Técnicas de Perceção de Redes
Network Awareness

Data Acquisition

Packet Data visualization/acquisition with Wireshark

1. Start a packet capture with Wireshark on your Internet access interface. Generate some traffic by browsing some web pages and accessing video streaming services.

>> Analyze the list of captured packets.

>> Try to identify the packets of the different services/applications/sessions.

>> Save the capture as a pcap or pcapng file.

2. From the captured packets apply different display filters

For more information about Wireshark display filters see: https://wiki.wireshark.org/DisplayFilters

3. Apply a display filter and save only the displayed packets. Use File→ Export Specified Packets… + Displayed + Save.

4. Using the functionality Statistics→ I/O Graph, display the graphs of the traffic (packets or bytes) over time, for the:

(i) total number of packets/bytes,

(ii) number of TCP packets/bytes, with filter “tcp”,

(iii) number of UDP packets/bytes, with filter “udp”,

(iv) number of uploaded packets/bytes, with filter “ip.src==<pc_ipaddr>”,

(v) number of downloaded packets/bytes, with filter “ip.dst==<pc_ipaddr>”,

>> Generate traffic from different applications and usage behaviors, e.g., browsing of simple websites, browsing of complex websites, video steaming, etc…

>> Analyze the different traffic profiles.

>> Close all applications and analyze the profile of your computer background traffic.

>> Try different sampling intervals: 1 second, 0.1 second, 10 seconds, etc…

>> Explore additional traffic filters and applications.

Note: replace <pc_ipaddr> by your PC’s IPv4 address.

5. Identify the end-points of all traffic conversations, grouped by Ethernet address, IPv4 address, IPv6 address, IP address and UDP port, or IP address and TCP port. Use Statistics→ Conversations + (Ethernet/IPv4/IPv6/UDP/TCP).

Packet Data acquisition with tshark/pcap

Identify the name of your Ethernet interface (*int_name*) and your IPv4 address (*pc_ipaddr*) with:

```
ifconfig
ip addr
```

6. Use tshark to capture packets:

```
tshark -i <int_name>
```

7. Use tshark to capture packets while saving data to a specific file:

```
tshark -i <int_name> -w test.pcap
```

8. Use tshark to capture packets while using different **capture filters**:

```
tshark -i <int_name> ip
tshark -i <int_name> tcp
tshark -i <int_name> udp
tshark -i <int_name> “udp and host <pc_ipaddr>”
```

For more information about tshark capture filters see: https://biot.com/capstats/bpf.html

>> Test more filters.

9. Use tshark to read the previously captured packets while using different Wireshark's **reading/display filters**, and write the output to another file:

```
tshark -r test.pcap -w test_filtered.pcap ip
tshark -r test.pcap -w test_filtered.pcap tcp
tshark -r test.pcap -w test_filtered.pcap udp
tshark -r test.pcap -w test_filtered.pcap "udp and ip.addr==<pc_ipaddr>"
```

10. Use tshark to capture packets, generated from your PC, from browsing sessions (TCP port 443) of sites with few multimedia content (text based webpages like forums, e.g., ubuntuforums.org):

```
tshark -i <int_name> -w browsing_light.pcap "host <pc_ipaddr> and tcp port 443"
```

11. Use tshark to capture packets, generated from your PC, from browsing sessions (TCP port 443) of sites with many multimedia contents (e.g., social networks):

```
tshark -i <int_name> -w browsing_heavy.pcap "host <pc_ipaddr> and tcp port 443"
```

12. With Tshark capture the packets from the streaming flows between your PC and YouTube servers during the visualization of videos. Discover your machine IP address (*pc_ipaddr*) and the range of YouTube servers IPv4 addresses for your location (*yt_net*). From UA network, *yt_net* should be (extending the network, and by approximation) 194.210.238.0/24.

```
tshark -i <int_name> -w youtube.pcap "host <pc_ipaddr> and net <yt_net>"
```

Conversation/Flow Data Acquisition

13. Identify and characterize the different conversation between devices, identifying which devices communicated between themselves. Aggregated (by IP address):

```
tshark -r <pcap.file> -z conv,ip -q
```

>> Analyze the extracted individual conversation data.

14. Identify and characterize the different UDP and TCP data flows between devices:

```
tshark -r <pcap.file> -z conv,udp -q
tshark -r <pcap.file> -z conv,tcp -q
```

>> Analyze the extracted individual flow data.

15. To create a clean data file, ordered by the start time of the flow:

```
tshark -r test.pcap  -z conv,ip -q \
    | sed 's/ kB/000/g' | sed 's/ MB/000000/g' | \
    | grep "^[0-9]" | tr -d "[:alpha:],<>-" | sed "s/ \+/ /g" \
    | awk '{print $9, $1, $2, $3, $4, $5, $6, $10}' | sort -n > flows.dat
```

>> Analyze the output file and identify the relevant data/columns.

Note: tshark does not allow to disable human readable values (kB, MB, …), must be converted to integer multiplier.

Flow Data Acquisition with NFStream

16. Using NFStream python package (https://www.nfstream.org/docs/), and the base script baseNFStream.py extract flow information from a previous packet capture.

Create a python virtual environment and install the NFStream package:

```
mkdir nfstream
python -m venv nfstream
source nfstream/bin/activate
pip install nfstream
```

Run the script:

```
python baseNFStream.py -r test.pcap -w flowsNF.dat
```

Type `exit` to terminate the virtual environment.

>> Identify the flow data extracted by NFStream (output file `flowsNF.dat`).

>> Identify the flows you generated, use IPv4 address and TCP/UDP ports to filter them.

17. Develop a python script to load the flow data file to a pandas structure (https://pandas.pydata.org/):

```
import pandas as pd
flows=pd.read_csv('flowsNF.dat')
flows=flows.sort_values(by=['bidirectional_first_seen_ms'])
flows_tcp=flows[flows['protocol']==6]
```

>> Sort the flow by the timestamp of the first packet (*bidirectional_first_seen_ms*).

>> Filter only UDP (`protocol` 17) or TCP (`protocol` 6) flows.

>> Filter only flows initiated (`src_ip`) from a scecific IPv4 address (your client PC).

(Optional – Linux only) Flow Data Acquisition with Suricata

18. Using Docker, pull the latest Suricata image:

```
docker pull jasonish/suricata:latest
```

Run the Suricata Docker, giving administration, special network, and process manipulation permissions. The Suricata log directory will be mapped to the ./logs directory in the host machine.

```
docker run --rm -it --net=host \
           --cap-add=net_admin --cap-add=net_raw --cap-add=sys_nice \
           -v $(pwd)/logs:/var/log/suricata \
           jasonish/suricata:latest -i eth0
```

Note: change the Docker network interface name (eth0) to the one you are using (check with: `ip addr`).

Let the docker start, and generate network flows (browsing, video, etc…). Analyse the Suricata events log JSON file and identify the recorded network flow data:

```
egrep logs/eve.json
```

>> Identify the flows you generated, use IPv4 address and TCP/UDP ports to filter them.

(Optional) Packet Data acquisition with pyshark

Python references: pyshark - Python packet parser using wireshark's tshark, http://kiminewt.github.io/pyshark/

- https://github.com/KimiNewt/pyshark

19. Download and test the basePCap.py script, by capturing all IPv4 packets between a source and destination passed by argument. Source and destination should be an network prefix (no mask for a single machine):

```
python basePCap.py -i eth0 -c <pc_ipaddr> -s 0.0.0.0/0
```

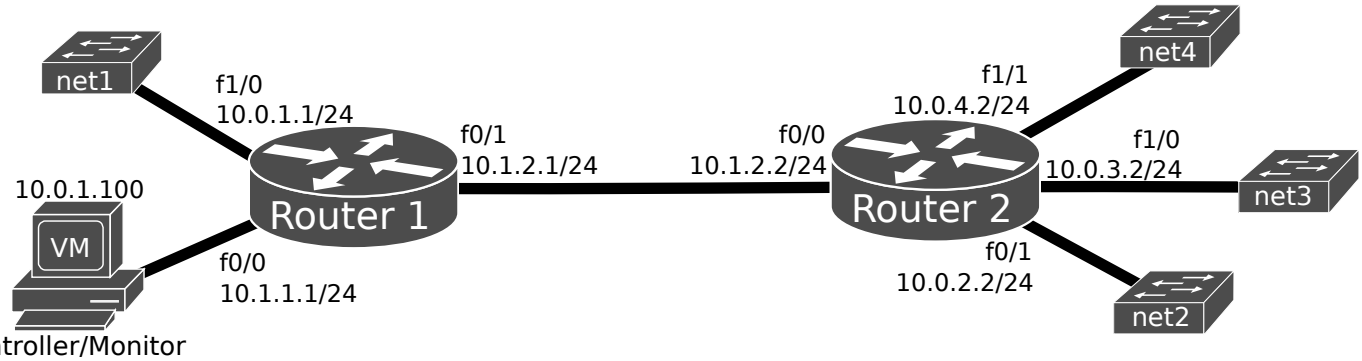
Note: change the capture network interface (eth0) to the one you are using.

20. Test the basePCap.py script, by capturing the HTTPS (port TCP 443) flows between your PC and multiple browsing servers. Discover your machine IP address (*pc_ipaddr*):

```
python basePCap.py -i eth0 -c <pc_ipaddr> -s 0.0.0.0/0 -t 443
```

(Optional) Data acquisition with SNMP

Assemble the network (in GNS3) according to the following figure. The PC should be a VM or the host PC, with Linux (Debian) with Python, SNMP tools, network MIBs and CISCO MIBs. The Routers should be from the **7200** family.



Controller/Monitor

MIB references: MIBS: IF-MIB, IP-MIB, and CISCO-QUEUE-MIB

Python references: *Snimpy* (version >=0.8.14) – API reference, https://snimpy.readthedocs.org/en/latest/api.html

argparse - Parser for command-line options, https://docs.python.org/3/library/argparse.html

matplotlib.pyplot - http://matplotlib.org/api/pyplot_api.html

21. Configure interfaces IPv4 addresses and OSPF routing protocol in both routers. Configuration example for Router 1 (only interface f0/0):

```
Router# configure terminal
Router(config)# interface f0/0
Router(config-if)# ip address 10.0.1.1 255.255.255.0
Router(config-if)# ip ospf 1 area 0
Router(config-if)# no shutdown
Router(config-if)# end
Router(config-if)# write
```

Add VPCs to all networks and test connectivity between all devices.

In both routers, configure a SNMP version 3 community (using the name “private”) with Read-Only permissions, and access with authentication (MD5, password authpass) and encryption (AES128, password: privpass), for user uDDR from group gDDR:

```
Router(config)# snmp-server user uDDR gDDR v3 auth md5 authpass priv aes 128 privpass
Router(config)# snmp-server group gDDR v3 priv
Router(config)# snmp-server community private RO
```

22. Download and test the baseSNMP.py script, and understand how different MIB objects can be accessed.

```
python baseSNMP.py -r 10.0.0.2
```

Note that relevant MIBS should be place on the script’s sub folder “./mibs/”.

Use the following MIB objects to access relevant interface traffic statistics:

- *ifHCOutUcastPkts*, *ifHCInUcastPkts*, *ifHCOutOctets*, *ifHCInOctets* from IF-MIB.

23. Create a time loop, with periodicity given by argument, and retrieve interface statistics. Display, and store, byte and packet increments (in both directions) with timestamp.

(Optional) Data acquisition with NetFlow/IPFIX

NetFlow references: NetFlow Overview

NetFlow Export Datagram Format (version 1 and 5 formats)

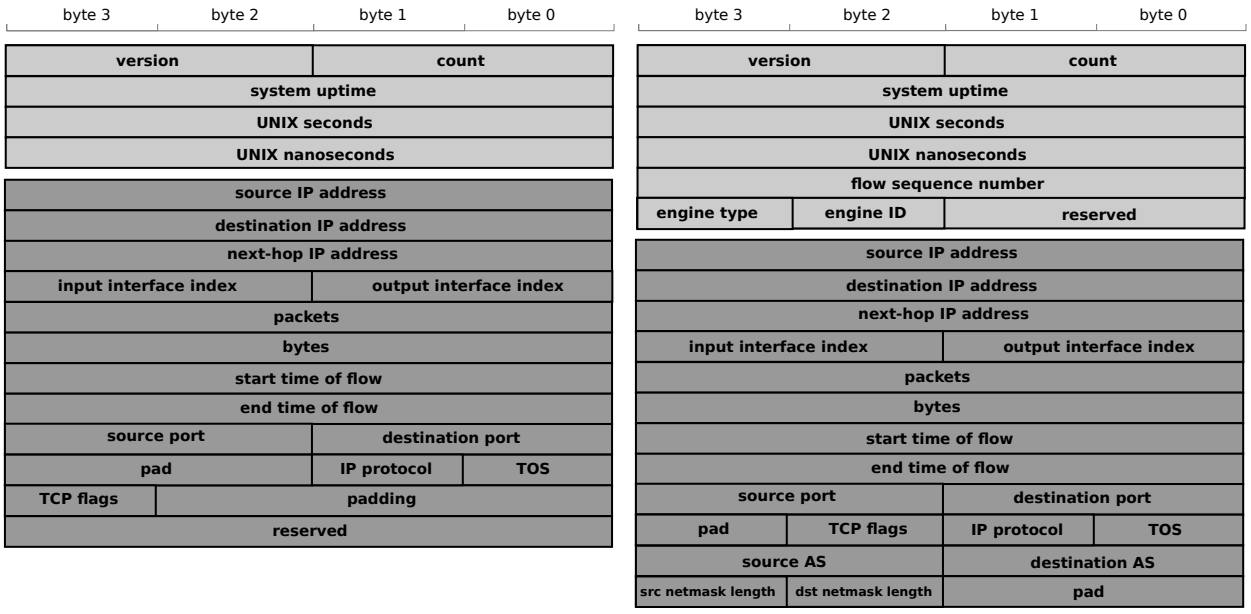
Python references: *socket* - Low-level networking interface, https://docs.python.org/2/library/socket.html

struct - Interpret strings as packed binary data, https://docs.python.org/2/library/struct.html

netaddr IPAddress and IPNetwork - https://netaddr.readthedocs.org/en/latest/tutorial_01.html

netaddr IPSet - https://netaddr.readthedocs.org/en/latest/tutorial_03.html

NetFlow v1 and v5 header and body formats



24. Configure Router2 to export (to PC VM) the flow statistics using NetFlow version 1 for all traffic egressing interface

f0/1.

```
Router2(config)# interface FastEthernet0/1
Router2(config-if)# ip flow egress
Router2(config)# ip flow-export destination 10.0.1.100 9996
Router2(config)# ip flow-export source Loopback 0
Router2(config)# ip flow-export version 1
```

25. Download and test the baseNetFlow.py script, generating traffic to and from terminals (VPCS) in networks net2, net3, and net4. Understand how the NetFlow packet is received and how data fields can be accessed.

```
python baseNetFlow.py -r 10.0.0.2 -n 10.0.2.0/24 10.0.3.0/24 10.0.4.0/24
```

26. Complete the code to retrieve the relevant NetFlow version 1 data and periodically infer the traffic matrix. Configure the required additional NetFlow export commands in Router1 and Router2.

Note: consider using Python library *netaddr* classes IPAddress, IPNetwork, and IPSet.

Note2: consider the output format (line):

```
timestamp_1, Trf_Net1_Net2,  Trf_Net1_Net3, …, Trf_Net4_Net2, Trf_Net4_Net3
timestamp_2, Trf_Net1_Net2,  Trf_Net1_Net3, …, Trf_Net4_Net2, Trf_Net4_Net3
...
```

27. Include support to NetFlow version 5.

Change routers' configurations to export flow data using NetFlow version 5:

```
Router(config)# ip flow-export version 5
```