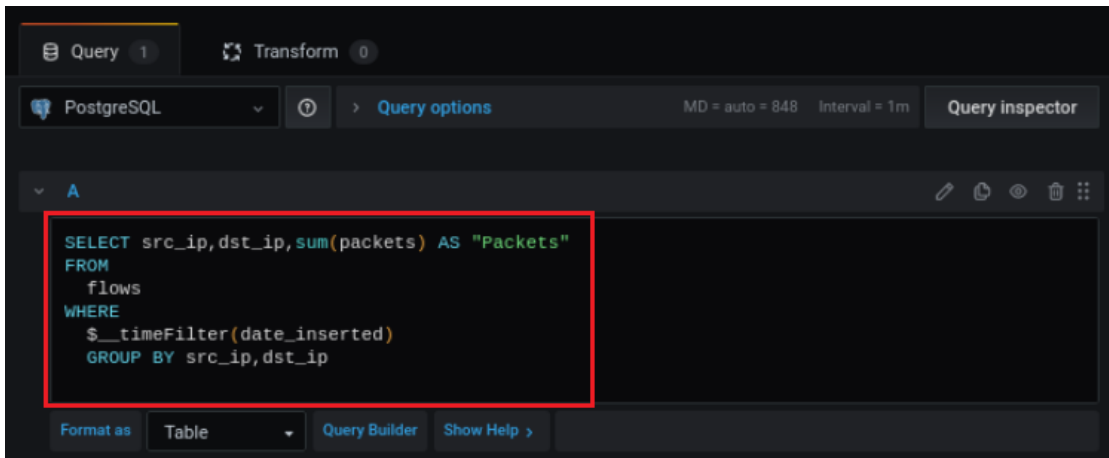


Figure 51. Customizing a panel.

Step 18. The panel will look like the following figure. The table shows source and destination IP addresses and number of packets sent from the source to the destination. Click on *Apply* to save the panel.

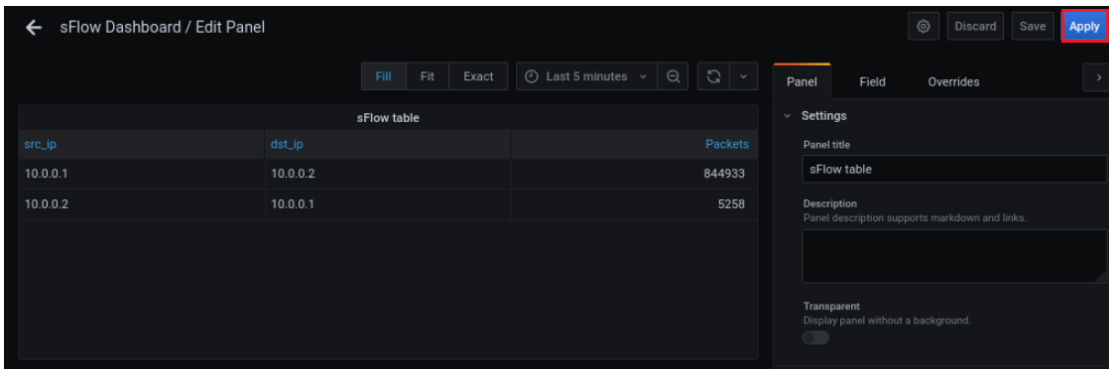


Figure 52. Saving a panel.

Step 19. The dashboard will look like the following figure.

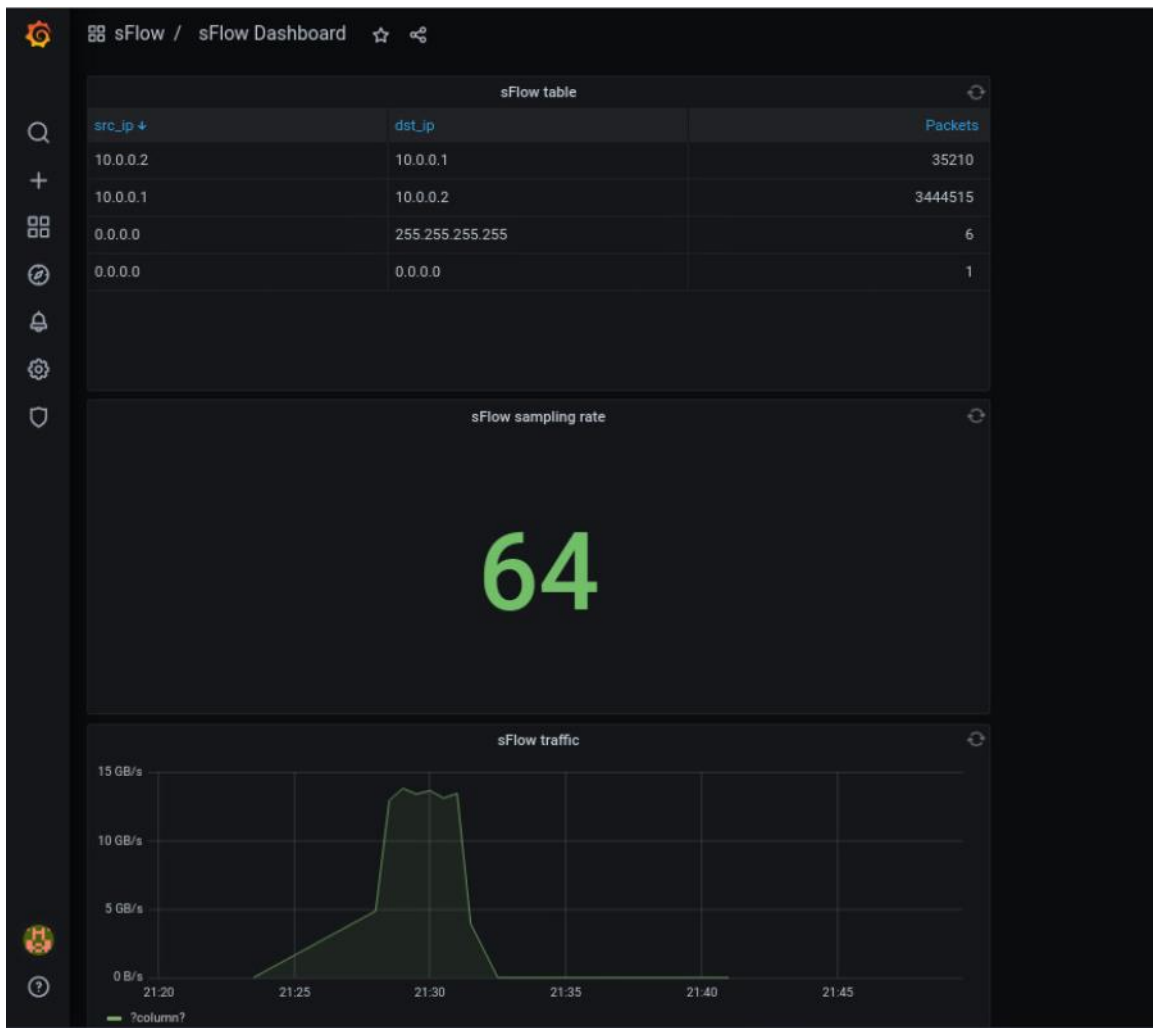


Figure 53. Visualizing the dashboard.

Change the time range to 30 minutes to get a better result.

Step 20. You can drag and drop each panel to rearrange the dashboard. From the right corner of the panel, you can drag to set the size as well (highlighted in the following figure).

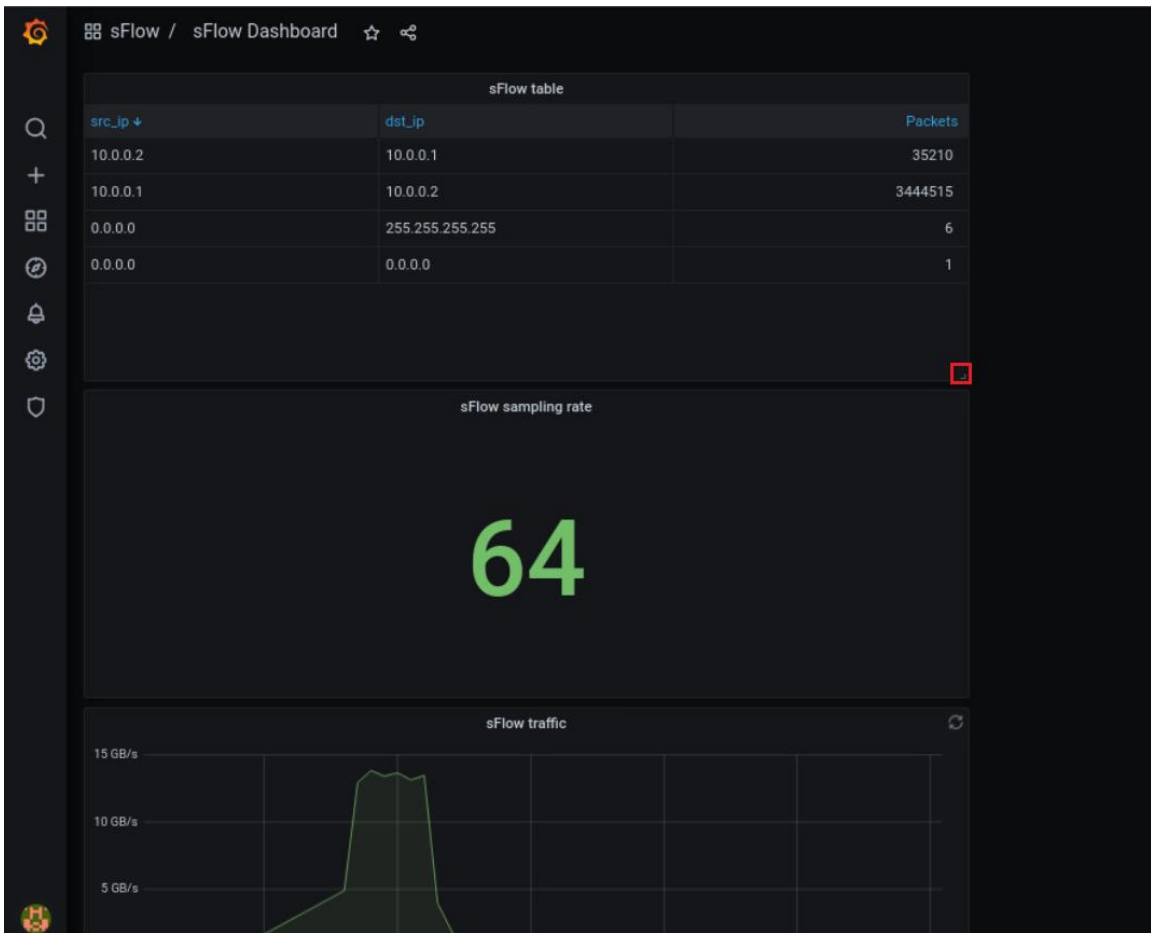


Figure 54. Customizing a dashboard.

Step 21. After rearranging, the final dashboard will look like the following figure.



Figure 55. Visualizing a dashboard.

4.5 Creating an alert in Grafana

Step 1. In this step, you will create an alert. Place your cursor on top of the *sFlow traffic* panel. A dropdown option will appear.

Lab 7: Collecting and Visualizing sFlow data using GoFlow and Grafana



Figure 56. Selecting a panel.

Step 2. Select *Edit* to edit the panel.



Figure 57. Editing a panel.

Step 3. Select *Alert* -> *Create Alert*.



Figure 58. Creating an alert.

Step 4. Set the threshold value of the alert to 7,000,000,000. If the traffic flow is more than 7GB, it will be under red zone. Select *Apply* to make changes.

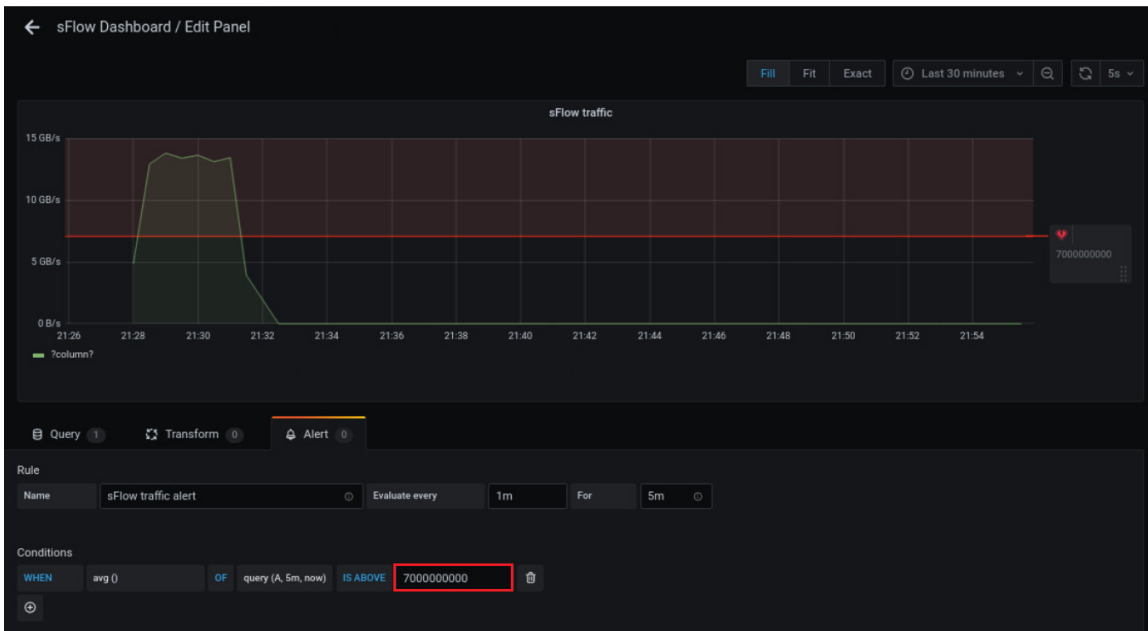


Figure 59. Creating an alert.

Step 5. At this point, the final dashboard will look like the following figure. Save the dashboard.

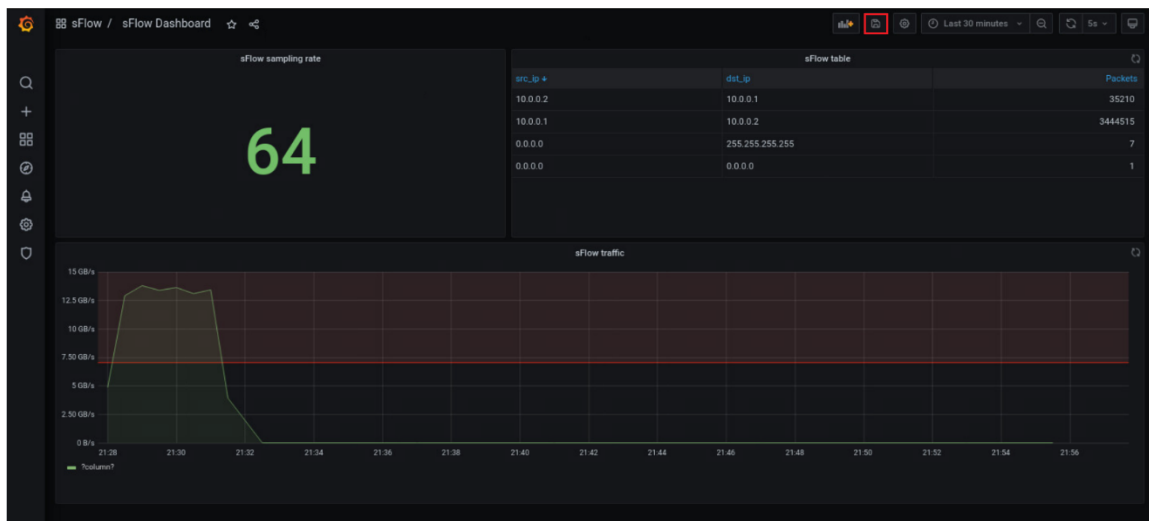


Figure 60. Visualizing a dashboard.

This concludes Lab 7. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

References

1. IBM, "Data Visualization", [Online].
<https://www.ibm.com/cloud/learn/data-visualization>
2. ThoughtCo., "Benefits of the Graphical User Interface", [Online].
<https://www.thoughtco.com/benefits-of-graphical-user-interface-1206357>
3. Grafana Labs, "Introduction to Grafana", [Online].
<https://grafana.com/docs/grafana/latest/introduction/>
4. GitHub, "Flow-pipeline", [Online].
<https://github.com/cloudflare/flow-pipeline>
5. AWS, "What is Apache Kafka?", [Online].
<https://aws.amazon.com/msk/what-is-kafka/>



UNIVERSITY OF
SOUTH CAROLINA

NETWORK MANAGEMENT

Lab 8: Collecting and Visualizing NetFlow data using GoFlow and Grafana

Document Version: **07-08-2022**



Award 1829698

“CyberTraining CIP: Cyberinfrastructure Expertise on High-throughput
Networks for Big Science Data Transfers”

Contents

Overview	3
Objectives.....	3
Lab settings	3
Lab roadmap	3
1 Introduction	3
1.1 Introduction to GoFlow and Grafana	4
2 Lab topology.....	4
2.1 Lab settings.....	5
2.2 Loading a topology	5
3 Launching NetFlow collector and exporter	8
4 Analyzing NetFlow records using Grafana	11
4.1 Launching Grafana	11
4.2 Creating dashboard in Grafana	13
4.3 Performing a connectivity test.....	18
4.4 Exploring Grafana dashboard.....	20
References	30

Overview

This lab introduces Grafana which is a multi-platform open-source analytics and interactive visualization web application. The focus of this lab is to enable NetFlow exporter in Open Virtual Switch (Open vSwitch) and analyze the collected flows using Grafana dashboard.

Objectives

By the end of this lab, you should be able to:

1. Understand the concept of GoFlow and Grafana.
2. Enable NetFlow in Open vSwitch.
3. Analyze NetFlow records using Grafana.

Lab settings

The information in Table 1 provides the credentials to access the Client's virtual machine.

Table 1. Credentials to access Client's virtual machine.

Device	Account	Password
Client	admin	password

Lab roadmap

This lab is organized as follows:

1. Section 1: Introduction.
2. Section 2: Lab topology.
3. Section 3: Launching NetFlow collector and exporter.
4. Section 4: Analyzing NetFlow records using Grafana.

1 Introduction

Data visualization transforms abstract data into a visual context, such as a charts, plots, or graph, to make data easier for the human brain to understand. We can visualize large volumes of data in an understandable and coherent way, which in turn helps us comprehend the information and draw conclusions and insights¹.

Before Graphical User Interface (GUI) systems became popular, command line interface (CLI) systems were the norm. On these systems, users had to input commands using lines

of coded text. The commands ranged from simple instructions for accessing files or directories to far more complicated commands that required many lines of code. Apparently, GUI systems have made computers far more user-friendly than CLI systems².

1.1 Introduction to GoFlow and Grafana

Grafana is an open and composable observability and data visualization platform. The purpose of Grafana dashboards is to bring data together in a way that is both efficient and organized. It allows users to better understand the metrics of their data through queries, informative visualizations and alerts³. Users can also share the dashboards with other team members within an organization, allowing all to explore the data together.

A Grafana dashboard is a powerful open source analytical and visualization tool that consists of multiple individual panels arranged in a grid. The panels interact with configured data sources, including Microsoft SQL server, Prometheus, MySQL, InfluxDB, PostgreSQL and many others. Each panel is connected to a data source. Since the Grafana dashboards support multiple panels in a single grid, users can visualize results from multiple data sources simultaneously.

GoFlow is an application by Cloudflare to collect Netflow/IPFIX/sFlow data. Flow-pipeline is a repository which contains GoFlow collector, kafka (provides a framework for storing, reading, and analyzing streaming data), a database and an inserter (to insert the flows in a database)⁴.

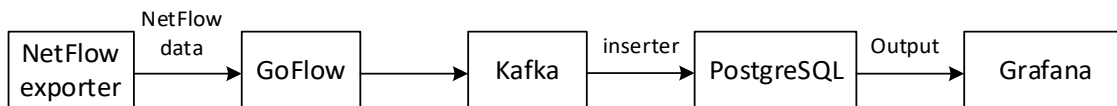


Figure 1. GoFlow and Grafana integration.

Consider Figure 1. GoFlow collector will collect NetFlow data from the NetFlow exporter. The data is sent to kafka. Apache Kafka is a distributed data store optimized for ingesting and processing streaming data in real-time. Streaming data is the data that is continuously generated by thousands of data sources, which typically send the data records simultaneously. A streaming platform needs to handle this constant influx of data and process the data sequentially and incrementally⁵. An inserter is responsible for inserting the flows in the PostgreSQL database from kafka. Grafana is connected to the database to visualize the data based on the user requirements⁴.

2 Lab topology

Consider Figure 2. There are three switches and two end hosts. Switch s3 is acting as a NetFlow exporter. Localhost 127.0.0.1 (loopback address of the Virtual Machine) is the collector. The collector is not visible in Mininet topology since it is a part of the operating system.

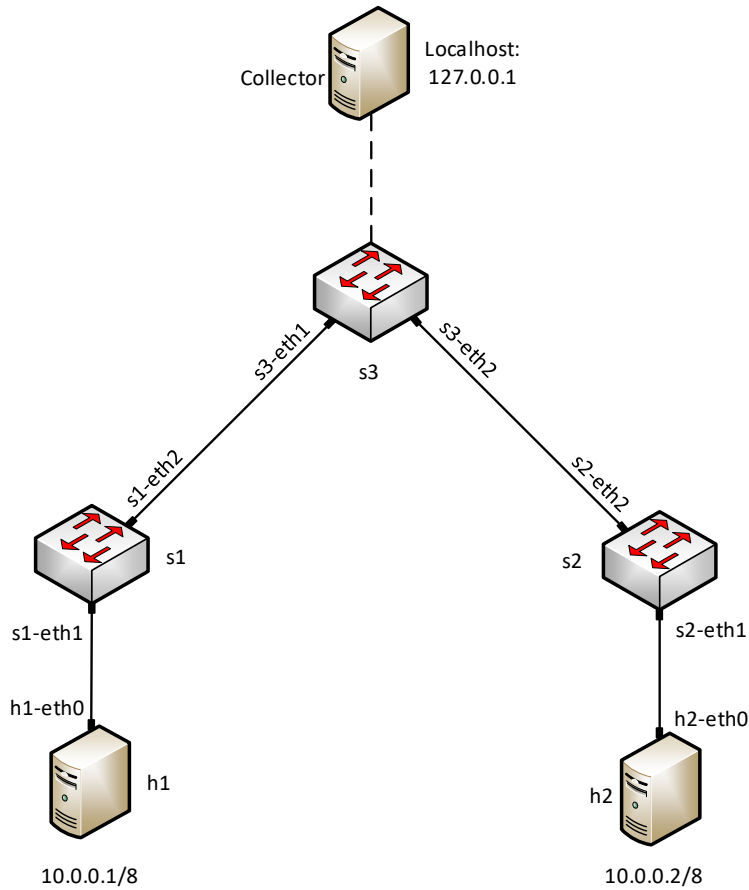


Figure 2. Lab topology.

2.1 Lab settings

The devices should be configured according to Table 2.

Table 2. Topology information.

Device	Interface	IP Address	Subnet
h1	h1-eth0	10.0.0.1	/8
h2	h2-eth0	10.0.0.2	/8

2.2 Loading a topology

Step 1. Click on the Client tab to access the Client PC.

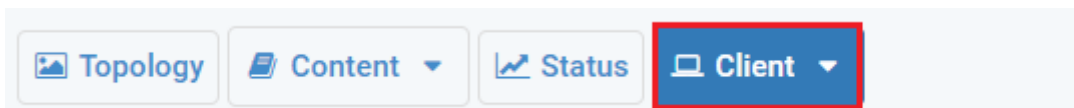


Figure 3. Accessing the Client PC.

Step 2. Start by launching MiniEdit by clicking on desktop's shortcut. When prompted for a password, type `password`.

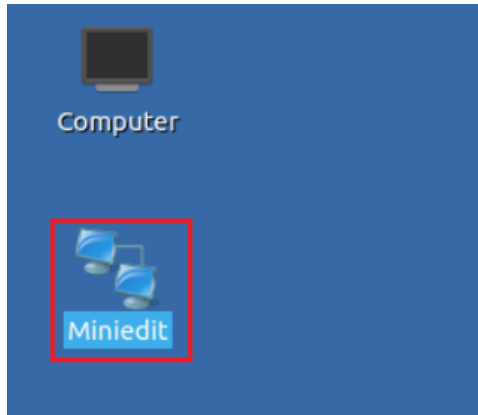


Figure 4. MiniEdit shortcut.

Step 3. On MiniEdit's menu bar, click on *File* then open to load the lab's topology. Open the directory called lab8 and select the file *lab8.mn*. Then, click on *Open* to open the topology.

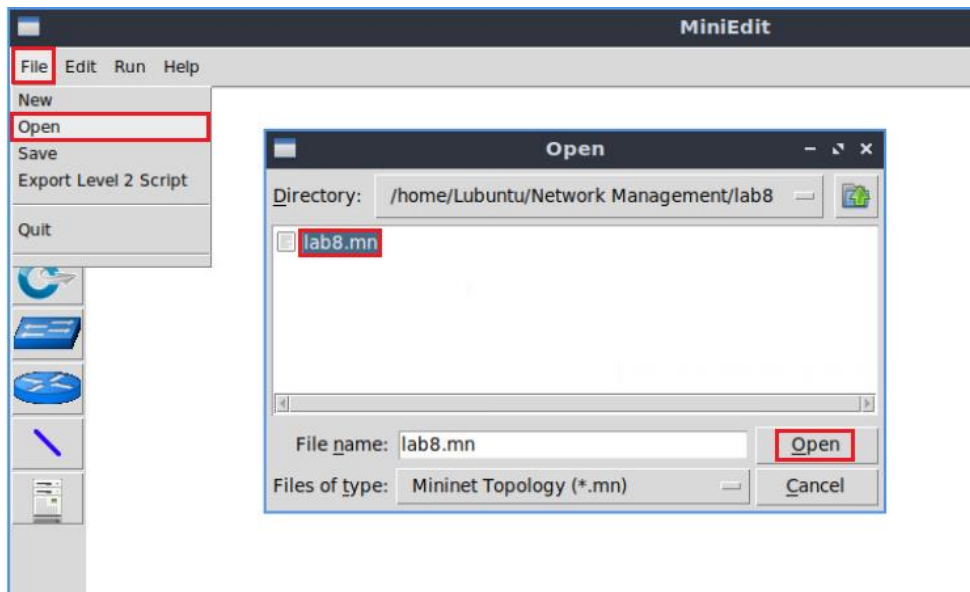


Figure 5. MiniEdit's Open dialog.

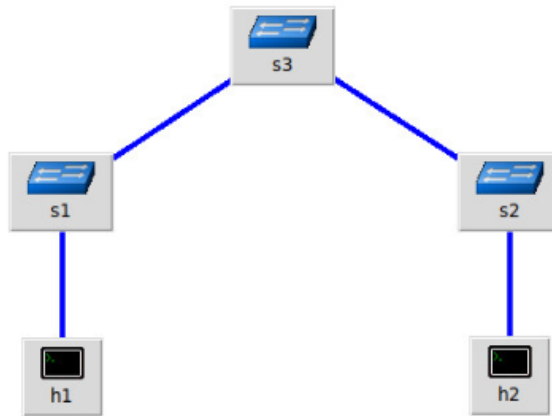


Figure 6. MiniEdit's topology.

Step 4. To proceed with the emulation, click on the *Run* button located on the lower left-hand side.



Figure 7. Starting the emulation.

Step 5. Click on Mininet's terminal, i.e., the one launched when MiniEdit was started.

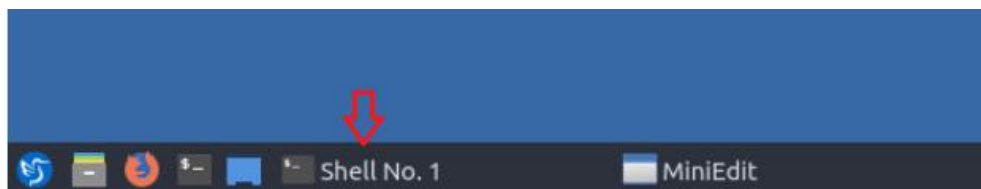


Figure 8. Opening Mininet's terminal.

Step 6. Issue the following command to display the interface names and connections.

```
links
```

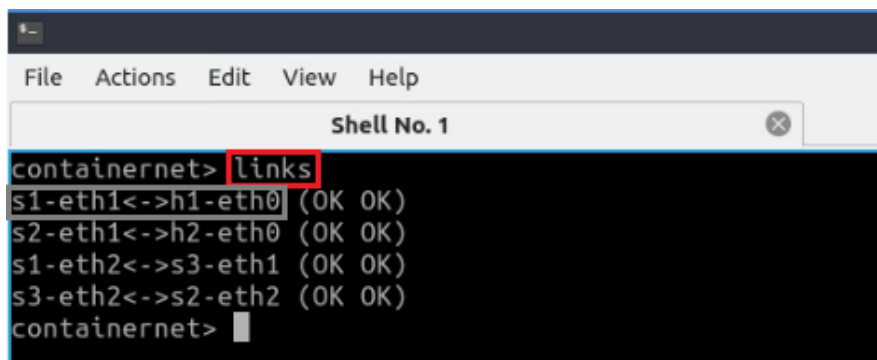


Figure 9. Displaying network interfaces.

In figure 9, the link displayed within the gray box indicates that interface eth1 of host s1 connects to interface eth0 of switch h1 (i.e., *s1-eth1*<-> *h1-eth0*).

3 Launching NetFlow collector and exporter

Step 1. Open the Linux terminal.



Figure 10. Opening Linux terminal.

Step 2. Type the following command to assign a valid IP address to all interfaces. If prompted for password, type `password`.

```
sudo dhclient
```

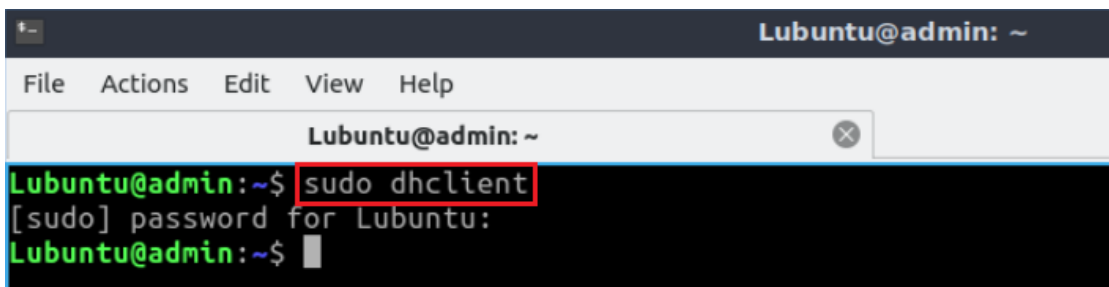


Figure 11. Assigning valid IP addresses to all interfaces.

Step 3. Navigate into *flow-pipeline/compose* directory by issuing the following command. The folder contains a startup file that includes GoFlow, Kafka, PostgreSQL, and an inserter.

```
cd flow-pipeline/compose/
```

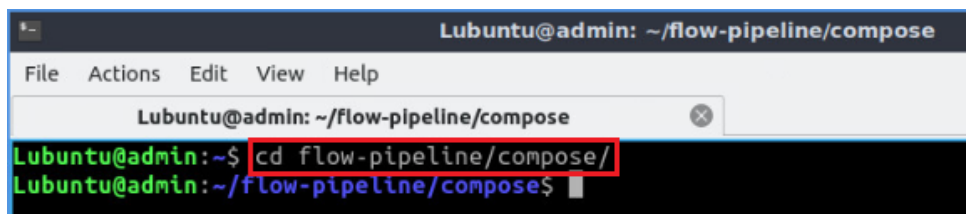


Figure 12. Navigating into flow-pipeline/compose directory.

Step 4. Type the following command to run the startup file.

```
sudo docker-compose -f docker-compose-postgres-collect.yml up
```

```

Lubuntu@admin: ~/flow-pipeline/compose
Lubuntu@admin:~/flow-pipeline/compose$ sudo docker-compose -f docker-compose-postgres-collect.yml up
Pulling goflow (cloudflare/goflow:latest)...
latest: Pulling from cloudflare/goflow
c9b1b535fdd9: Pull complete
13ecdff041c73: Pull complete
f78a00b494eb: Pull complete
Digest: sha256:dc78fad655a60d2a46ff772d3db38d6d8af4817c2244b1671ac4fe7e0302b6f
Status: Downloaded newer image for cloudflare/goflow:latest
Pulling postgres (postgres:latest)...
latest: Pulling from library/postgres
7d63c13d9b9b: Extracting [====>] 2.949MB/31.36MB
cad0f9d5f5fe: Download complete
ff74a7a559cb: Download complete
c43dfd845683: Download complete
kafka_1 | [2021-11-01 17:40:16,456] INFO [GroupMetadataManager brokerId=1001] Finished loading offsets and group metadata from __consu
inserter_1 | time="2021-11-01T17:40:20Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:40:25Z" level=info msg="Processed 0 records in the last iteration."
prometheus_1 | level=info ts=2021-11-01T17:40:26.062Z caller=compact.go:509 component=tsdb msg="write block resulted in empty block" mint=1
635717600000 maxt=1635724800000 duration=22.041249ms
prometheus_1 | level=info ts=2021-11-01T17:40:26.064Z caller=head.go:798 component=tsdb msg="Head GC completed" duration=1.278927ms
prometheus_1 | level=info ts=2021-11-01T17:40:26.064Z caller=checkpoint.go:97 component=tsdb msg="Creating checkpoint" from_segment=5 to_se
gment=7 mint=1635724800000
prometheus_1 | level=info ts=2021-11-01T17:40:26.067Z caller=head.go:967 component=tsdb msg="WAL checkpoint complete" first=5 last=7 durati
on=2.871342ms
goflow_1 | time="2021-11-01T17:40:28Z" level=info msg="Starting GoFlow"
goflow_1 | time="2021-11-01T17:40:28Z" level=info msg="Listening on UDP :2056" Type=NetFlowLegacy
goflow_1 | time="2021-11-01T17:40:28Z" level=info msg="Listening on UDP :6343" Type=sFlow
goflow_1 | time="2021-11-01T17:40:28Z" level=info msg="Listening on UDP :2055" Type=NetFlow
initializer_1 | Exception in thread "main" java.util.UnrecognizedOptionException: zookeeper is not a recognized option
initializer_1 | at joptsimple.OptionException.unrecognizedOption(OptionException.java:108)
initializer_1 | at joptsimple.OptionParser.handleLongOptionToken(OptionParser.java:510)
initializer_1 | at joptsimple.OptionParserState$2.handleArgument(OptionParserState.java:56)
initializer_1 | at joptsimple.OptionParser.parse(OptionParser.java:396)
initializer_1 | at kafka.admin.TopicCommand$TopicCommandOptions.<init>(TopicCommand.scala:517)
initializer_1 | at kafka.admin.TopicCommand$.main(TopicCommand.scala:47)
initializer_1 | at kafka.admin.TopicCommand.main(TopicCommand.scala)
compose_initializer_1 exited with code 1
inserter_1 | time="2021-11-01T17:40:30Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:40:35Z" level=info msg="Processed 0 records in the last iteration."

```

Figure 13. Enabling the collector.

Consider the figure above. It may take some time to start the collector. The highlighted part shows that the collector *GoFlow* is starting. UDP port 6343 is for sFlow, 2056 is for Netflow and port 2055 is for IPFIX. Once the collector is running, you will see the number of records processed by the inserter.

Once you run a test between hosts h1 and h2, you will notice number of records increasing.

Step 5. Click on the *File* option and select *+ New Tab* or press `Ctrl+Shift+T`.

```

Lubuntu@admin: ~/flow-pipeline/compose
File Actions Edit View Help
+ New Tab Ctrl+Shift+T
New Tab From Preset
Close Tab Ctrl+Shift+W
New Window Ctrl+Shift+N
Preferences...
Quit
inserter_1 | time="2021-11-01T17:49:20Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:49:25Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:49:30Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:49:35Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:49:40Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:49:45Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:49:50Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:49:55Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:50:00Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:50:05Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:50:10Z" level=info msg="Processed 0 records in the last iteration."
inserter_1 | time="2021-11-01T17:50:15Z" level=info msg="Processed 0 records in the last iteration."

```

Figure 14. Opening a new terminal tab.

Step 6. Type the following command to enable a NetFlow exporter. If prompted for password, type `password`.

```

sudo ovs-vsctl -- set Bridge s3 netflow=@nf -- --id=@nf create netflow
targets="\127.0.0.1:2056\" active_timeout=30

```

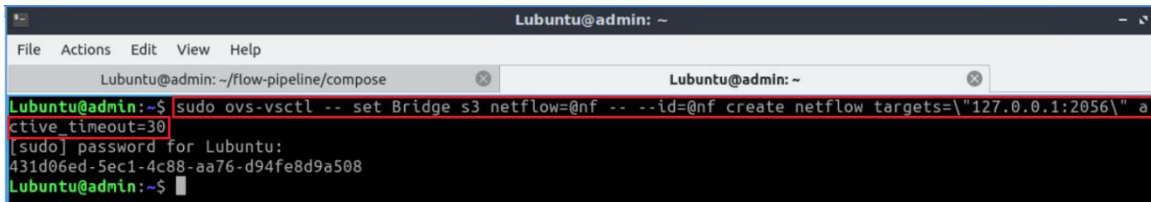


Figure 15. Enabling Netflow exporter.

Consider the figure above. The command creates a NetFlow ID and attaches it to switch s3. Switch s3 is acting as an exporter and transmits data to the collector. 127.0.0.1 is the collector IP and the port is 2056. *Active_timeout* is the frequency of active flow records that are exported from the database to the collector.

Step 7. Type the following command to verify NetFlow configuration.

```
sudo ovs-vsctl list netflow
```

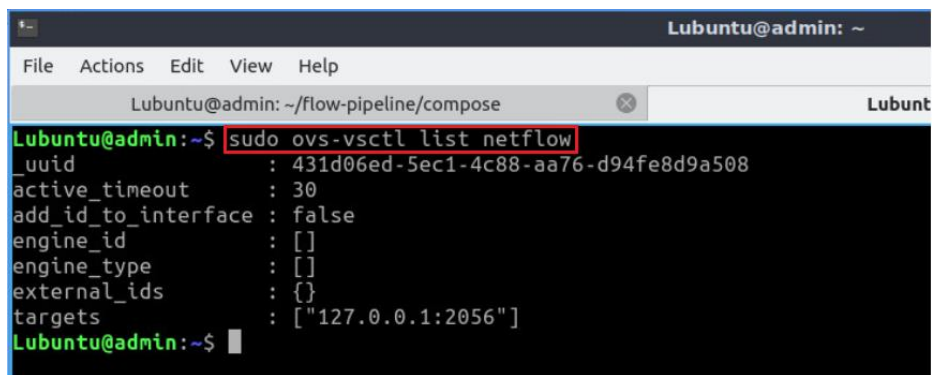


Figure 16. Verifying NetFlow configuration.

Consider the figure above. One NetFlow exporter is running, target collector IP is 127.0.0.1 and the port is 2056.

Step 8. Click on the *File* option and select *+ New Tab* or press `Ctrl+Shift+T`.

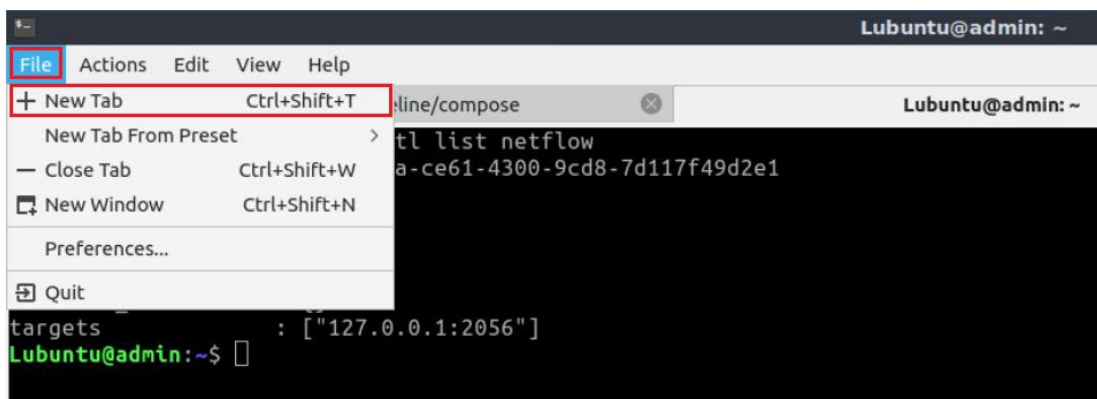


Figure 17. Opening a new terminal tab.

Step 9. Navigate into *flow-pipeline/compose/postgres* directory by issuing the following command.

```
cd flow-pipeline/compose/postgres/
```

```
Lubuntu@admin: ~/flow-pipeline/compose/postgres
Lubuntu@admin:~$ cd flow-pipeline/compose/postgres/
Lubuntu@admin:~/flow-pipeline/compose/postgres$
```

Figure 18. Navigating into flow-pipeline/compose/postgres directory.

Step 10. Type the following command to show file *create.sh* located inside the directory.

```
Lubuntu@admin:~/flow-pipeline/compose/postgres$ cat create.sh
#!/bin/bash
set -e

psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_DB" <<-EOSQL
CREATE TABLE IF NOT EXISTS flows (
    id bigserial PRIMARY KEY,
    date_inserted timestamp default NULL,

    time_flow timestamp default NULL,
    type integer,
    sampling_rate integer,
    src_as bigint,
    dst_as bigint,
    src_ip inet,
    dst_ip inet,

    bytes bigint,
    packets bigint,

    etype integer,
    proto integer,
    src_port integer,
    dst_port integer
);
EOSQL
Lubuntu@admin:~/flow-pipeline/compose/postgres$
```

Figure 19. Displaying a file located inside the directory.

Consider the figure above. If any data is stored in the postgres database, a table *flows* will be created. The table has different columns such as *src_ip*, *dst_ip*, *bytes* and *packets*. You will explore more about the database when you analyze records in Grafana dashboard.

4 Analyzing NetFlow records using Grafana

In this section, you will analyze NetFlow data. You will use graphical interface of Grafana to view the results. You will create a folder and a dashboard which can be saved and used later.

4.1 Launching Grafana

Step 1. Open the browser.

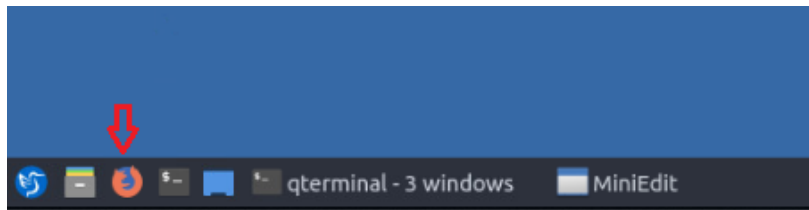


Figure 20. Opening a browser.

Step 2. In the search engine, type `localhost:3000/login`. The username is `admin` and the password is `admin`. Then, click on *Login*.

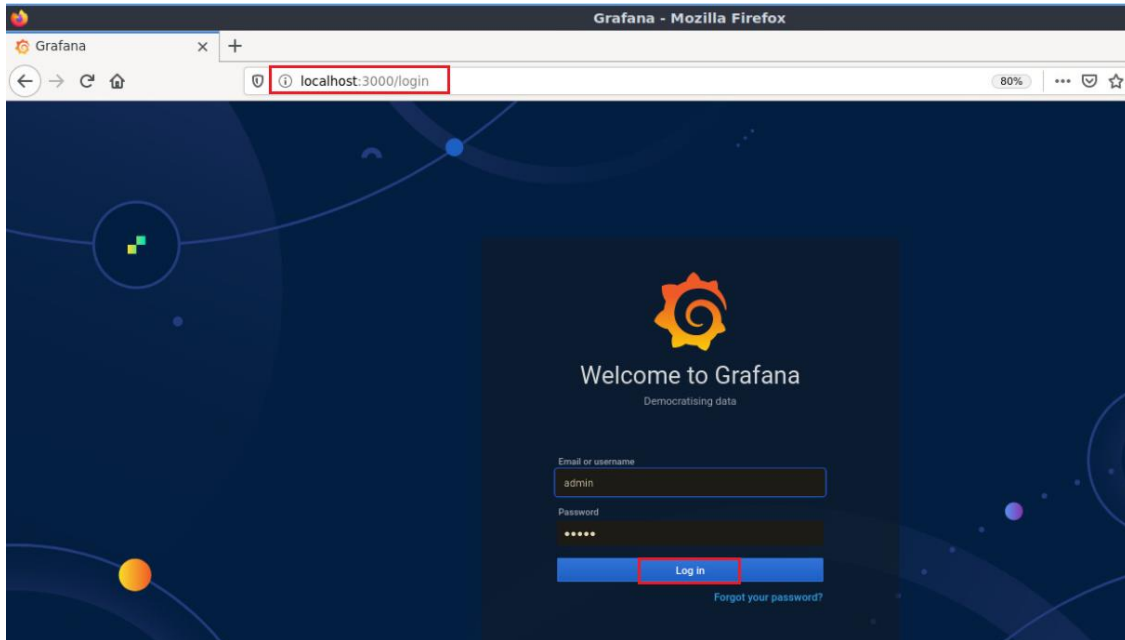


Figure 21. Login to Grafana.

Step 3. There is an option to change the password. You can also click on *skip* if you do not want to change the password.

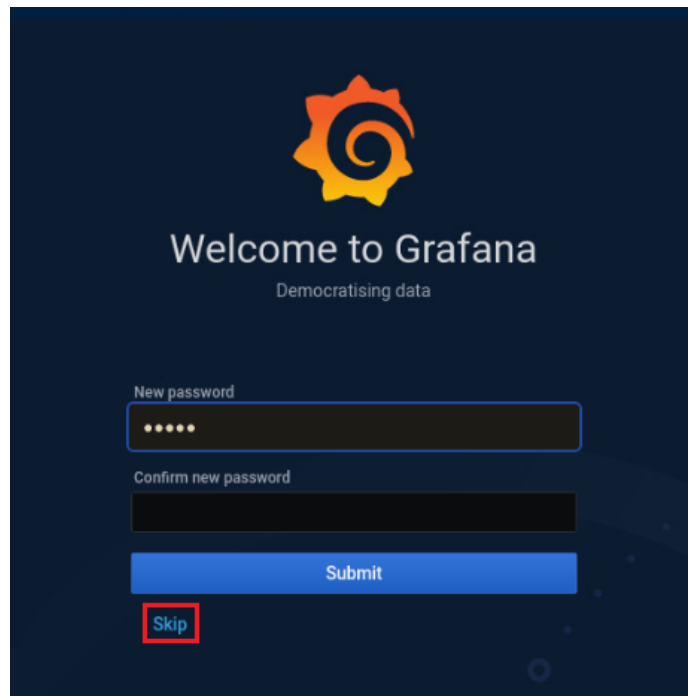


Figure 22. Login to Grafana.

Step 4. Once you login, you will be directed to the homepage of Grafana.

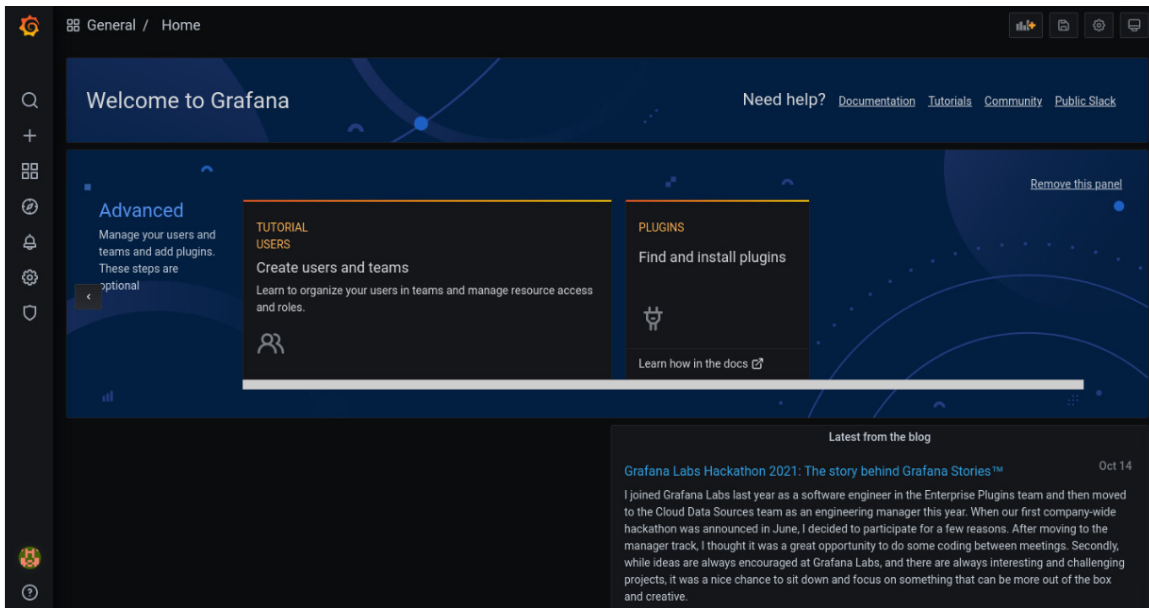



Figure 23. Visualizing Grafana homepage.

4.2 Creating dashboard in Grafana

In this section, you will create a Grafana dashboard.

Step 1. In this step, you will create a folder for NetFlow. You will be able to add dashboards in that particular folder. Click on the  sign located on the left side. Select Create -> Folder.

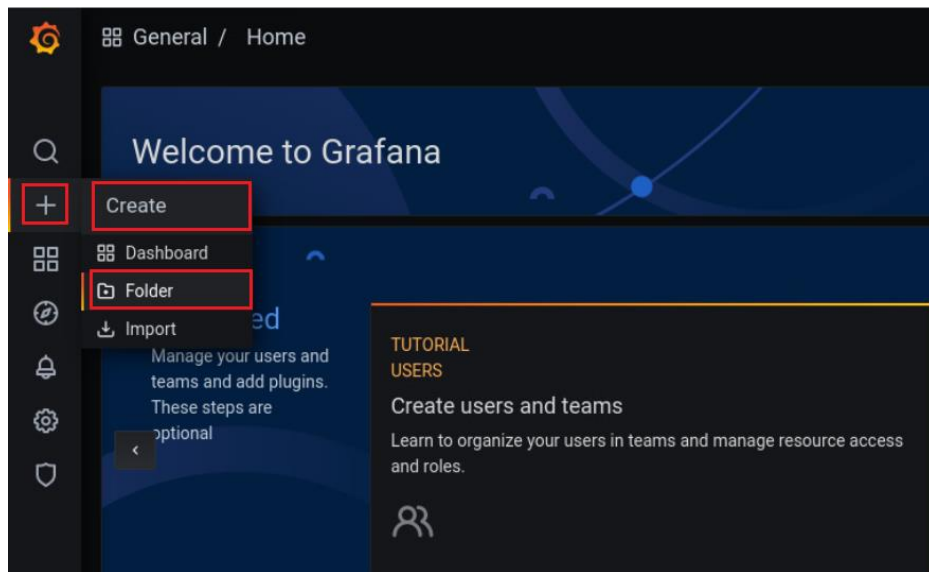


Figure 24. Creating a folder.

Step 2. Name the folder as *NetFlow* and select *Create*.

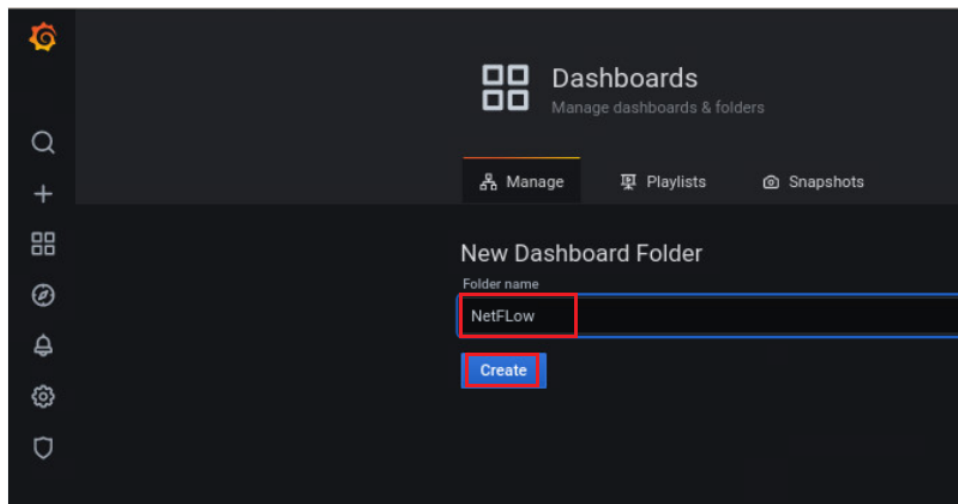


Figure 25. Creating a folder.

Step 3. Once you create a folder, you will see an option to create dashboard. Click on *Create Dashboard*. You can create multiple panels to make a dashboard.

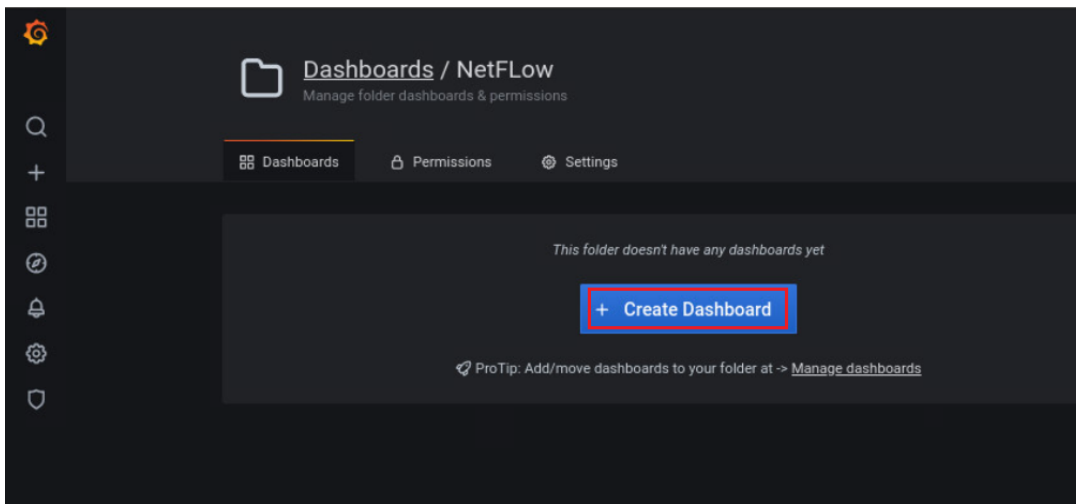


Figure 26. Creating a dashboard.

Step 4. Select *Add an empty panel*.

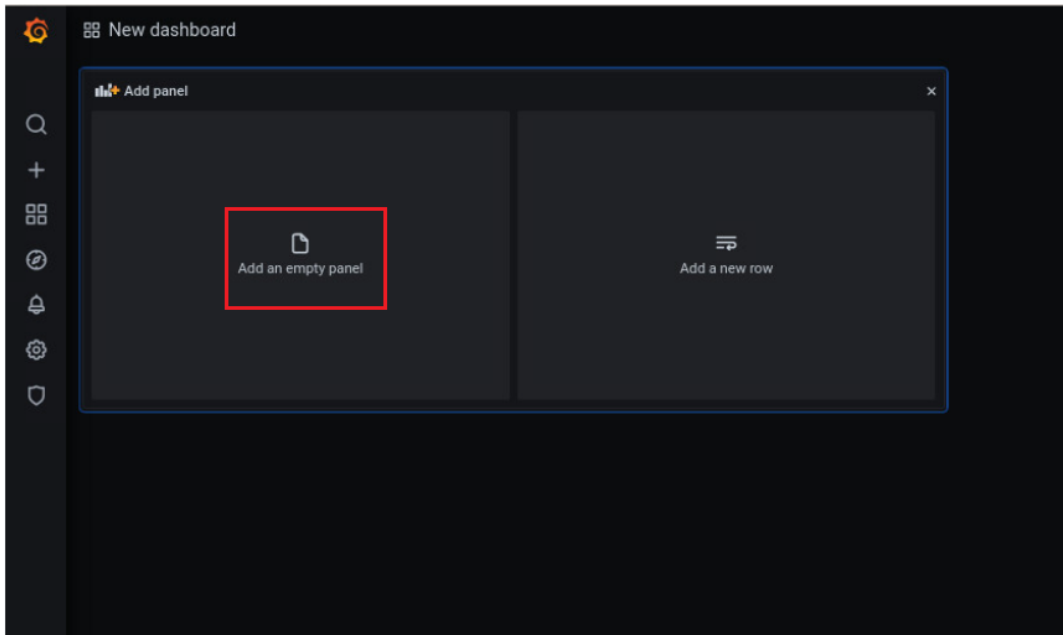


Figure 27. Adding an empty panel into Grafana dashboard.

Step 5. By default, it shows a demo graph. Select the source, *PostgreSQL* from the dropdown box showed in the following figure.

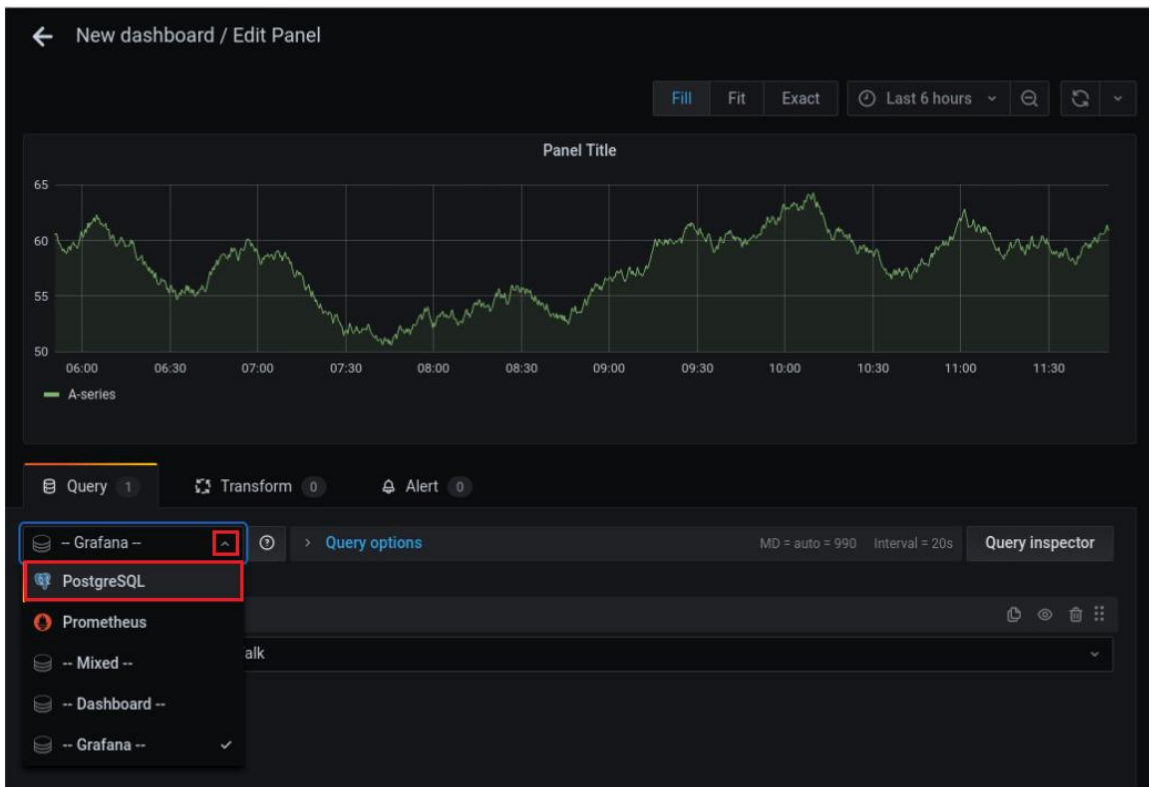


Figure 28. Customizing a panel.

Step 6. Grafana uses queries to communicate with data sources to get data for the visualization. A query is a question written in the query language used by the data source. Data sources have different query languages and syntaxes to ask for the data. Grafana provides a query editor which helps you to write queries. Depending on your data source, the query editor might provide auto-completion, metric names, or variable suggestion. In this lab, you will write queries manually. Select *Edit SQL* so that you can write queries.

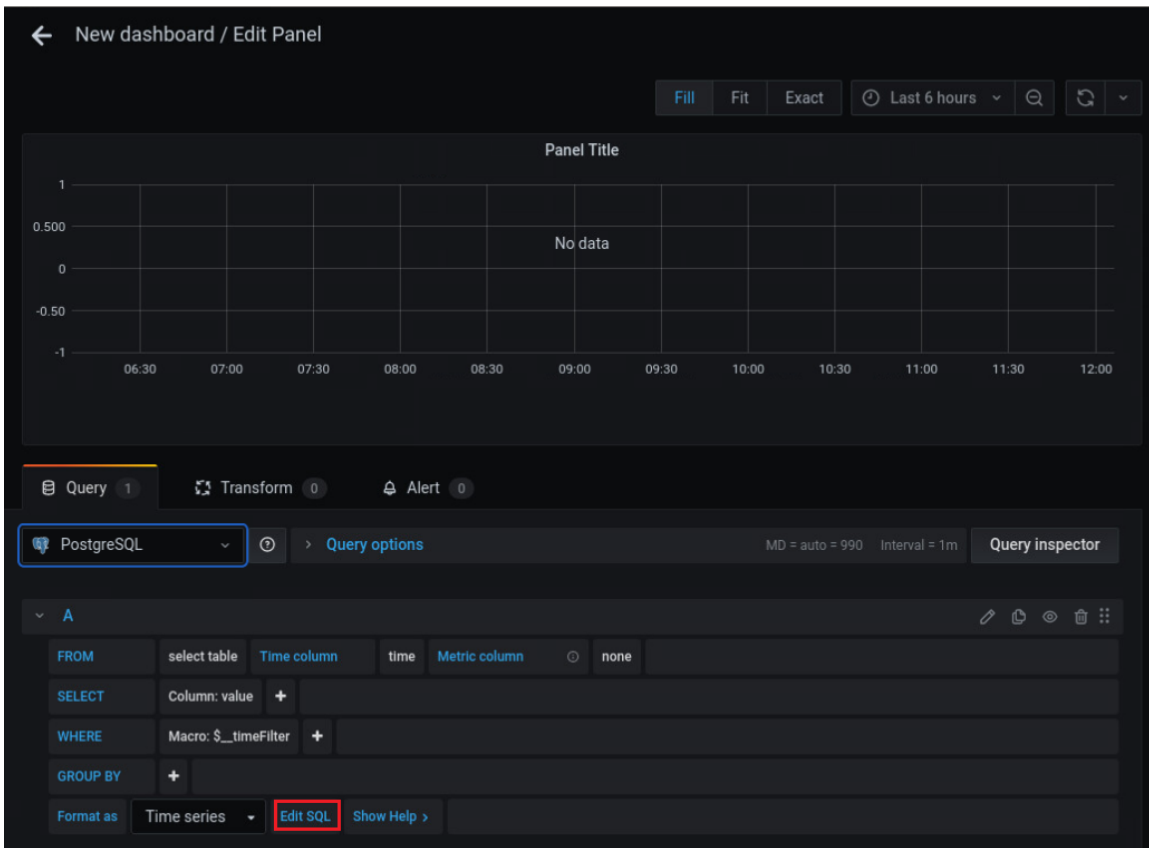


Figure 29. Customizing a panel.

Step 7. Type the query as shown in the following figure to create a graph. The query will extract data from the database, and you will visualize a graph for time vs flows in real-time.

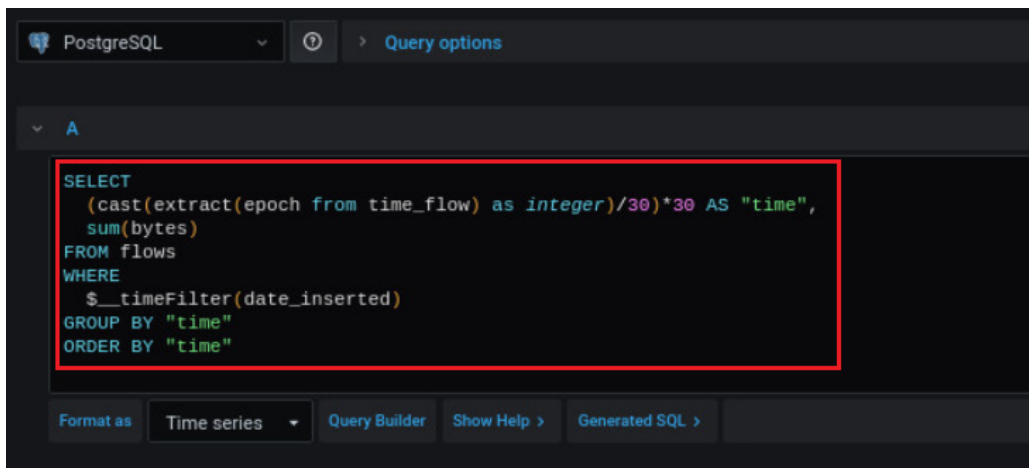


Figure 30. Customizing a panel.

Step 8. Select *Panel* located on the top right of the panel. From the dropdown box of Axes, select *Left Y-> Unit -> Data rate -> bytes/sec*.

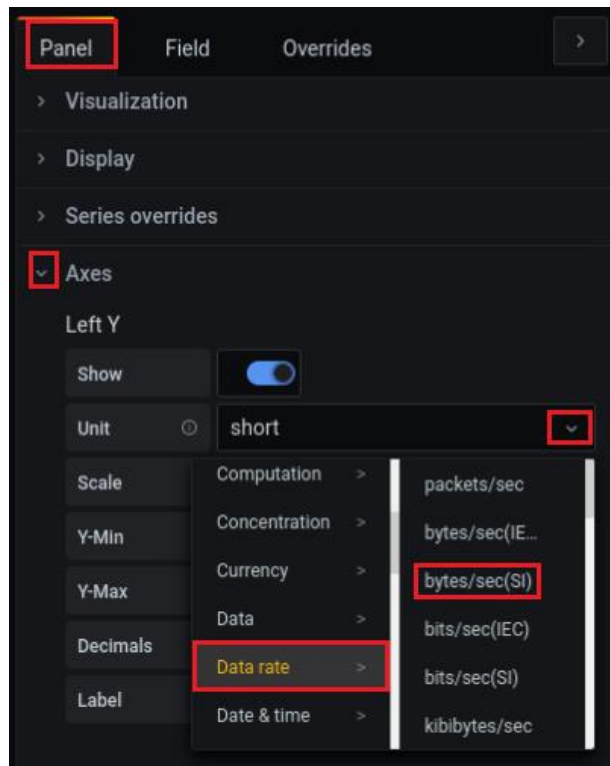


Figure 31. Customizing a panel.

Consider the figure above. This will change the unit for Y-axis.

4.3 Performing a connectivity test

Step 1. Go back to MiniEdit. Hold right-click on host h2 and select *Terminal*. This opens the terminal of host h2 and allows the execution of commands on that host.

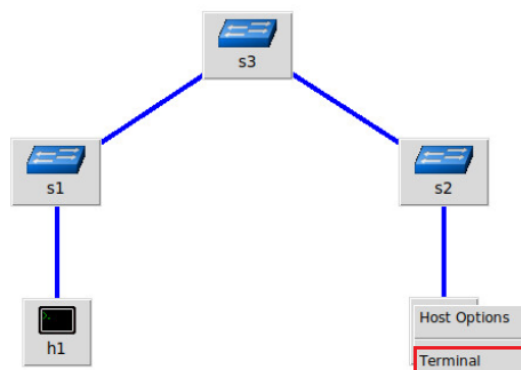


Figure 32. Opening a terminal on host h2.

Step 2. In host h2 terminal, type the following command to run the host in server mode.

```
iperf3 -s
```



Figure 33. Running host h2 in server mode.

Consider the figure above. The figure shows that host h2 is acting as a server and listening to port 5201.

Step 3. Open host h1 terminal and type the following command to run the host in client mode.

```
iperf3 -c 10.0.0.2 -t 100
```

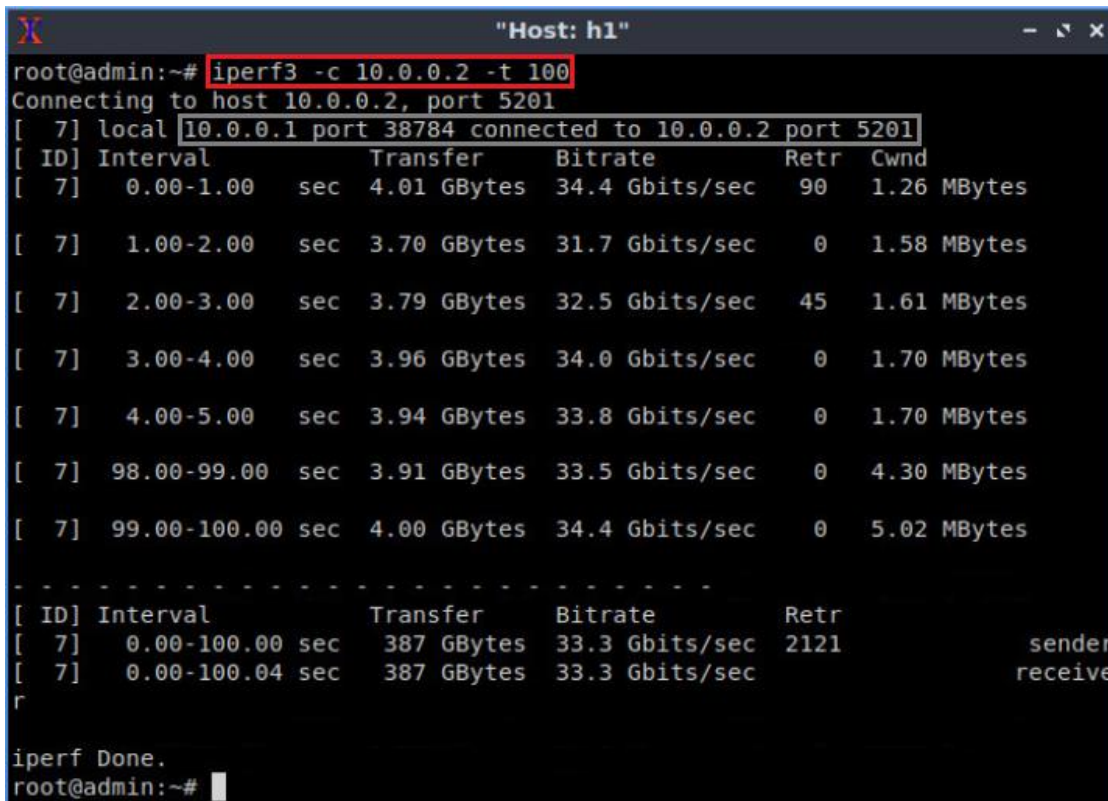


Figure 34. Running host h1 in client mode.

Consider the figure above. The test runs for 100 seconds. You will notice total transfer is 387 Gbytes.

You might get a different result.

Step 4. You will notice that host h2 is creating a connection with host h1 using port 38782. Once the connection is established, port 5201 is connected to port 38784.

```

Host: h2
root@admin:~# iperf3 -s
-----
Server listening on 5201
-----
Accepted connection from 10.0.0.1, port 38782
[ 7] local 10.0.0.2 port 5201 connected to 10.0.0.1 port 38784
[ ID] Interval           Transfer             Bitrate
[ 7]  0.00-1.00   sec  3.86 GBytes        33.1 Gbits/sec
[ 7]  1.00-2.00   sec  3.69 GBytes        31.7 Gbits/sec
[ 7]  2.00-3.00   sec  3.79 GBytes        32.6 Gbits/sec
[ 7]  3.00-4.00   sec  3.95 GBytes        34.0 Gbits/sec
[ 7]  4.00-5.00   sec  3.94 GBytes        33.8 Gbits/sec
[ 7]  5.00-6.00   sec  3.91 GBytes        33.6 Gbits/sec
[ 7]  6.00-7.00   sec  3.90 GBytes        33.5 Gbits/sec
[ 7]  7.00-8.00   sec  3.92 GBytes        33.7 Gbits/sec
[ 7]  8.00-9.00   sec  3.77 GBytes        32.3 Gbits/sec
[ 7]  9.00-10.00  sec  3.05 GBytes        26.2 Gbits/sec
    
```

Figure 35. Verifying result in host h2.

You might get different ports.

Step 5. Go to the flow-pipeline terminal. You will notice number of records increasing.

```

Lubuntu@admin: ~/flow-pipeline/compose
File Actions Edit View Help
Lubuntu@admin: ~/flow-pipeline/compose
Lubuntu@admin: ~

insenter_1 | time="2021-11-22T18:25:46Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:25:51Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:25:56Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:01Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:06Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:11Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:16Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:21Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:26Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:31Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:36Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:41Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:46Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:51Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:26:56Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:01Z" level=info msg="Processed 1 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:06Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:11Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:16Z" level=info msg="Processed 2 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:21Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:26Z" level=info msg="Processed 2 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:31Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:36Z" level=info msg="Processed 1 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:41Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:46Z" level=info msg="Processed 16 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:51Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:27:56Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:28:01Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:28:06Z" level=info msg="Processed 0 records in the last iteration."
insenter_1 | time="2021-11-22T18:28:11Z" level=info msg="Processed 1 records in the last iteration."
insenter_1 | time="2021-11-22T18:28:16Z" level=info msg="Processed 15 records in the last iteration."
    
```

Figure 36. Verifying NetFlow records collected by the inserter.

4.4 Exploring Grafana dashboard

Step 1. From the dropdown box showed in the following figure, you can choose time range to view data. Select 5 minutes to see the real-time traffic.

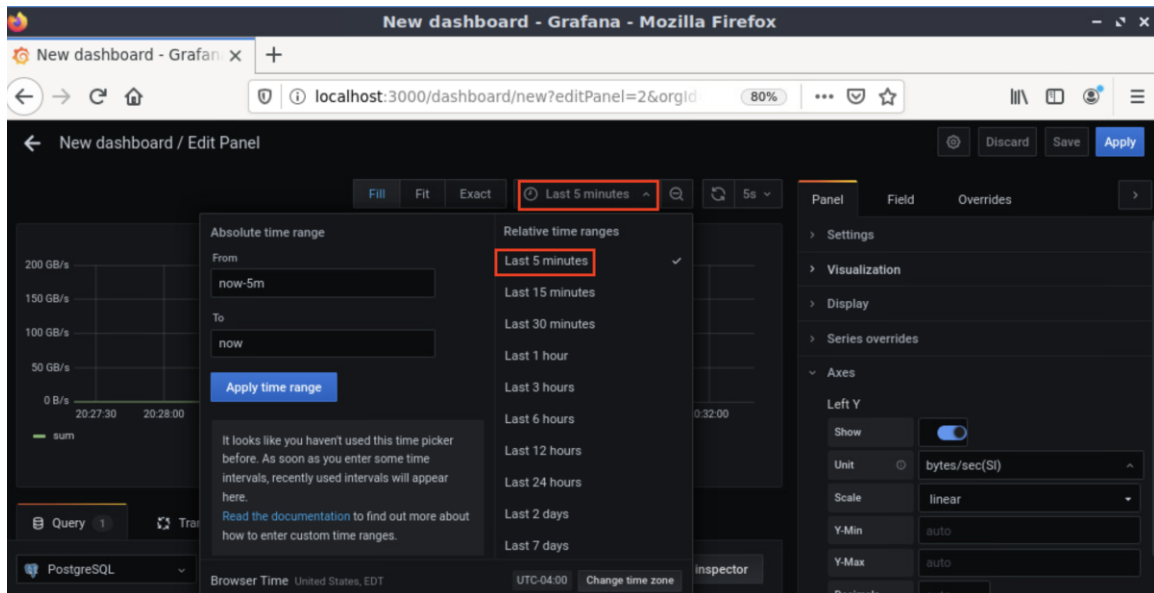


Figure 37. Customizing a panel.

Step 2. From the dropdown box showed in the figure, you can choose time to refresh the dashboard. Select 5s (5 seconds) to see the real-time traffic.

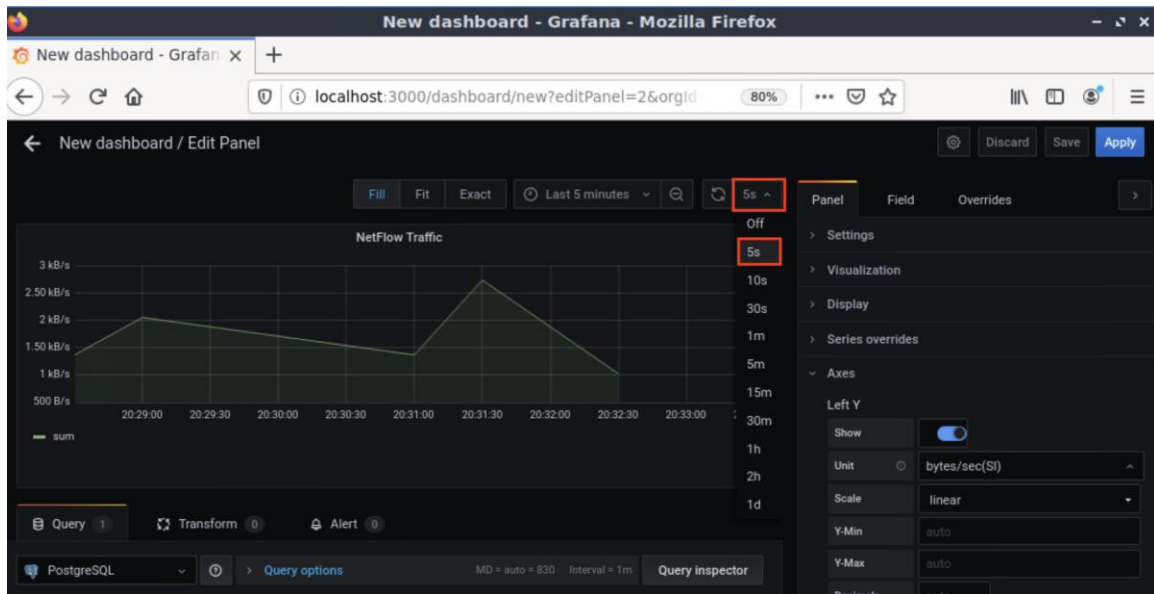


Figure 38. Customizing a panel.

The figure above shows the traffic for the test running between hosts h1 and h2.

Step 3. In this step, you will save the panel. Go to the *panel* option, select *settings*, and change the panel title to *NetFlow Traffic*. Click on *apply* to save the panel in the dashboard.

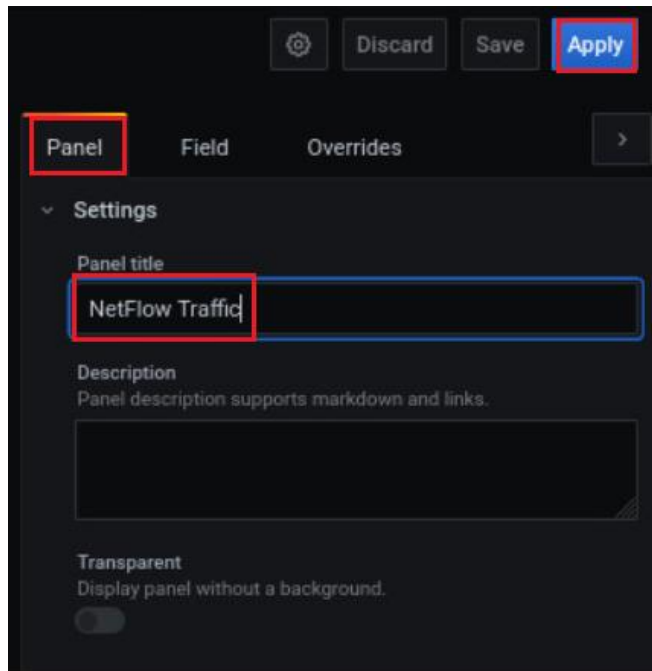


Figure 39. Saving a panel.

Step 4. The dashboard will look like the following figure. Click on the *save* option located on the top right side.

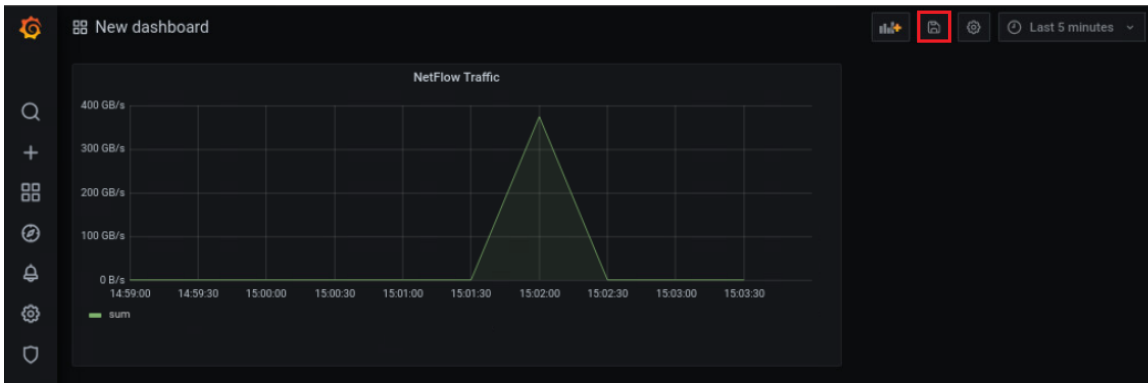


Figure 40. Saving a dashboard.

Consider the figure above. Notice that the total transfer is 387 Gbytes.

You might get a different result based on the connectivity test.

Step 5. Change the dashboard name to *NetFlow Dashboard* and click on *save*.

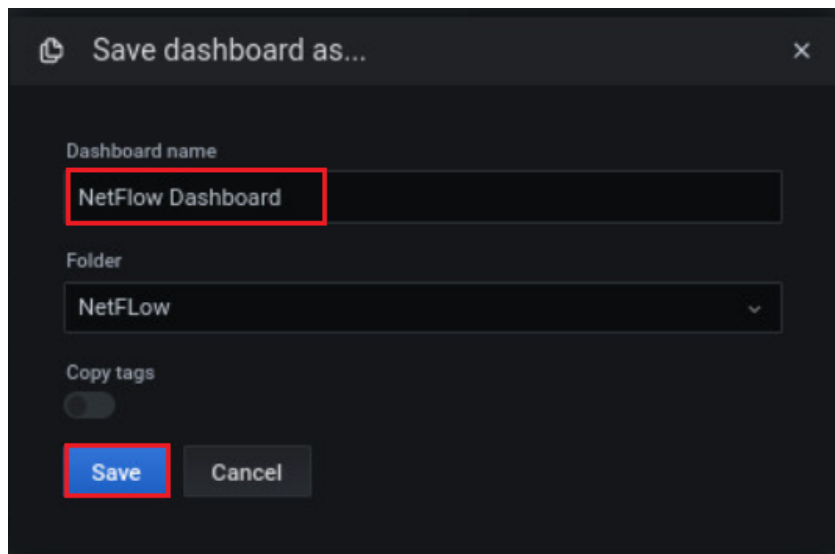


Figure 41. Saving a dashboard.

Step 6. Click on the *add panel* option located on the top right (showed in the following figure).

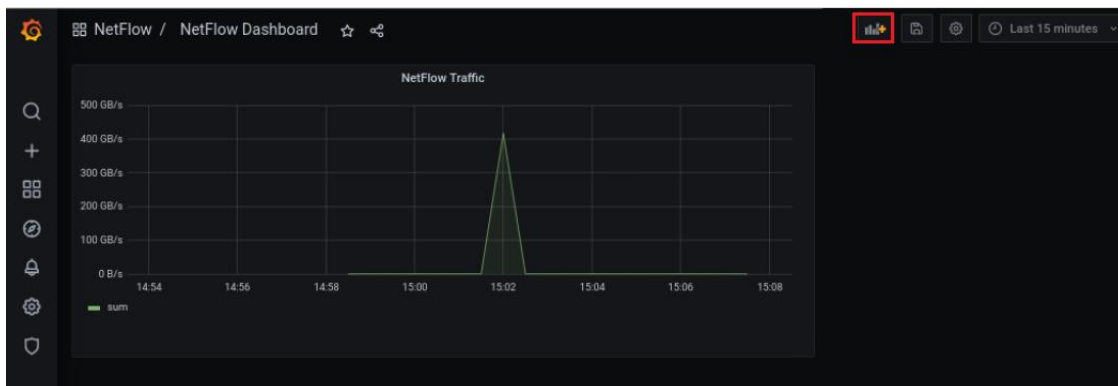


Figure 42. Adding a panel.

The time range of the data is set to 15 minutes. You can also change it to any time range (30 minutes/1 hour) as required.

Step 7. Select *Add an empty panel*.

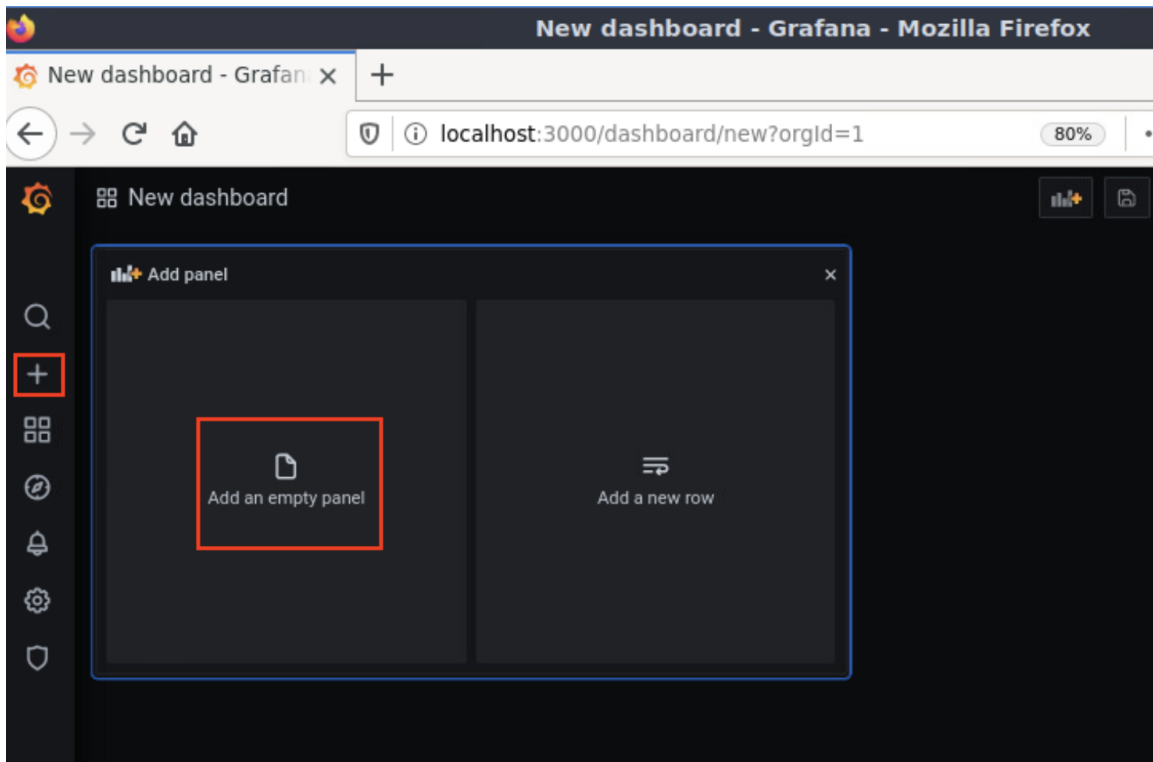


Figure 43. Adding a panel.

Step 8. Change the panel title to *NetFlow Table* from the panel setting.

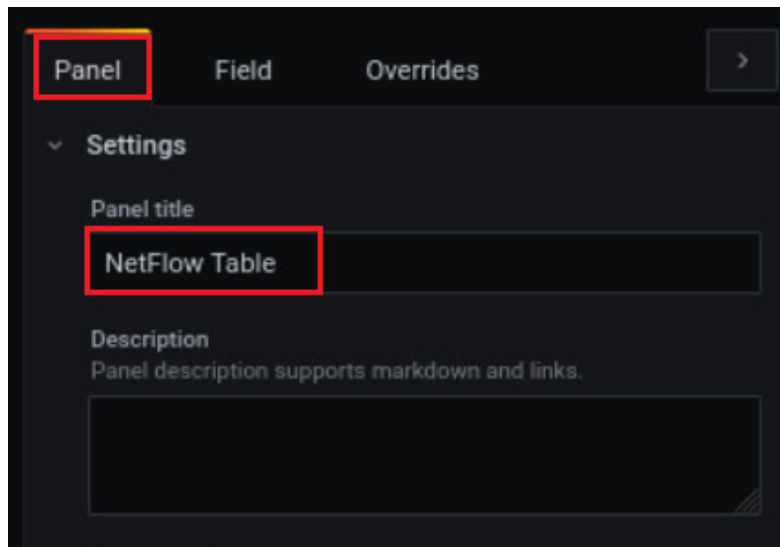


Figure 44. Changing panel title.

Step 9. From panel visualization option, select *Table*.



Figure 45. Customizing a panel.

Step 10. Select *PostgreSQL* as the data source, write the queries and select the format as *Table*. All the steps are highlighted in the following figure.

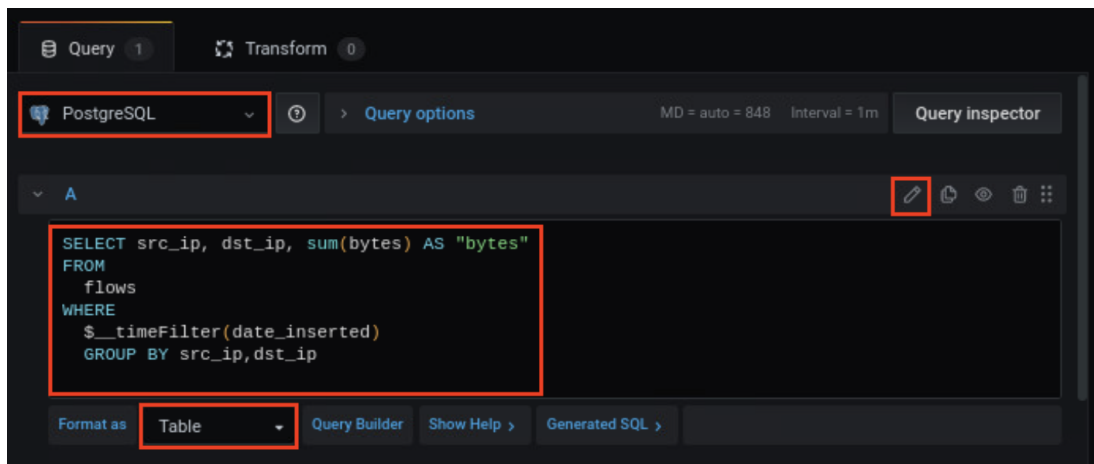


Figure 46. Customizing a panel.

Step 11. Go to the *Field* option, select *Standard options*, and type in the *Unit* entry box *bytes/sec(SI)*.

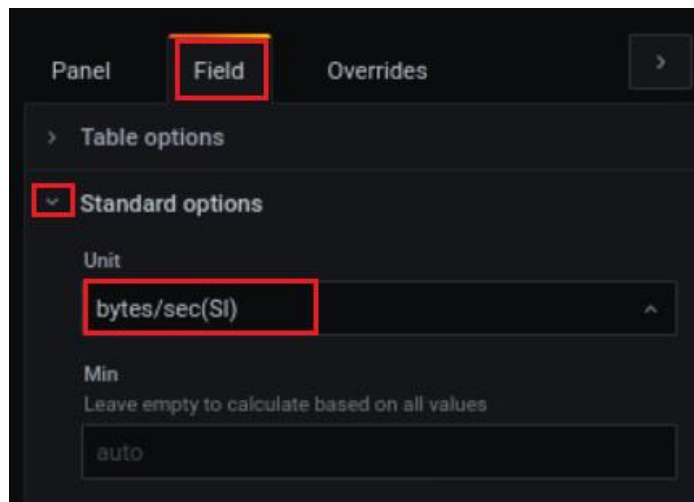


Figure 47. Customizing a panel.

Step 12. The panel will look like the following figure. The table shows source and destination IP addresses and transferred data from the source to the destination. Click on *Apply* to save the panel.

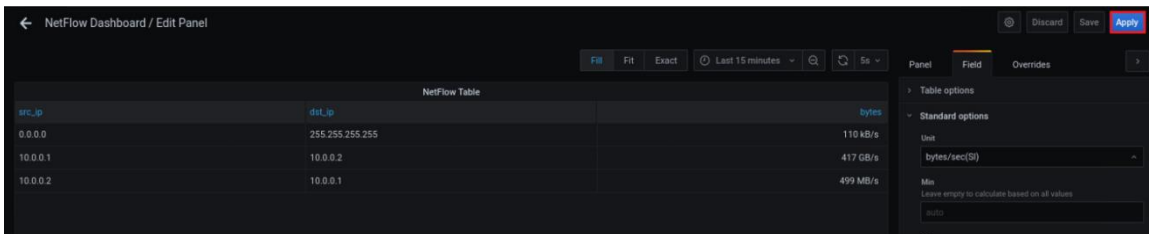


Figure 48. Saving a panel.

Step 13. The NetFlow dashboard will look like the following figure.

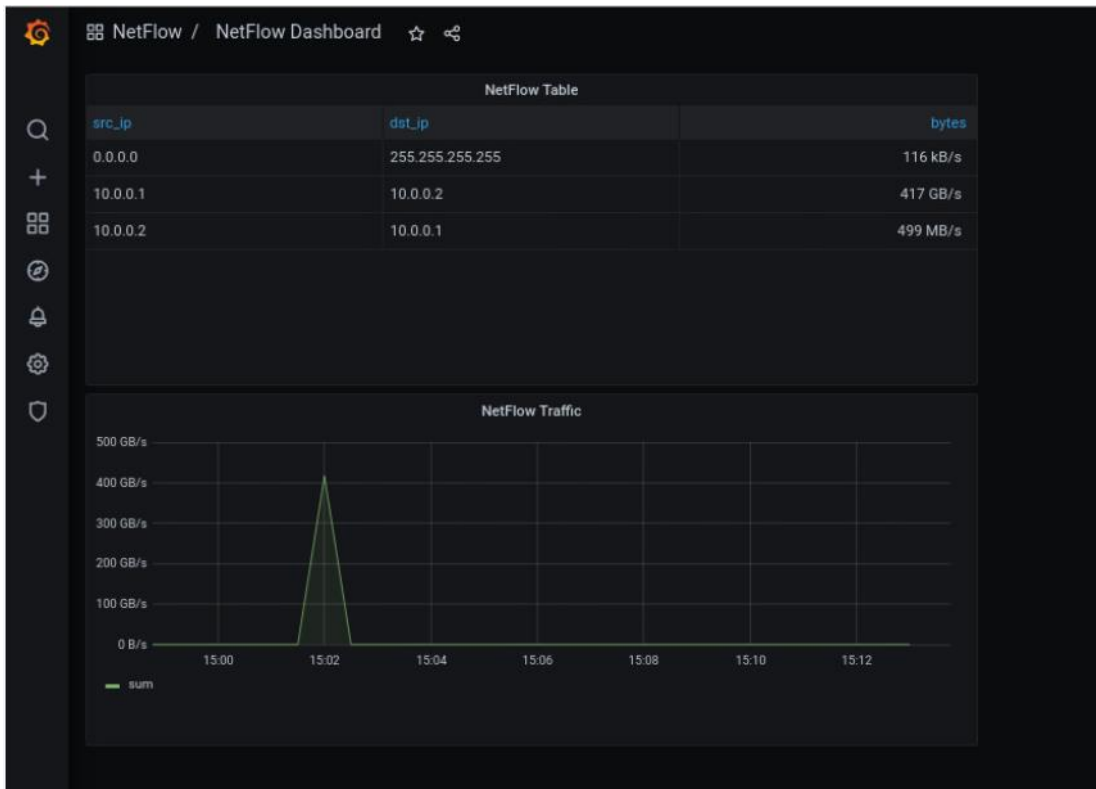


Figure 49. Visualizing the dashboard.

Step 14. Repeat steps 6 and 7 to create a new panel. Change the panel title to *NetFlow Table 2*.

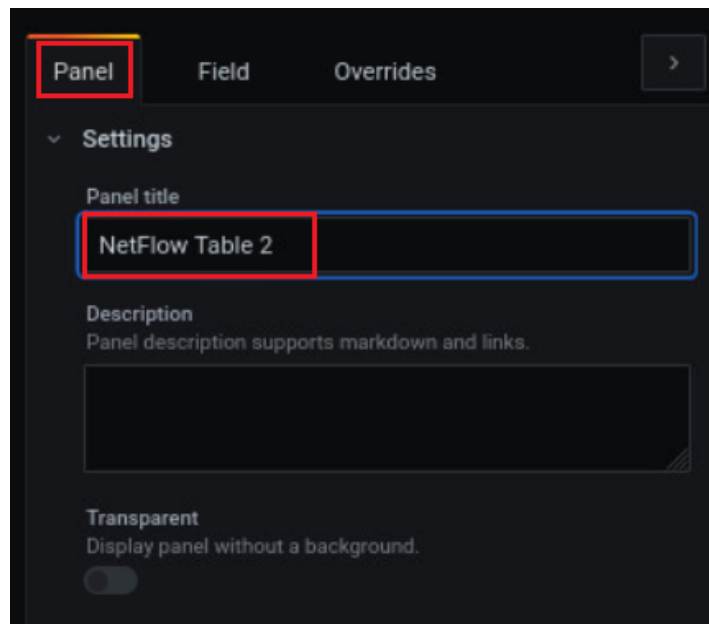


Figure 50. Changing panel title.

Step 15. From panel visualization option, select *Table*.



Figure 51. Customizing a panel.

Step 16. Select *PostgreSQL* as the data source, write the queries and select the format as *Table*. All the steps are highlighted in the following figure.

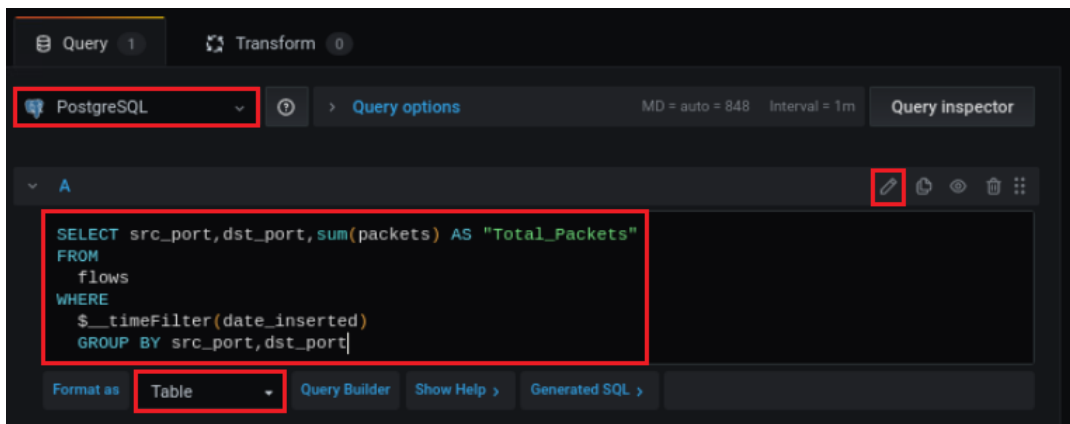


Figure 52. Customizing a panel.

Step 17. The panel will look like the following figure. The table shows source and destination ports and number of packets sent from the source to the destination. Click on *Apply* to save the panel.

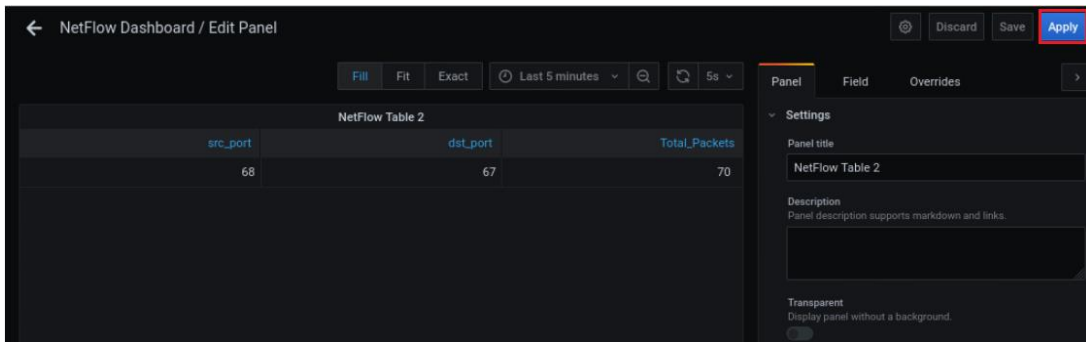


Figure 53. Saving a panel.

Step 18. The dashboard will look like the following figure.

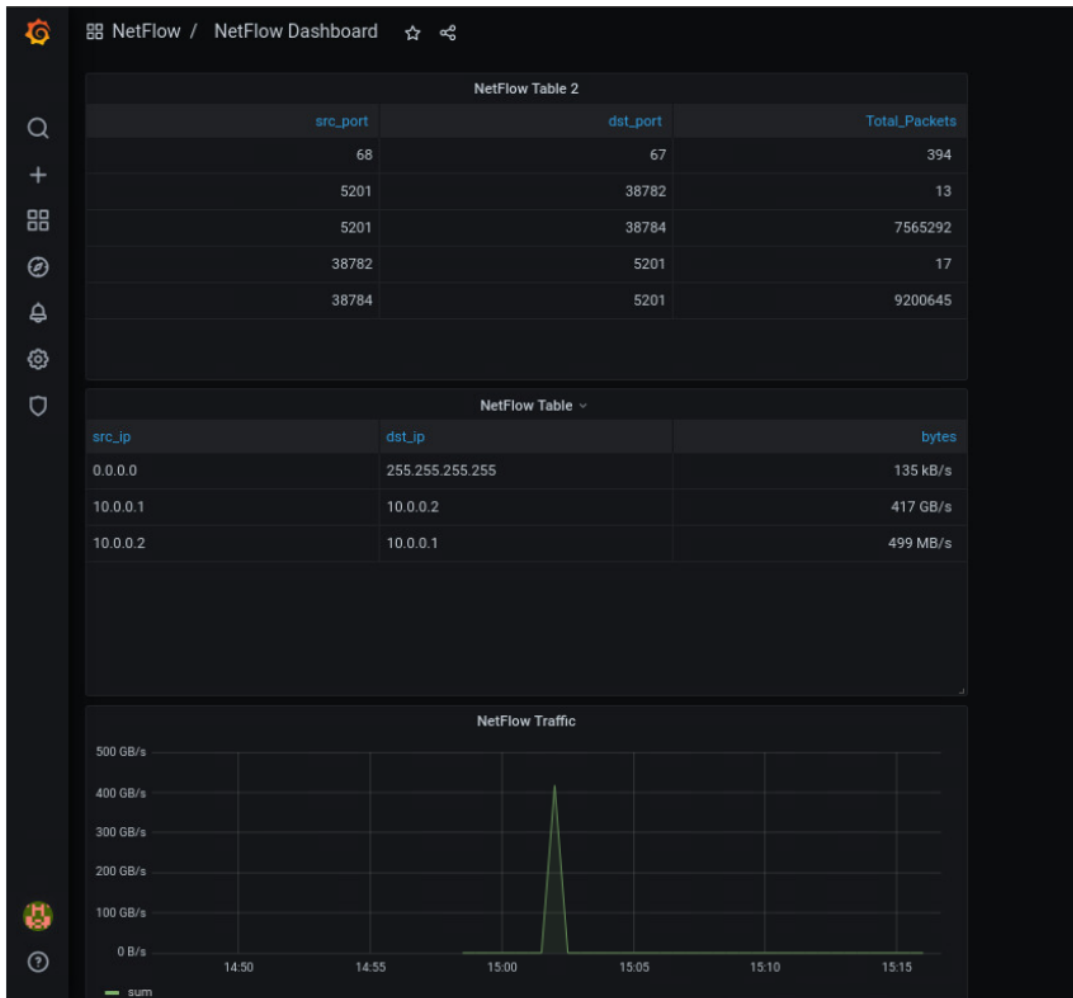


Figure 54. Visualizing the dashboard.

Step 19. You can drag and drop each panel to rearrange the dashboard. After rearranging, the final dashboard will look like the following figure. Save the dashboard.



Figure 55. Visualizing a dashboard.

This concludes Lab 8. Stop the emulation and then exit out of MiniEdit and the Linux terminal.

References

1. IBM, *"Data Visualization"*, [Online].
<https://www.ibm.com/cloud/learn/data-visualization>
2. ThoughtCo., *"Benefits of the Graphical User Interface"*, [Online].
<https://www.thoughtco.com/benefits-of-graphical-user-interface-1206357>
3. Grafana Labs, *"Introduction to Grafana"*, [Online].
<https://grafana.com/docs/grafana/latest/introduction/>
4. GitHub, *"Flow-pipeline"*, [Online].
<https://github.com/cloudflare/flow-pipeline>
5. AWS, *"What is Apache Kafka?"*, [Online].
<https://aws.amazon.com/msk/what-is-kafka/>