Network Capturing and Manipulation Introduction to Wireshark and Scapy

Gilles Callebaut
gilles.callebaut@kuleuven.be
Fan Wu
fan.wu@kuleuven.be
Emanuele Peschiera
emanuele.peschiera@kuleuven.be

Ghent Technology Campus, KU Leuven
Monday 5th February, 2024
v1.12

This document is subjected to change, please consult the most recent version at: github.com/dramco-edu/Network-Capturing-and-Manipulation----Introduction-to-Wireshark-and-Scapy-Industrial-Automation

Revision History

Revision	Date	Author(s)	Description
1.0	25/04/2019	GC	Created
1.1	01/04/2019	GC	Revised deadlines
1.2	29/04/2019	GC	Clarify Question 4 and Update Listing 6
1.3	12/03/2020	GC	Updated deadlines and removed Anaconda
1.4	15/03/2020	GC	Adjusted so the lab session could be done at home
1.5	29/03/2020	GC	Added example scapy project set-up Section 3
1.6	18/04/2020	GC	Updated questions with more details and report description
1.6.1	20/04/2020	GC	Update Q8 and the report
1.7	08/03/2021	GC	Added case studies
1.8	16/09/2021	GC	Updates case studies
1.9	03/01/2022	GC	Moved from reports to small intermediate tests at the end of each session.
1.10	09/11/2022	GC	Extended the tracerouting use case and explicitly mentioned that students need to communicate with their teaching assistant and not only me.
1.11	14/12/2022	GC	Added formatting requirement for the final `.zip` file.
1.12	01/02/2024	GC	Updated for course Industrial Automation

Contents

1 Introduction

One's understanding of network protocols can often be greatly deepened by "seeing protocols in action" and by "playing around with protocols" observing the sequence of messages exchanged between two protocol entities, delving down into the details of protocol operation, and causing protocols to perform certain actions and then observing these actions and their consequences. This can be done in simulated scenarios or in a "real" network environment such as the Internet.

In these labs you will be observing as well as manipulating the network protocols in your computer "in action" interacting and exchanging messages with protocol entities executing elsewhere in the Internet. You could think of a network packet analyzer as a measuring device used to examine what's going on inside a network cable, just like a voltmeter is used by an electrician to examine what's going on inside an electric cable. We will focus on Wireshark to monitor packets and Scapy, a Python library, to manipulate them.

The first goal of this lab is to introduce you to Wireshark and get you acquainted with it's interface. The following questions will demonstrate that you have been able to get Wireshark up and running, and have explored some of its capabilities. But first, install Wireshark and capture some data!

But before delving into the lab session, take a look at the evaluation of this course detailed in Section 7. In this way, you know how to proceed with the lab session and are aware of the timeline and expectations.

2 Getting to know Wireshark

In this first Wireshark lab, you will get acquainted with Wireshark, and make some simple packet captures and observations. The basic tool for observing the messages exchanged between executing protocol entities is called a packet sniffer. As the name suggests, a packet sniffer captures ("sniffs") messages being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured messages. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself. Similarly, received packets are never explicitly addressed to the packet sniffer. Instead, a packet sniffer receives a copy of packets that are sent/received from/by application and protocols executing on your machine.

Question 1

Is Wireshark able to capture packets that are not addressed to your computer?

2.1 Installing Wireshark

In order to run Wireshark, you will need to have access to a computer that supports both Wireshark and the libpcap or WinPCap packet capture library. In case you are unable to see any interfaces when opening Wireshark, the libcap software is probably not installed. Usually, the libpcap software will be installed for you when you install Wireshark.

Question 2

What is the purpose of the libcap software?

2.2 Running Wireshark

When you run the Wireshark program, you'll get a startup screen that looks something like the screen below. Different versions of Wireshark will have different startup screens – so don't panic if yours doesn't look exactly like the screen below!

There's not much interesting on this screen. But note that under the Capture section, there is a list of so-called interfaces. Be sure to locate the interface on the startup page through which you are getting Internet connectivity, e.g., mostly likely a WiFi or Ethernet interface, and select that interface.

Question 3

Try to explain the different interfaces displayed on the first screen. You can find out more about the interfaces by running `Get-NetAdapter -IncludeHidden` in Windows PowerShell.

Let's take Wireshark out for a spin! If you click on one of these interfaces to start packet capture (i.e., for Wireshark to begin capturing all packets being sent to/from that interface), a screen like the one below will be displayed, showing information about

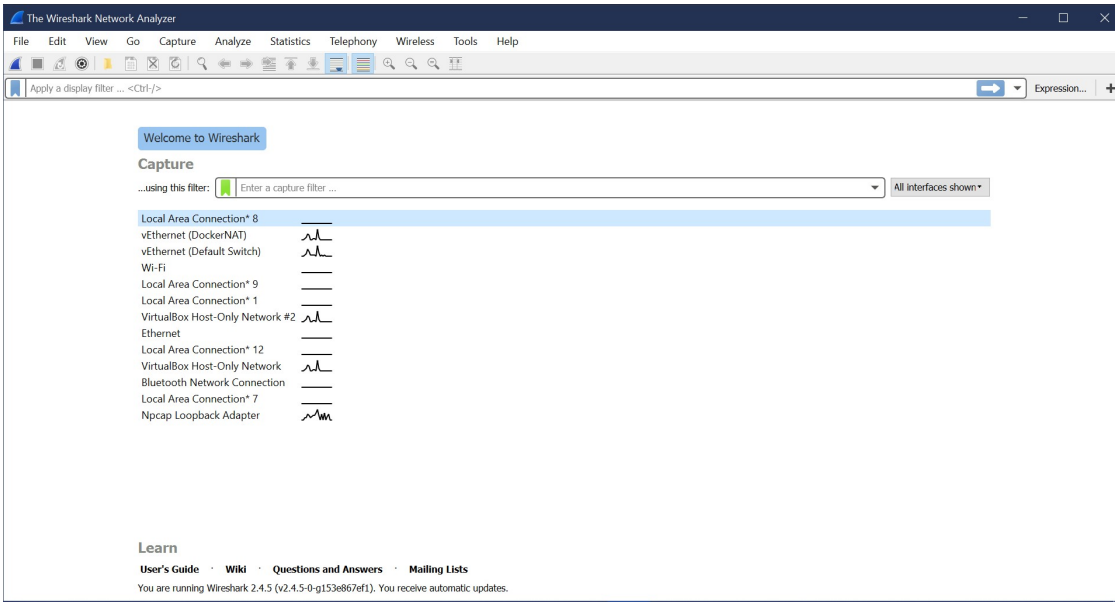

Figure 1: Initial Wireshark Screen

the packets being captured. Once you start packet capture, you can stop it by using the Capture pull down menu and selecting Stop.

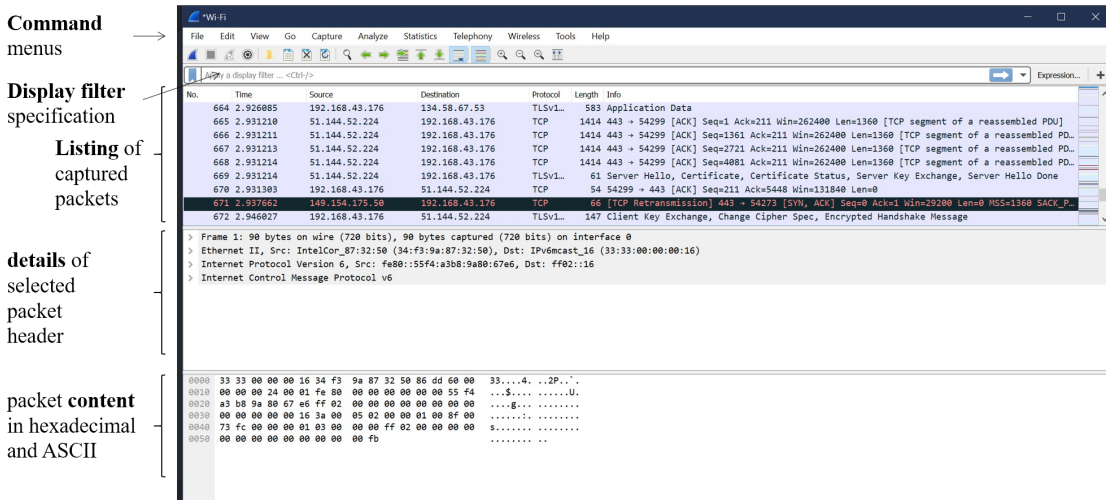

Figure 2: Wireshark Graphical User Interface, during packet capture and analysis.

This looks more interesting! The Wireshark interface has five major components:

- The **command menus** are standard pulldown menus located at the top of the

window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data, and exit the Wireshark application. The Capture menu allows you to begin packet capture.

- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is not a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest-level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.

- The **packet-header details window** provides details about the packet selected (highlighted) in the packet-listing window. (To select a packet in the packetlisting window, place the cursor over the packet's one-line summary in the packet-listing window and click with the left mouse button.). These details include information about the Ethernet frame (assuming the packet was sent/received over an Ethernet interface) and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the caret signs to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest-level protocol that sent or received this packet are also provided.

- The **packet-contents window** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.

- Towards the top of the Wireshark graphical user interface, is the **packet display filter field**, into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows). In the example below, we'll use the packet-display filter field to have Wireshark hide (not display) packets except those that correspond to HTTP messages.

2.3 Taking Wireshark for a Test Run

1. Start up your favorite web browser, which will display your selected homepage.

2. Start up the Wireshark software. You will initially see a window similar to that shown in Figure 1. Wireshark has not yet begun capturing packets.

3. Youll see a list of the interfaces on your computer as well as the current activity on that interface. Click on the interface on which you want to begin packet capture. Packet capture will now begin – Wireshark is now capturing all packets being sent/received from/by your computer!

4. Once you begin packet capture, a window will appear showing the packets being captured as also shown in Fig. 2. By selecting Capture pulldown menu and selecting Stop, you can stop packet capture. But dont stop packet capture yet. Let's capture some interesting packets first. To do so, we'll need to generate some network traffic. Lets do so using a web browser, which will use the HTTP protocol that we will study in detail in class to download content from a website.

5. While Wireshark is running, enter the URL: `http://gaia.cs.umass.edu/wireshark-labs/INTRO-wireshark-file1.html` and have that page displayed in your browser. In order to display this page, your browser will contact the HTTP server at `http://gaia.cs.umass.edu` and exchange HTTP messages with the server in order to download this page. The Ethernet frames containing these HTTP messages (as well as all other frames passing through your Ethernet adapter) will be captured by Wireshark.

6. After your browser has displayed the `INTRO-wireshark-file1.html` page (it is a simple one line of congratulations), stop Wireshark packet capture by selecting stop in the Wireshark capture window. The main Wireshark window should now look similar to Figure 3. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP message exchanges with the `gaia.cs.umass.edu` web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well (see, e.g., the many different protocol types shown in the Protocol column in Figure 3). Even though the only action you took was to download a web page, there were evidently many other protocols running on your computer that are unseen by the user. Be aware that there is often much more going on than "meets the eye"!

7. Type in "http" (without the quotes, and in lower case all protocol names are in lower case in Wireshark) into the display filter specification window at the top of the main Wireshark window. Then select Apply (to the right of where you entered "http"). This will cause only HTTP message to be displayed in the packet-listing window.

Question 4

- List **ten different protocols** that appear in the protocol column in the unfiltered packet-listing window.
- Explain the **most important fields** of, at least, three of these protocols. Thereby **illustrating why** these fields are **necessary** to **fulfill** the protocol's **function**. An example is the window size in TCP to throttle the number of packets sent, so the receiver is not overloaded.
- What is the **purpose** of these ten protocols and **where** are they located in the **OSI model**?
- Include a readible screenshot (Wireshark) of each protocol including its field.

8. Find the HTTP GET message that was sent from your computer to the `gaia.cs.umass.edu` HTTP server. (Look for an HTTP GET message in the "listing of captured packets" portion of the Wireshark window that shows "GET" followed by the `gaia.cs.umass.edu` URL that you entered. When you select the HTTP GET message, the Ethernet frame, IP datagram, TCP segment, and HTTP message header information will be displayed in the packet-header window.

Question 5

How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received?[a]

[a]By default, the value of the Time column in the packet listing window is the amount of time, in seconds, since Wireshark tracing began. To display the Time field in time-of-day format, select the Wireshark View pull down menu, then select Time Display Format, then select Time-of-day.

Question 6

What is the Internet address of the `gaia.cs.umass.edu` (also known as `www-net.cs.umass.edu`)? What is the Internet address of your computer?[a]

[a]Validate the displayed Internet address in Wireshark by comparing it to your own Internet address through `ipconfig`.

Congratulations! You are now a Wireshark-expert.

3 Getting to know Scapy

After monitoring packets in Wireshark, let's manipulate some packets with Scapy!

Scapy is a powerful interactive packet manipulation program. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. Scapy can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery. It can replace hping, arpspoof, arp-sk, arping, p0f and even some parts of Nmap, tcpdump, and tshark.

Scapy can be run in two different modes, interactively from a terminal window and programmatically from a Python script. In these sessions, we will primarily use scripts. However, the interactive shell is useful to consult the documentation and access the `ls` function. I advise you to open a terminal to use the functions described in Section A to better understand the code.

Try to verify your Scapy code with Wireshark.

3.1 Installing Scapy

Follow the installation guide of Scapy: `https://scapy.readthedocs.io/en/latest/installation.html`. We suggest creating one virtual environment for this lab, containing all necessary packages.

Question 7

Open a Terminal and open the Scapy interactive shell. Try to find the default interface by executing `conf.iface`. Is this the correct interface?

3.2 Creating Packets with Scapy

Packets consists of different nested layers, cfr. OSI-model. This is depicted in Figure 3. Equivalently, stacking layers is done by using the `'/'` operator, as illustrated in Listing 1.

```
from scapy.all import *

packet = Ether()/IP()/TCP()
```

Listing 1: Illustration of stacking layers in Scapy.

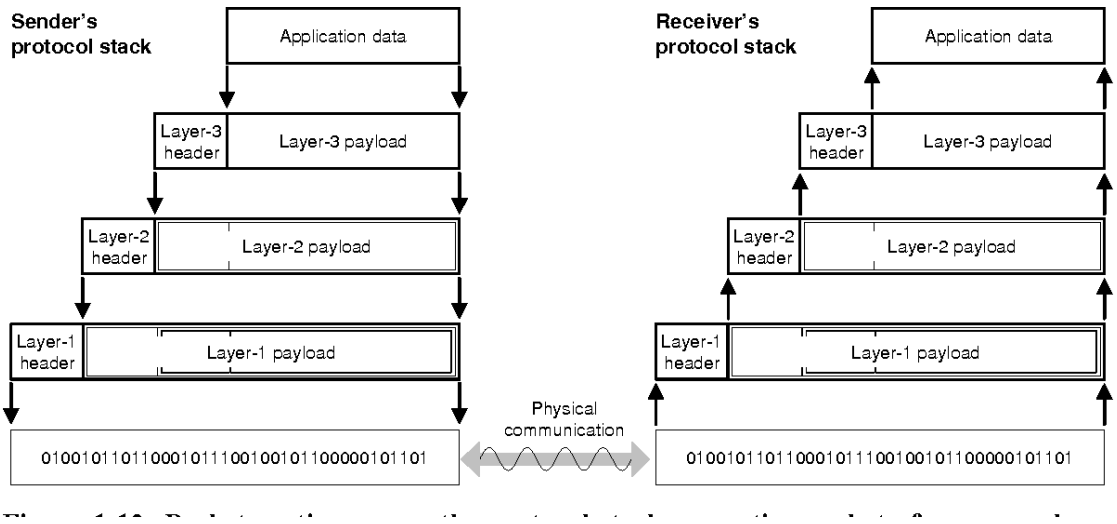

Figure 3: A Packet consists of different nested layers

3.2.1 Sending Packets

Scapy makes a distinction between sending packets at Layer-2 (L2) and Layer-3 (L3). For instance, a L3 packet is sent with the `send()`, while `sendp()` is used for L2 packets. To be able to receive answers from the network, you need to use the `sr()` and `srp()` methods. They return a tuple of two lists. The first element is a list of couples (packet sent, answer), and the second element is the list of unanswered packets. These two elements are lists, but they are wrapped by an object to present them better, and to provide them with some methods that do most frequently needed actions, e.g. `show()`.

Info: Before checking the listing below, try to understand the ICMP protocol.

The functions `sr1()` and `srp1()` are variants that only return one packet that answered the packet sent. If there is no response, a `None` value will be returned when the timeout is reached.

```python
from scapy.all import *

ans, unans = sr(IP(dst="192.168.0.155")/ICMP(), timeout=5)
ans.show()
unans.show()

for s,r in ans:
    # iterate over send and received messages
```

Listing 2: Illustration of sending and receiving packets in Scapy.

Question 8

Use wireshark to monitor the result of `ping -4 google.com` and explain. What is the purpose of the `-4` argument? Use `srloop()` or `srploop()` to replicate such a ping request.[a] Validate the content of the DNS response in Wireshark due to pinging "google.com".

[a] Specify the `id` field (e.g., 1), because the default value is zero and is often blocked by firewalls.

3.2.2 Creating packets at the Application Layer

An example of an HTTP GET request is shown in Listings 3. It demonstrates how easy it is to include the application layer.

```python
from scapy.all import *

# Build our packet layer by layer
server = 'google.com'

packet = Ether(src='00:00:00:11:11:11')
print(F'Ethernet: {repr(packet)}\n')

ip = packet/IP(dst=server)
print(F'IP: {repr(ip)}\n')

tcp = ip/TCP(dport=80)
print(F'TCP: {repr(tcp)}\n')

http = tcp/"GET /index.html HTTP/1.0\r\n\r\n"
print(F'HTTP-1: {repr(http)}')

# Build the packet in one step
http = Ether()/IP(dst=server)/TCP(dport=80)/"GET /index.html HTTP/1.0\r\n\r\n"
print(F'HTTP-2: {repr(http)}')
```

Listing 3: Creating an HTTP GET request.

4 Resolve a Domain Name

Try to complete the **DNS query** code in Listing 4 and change the source port to random (allowed) number. if you are connected to a KU Leuven network, change the IP address of the DNS server to the current used DNS server by your computer because external DNS servers are blocked. The DNS server used by your computer can be found by monitoring DNS queries in Wireshark or by executing `ipconfig \all` in your terminal.

Question 9

Explain the choice of **variables**, e.g., which DNS type did you use and why and which source port did you use and why is it allowed to be used by user applications? Validate the content of the DNS response in Scapy with the DNS response shown in Wireshark.

Question 10

Why is the `rd` parameter specified in the DNS packet? Use `ls(DNS)` to check its default value.

```
16 from scapy.all import *
17
18 # FILL IN THE FOLLOWING VARIABLES
19 dst_ip_address =                  # Use IP address of Google Public DNS or local DNS
20 sends_over_udp =                   # False or True?
21 dest_port =                           # Port DNS listens to
22 src_port =                                  # Port where the query was sent from (make
   ↪  random allowed port)
23 site_name =                           # name of website
24 layer_two =                          # True of False? L2 or L3?
25 dns_transaction_id =          # generate random  16-bit transaction id
26 dns_qtype =                           # DNS query type
27
28
29 # DO NOT TOUCH THIS CODE, but feel free to check out the code :)
30 network_layer = IP(dst=dst_ip_address)
31 transport_layer = UDP() if sends_over_udp else TCP()
32 transport_layer.dport = dest_port
33 transport_layer.sport = src_port
34 app_layer = DNS(id=dns_transaction_id, rd=1,qd=DNSQR(qname=site_name,qtype= dns_qtype))
35
36 dns_query = network_layer/transport_layer/app_layer
37 print(dns_query.summary())
38
39 #send query and wait for 1 response
40 if layer_two:
41         response = srp1(dns_query)
42 else:
43         response = sr1(dns_query)
44 print(response.summary())
```

Listing 4: DNS Query

5 Make an ARP Request to a Friend

Question 11

- On which layer does ARP operate?
- Why is this protocol needed?
- What is an ARP table?
- What is your current ARP table and explain (what are the types)?
- How can you broadcast a link layer frame?
- Why do we need a MAC address of the other machine if we have the IP address of that machine?

Question 12

Try to resolve the MAC address of a system in your LAN[a] using the example code from Listing 5.

[a] You can easily find connected systems to your local network by navigating to your default gateway or by using other services provided by your ISP, e.g., `mijn.telenet.be`. To find the IP address of your default gateway, execute `ipconfig` in your terminal.

```python
from scapy.all import *

# Fill in the MAC address for ARP request
mac_addr = ""

# Fill in the IP address of the machine
ip_addr = ""

# Are we sending at L2 or L3? (bool)
# Explain why!
layer_two =

# The following line creates an Ethernet frame (Layer 1)
# An ARP message operates on top of an Layer 1 frame.
arp_frame = Ether(dst=mac_addr) / ARP(op=1, pdst=ip_addr)

# Send the frame and wait for answers
if layer_two:
        resp, unans = srp(arp_frame)
else:
        resp, unans = sr(arp_frame)

for s, r in resp:
    print(r[Ether].src)
```

Listing 5: ARP

6 Stealth Scanning

Stealth scanning exploits the TCP three way handshake to discover if ports are open on an Internet-accessible computer. In order to understand the working principle of this technique, study the TCP three way handshake and include this in you report.

Question 13

Extend the example code for the stealth attack in order to discover open ports on servers. Try to use the `report_ports` function of Scapy to evaluate the results of your code.

```python
import socket  # to convert port number to protcol/service name

import numpy as np
from scapy.all import *

SYNACK = 'SA'


ip_range = # add list of IP addresses
port_range = # add list of port ranges


def scan_port(target, port):
    synack_pkt = sr1(IP(dst=target)/TCP(sport=RandShort(),
                                        dport=port, flags="S"), timeout=1,
                                        ↪  verbose=False)
    if(synack_pkt):
        flag = synack_pkt['TCP'].flags
        if flag == SYNACK:
            return True
    return False
    # sending a RST packet (to halt the handshake) is not needed because the OS will do
    ↪  this for us,
    # because it has no knowledge of sensing a SYN packet to that server
    # TODO check in wireshark if this is the case


def is_alive(target):
    # TODO use ARP to see if the host is alive

for target in ip_range:
    if is_alive(target):
        print("{} is alive".format(target))
        for port in port_range:
            if(scan_port(target, port)):
                    protocol = socket.getservbyport(port, 'tcp')
                    print(f"\t {target} {port} {protocol}")
```

Listing 6: Stealth Scanning

7 Evaluation

7.1 Small intermediate tests

The lab sessions will be evaluated by means of small tests at the beginning of several lab sessions. This assignment contains several questions which can help the students to assess how well they have mastered the subjects. Note that the student needs to be able to show the programming code when requested by the teaching assistant for evaluation.

7.2 Written report (Question 4)

In addition to the tests, Question 4 has to be executed, reported, and uploaded to Toledo. It must be written in English.

- Use the naming format: `Q4_<firstname>_<lastname>.pdf`.
- Include your name, class and year.

The following report structure is advised:

- Title, name, year, date,...
- Table of contents
- Introduction
  - What?
  - Why?
  - Explain the difference between L2 and L3 communication.
- Overview of protocols with their details (see *Q 4*)
- Conclusion(s)
- References (referenced in the report)
  More information regarding citing other sources can be found in this video: `vimeo.com/220916942`.

A LaTeX template and tips-and-tricks can be found here: `github.com/dramco-edu/LaTex`. More general tips-and-tricks for writing reports can be accessed

here: dramco-edu.github.io/Thesis-Tips-and-Tricks/ or https://github.com/DRAMCO/writing-scientific-papers-in-latex-tips-and-tricks.

The report is expected to be structured and styled as described in:

- Hoogenboom BJ, Manske RC. How to write a scientific article. Int J Sports Phys Ther. 2012 Oct;7(5):512-7. PMID: 23091783; PMCID: PMC3474301.
- Kallestinova ED. How to write your first research paper. Yale J Biol Med. 2011 Sep;84(3):181-90. PMID: 21966034; PMCID: PMC3178846.

7.3 Timeline

Deviations from the following timeline can occur and will be communicated during the lab sessions.

Lab 1

- Sections 1, 2 and 3 need to be fully done.

Lab 2

- Small test about lab session 1 at the start of lab 2.
- Sections 4, 5 and 6 need to be fully done.

Lab 3

- Small test about lab sessions 2 at the beginning of lab 3.
- Report on *Q4*.

End of semester

- Send in your report on *Q4*. The deadline will be communicated through Toledo.

A Scapy interactive shell

Scapy's interactive shell is run in a terminal session. Root privileges are needed to send the packets. However, in these lab sessions, you are primarily going to use these functions to display information concerning default values or available layers and fields.

A.1 Entering the Scapy Environment

The Scapy environment is accessible by executing `scapy` in a terminal:

```
>scapy
```

You can go out the environment with the `exit()` command.

A.2 Available Commands

You can check the available commands in Scapy through running `lsc()`. In the following subsections, we will discuss the commands which we will be using in these lab sessions.

A.2.1 Help

The `help()` function is a wrapper around pydoc.help[1] that provides a helpful message when 'help' is typed at the Python interactive prompt.

- Calling `help()` at the Python prompt starts an interactive help session.
- Calling `help(thing)` prints help for the python object 'thing'.

A.2.2 List Available Protocols

You can list the available protocols and fields for each layer by using `ls()`. If you run this command without an argument, all the supported protocols are listed. When including a protocol as the argument, e.g. `ls(ARP)`, the fields and their defaults will be displayed.

[1]The pydoc module automatically generates documentation from Python modules. The documentation can be presented as pages of text on the console, served to a Web browser, or saved to HTML files