CYB 220-T01

Final Project

Professor Grech

Mikayla Hubbard

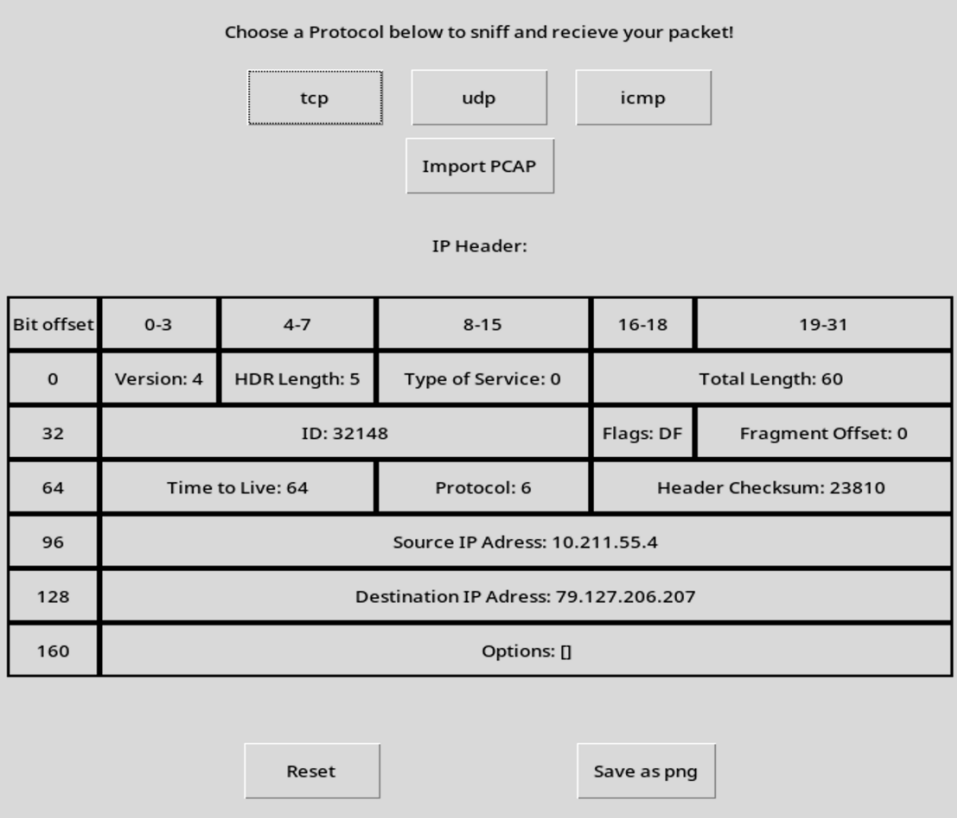
May 5, 2025

Table of Contents

Overview

The goal of this program is to allow the user to sniff or import packets and in turn receive a graphed version of that packet's IP protocol header. The user can choose to sniff network traffic for either TCP, UDP, or ICMP. They could also import a PCAP. The program then captures the first packet it finds and graphs the information. This is meant to display packet header information in a way that helps users visually understand the IP headers.

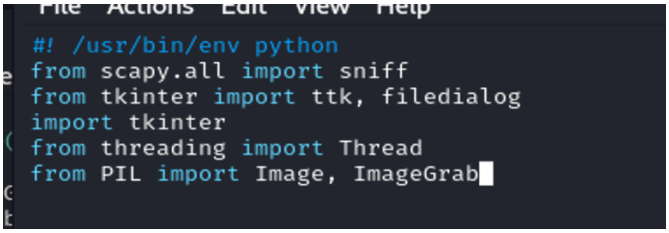
^ this is the initial screen. I will show the output of what happens when the user clicks the TCP button (further walkthrough will be in the demo video)



^ after sniffing, this is the resulting graph. Each section of the header data is sorted into boxes that are proportional to the amount of bits each value takes up. You can then click the "save as png" button, and a screenshot of the graph will be saved to your directory. If you click the "Reset" button, the graph will disappear and you could sniff for a different packet. Further explanation and demonstration of the features working is in the demo video.

Technical Details

I shall now walk through the various important portions of the code.



^ the imports: I used scapy for sniffing and catching the values of a packet, tkinter for visuals, threading for the timing of the “sniffing…” label, and ImageGrab for the screenshot of the graph.

```
class PacketGUI:
    def __init__(self, root):

        self.num_captures=0
        #create a frame for the text
        self.textFrame = ttk.Frame(root)
        #add columns to the frame - 2 equal columns
        self.textFrame.columnconfigure(0, weight = 2)
        self.textFrame.columnconfigure(1, weight = 2)

        # put a label into the frame
        self.text = ttk.Label(self.textFrame, wraplength=500, anchor="center",
                    text = "Choose a Protocol below to sniff and recieve your packet!")
        #grid it in into the frame, with a columnspan so it is centered
        self.text.grid(row=0, column=0, columnspan = 2, sticky = "wens", padx=70, pady=20)

        #pack the text frame
        self.textFrame.pack()
```

^ the program is set up in a class called PacketGUI. In this screenshot you can see the num_captures counter, which is used later for naming the screenshots we take. You also see the first Frame: a text frame which holds the “Choose a Protocol…” message.

```
#button frame - like the textFrame, but 3 equal cols
self.btnFrame = ttk.Frame(root)
self.btnFrame.columnconfigure(0, weight = 1)
self.btnFrame.columnconfigure(1, weight=1)
self.btnFrame.columnconfigure(2, weight=1)

#create and gird 3  buttons: for tcp, udp, or icmp - we can add more later?
self.b1 = ttk.Button(self.btnFrame, padding=10, text="tcp", command = lambda:self.startSniffing('tcp'))
self.b2 = ttk.Button(self.btnFrame, padding=10, text="udp", command = lambda:self.startSniffing('udp'))
self.b3 = ttk.Button(self.btnFrame, padding=10, text="icmp", command = lambda:self.startSniffing('icmp'))

#grid in the buttons
self.b1.grid(row=0, column=0, sticky="wens", padx=10)
self.b2.grid(row=0, column=1, sticky="wens", padx=10)
self.b3.grid(row=0, column=2, sticky="wens", padx=10)

self.btnFrame.pack()

#import button
self.importFrame = ttk.Frame(root)
self.importFrame.columnconfigure(0, weight=2)
self.importFrame.columnconfigure(1, weight=2)

self.importBtn = ttk.Button(self.importFrame, padding=10, text="Import PCAP",
                        command = lambda:self.importPcap())
self.importBtn.grid(row=0, column=0, columnspan=2, sticky="wens", padx=10, pady=10)

self.importFrame.pack()
```

^ these are the frames for the button. Each button does something slightly different when clicked. The TCP, UDP, and ICMP buttons call the startSniffing function with their respective names as parameters (these will be the filters used during the sniff). The import button calls the importPcap function. Each of the buttons are placed in the grid and packed to the screen.

```
#another text frame for updates or titles
self.textFrame2 = ttk.Frame(root)

self.textFrame2.columnconfigure(0, weight = 2)
self.textFrame2.columnconfigure(1, weight = 2)

self.message = ttk.Label(self.textFrame2, text = "", anchor = "center")
self.message.grid(row=0, column=0, columnspan = 2, sticky = "wens", padx=70, pady=20)


self.textFrame2.pack()
```

^ this is a second text frame that is placed under the buttons. Initially, the text is empty. This is where the text “sniffing…” or “IP Header” is displayed based on whether the program is actively sniffing or whether it is displaying data.

```
#create the table frame, with columns for the different table columns/bits
self.tblFrame = ttk.Frame(root)
for i in range(33):
    self.tblFrame.columnconfigure(i, weight=1)

# labels


self.c1 = ttk.Label(self.tblFrame, text = "Bit offset", anchor="center", borderwidth=2, relief="solid")
self.c2 = ttk.Label(self.tblFrame, text="0-3", anchor="center", borderwidth=2, relief="solid")
self.c3 = ttk.Label(self.tblFrame, text="4-7", anchor="center", borderwidth=2, relief="solid")
self.c4 = ttk.Label(self.tblFrame, text="8-15", anchor="center", borderwidth=2, relief="solid")
self.c5 = ttk.Label(self.tblFrame, text="16-18", anchor="center", borderwidth=2, relief="solid")
self.c6 = ttk.Label(self.tblFrame, text="19-31", anchor="center", borderwidth=2, relief="solid")
self.c1.grid(row=0, column=0, sticky="wens", ipady=10)
self.c2.grid(row=0, column=1, columnspan=4, sticky="wens", ipady=10)
self.c3.grid(row=0, column=5, columnspan=4, sticky="wens", ipady=10)
self.c4.grid(row=0, column=9, columnspan=8, sticky="wens", ipady=10)
self.c5.grid(row=0, column=17, columnspan=3,  sticky="wens", ipady=10)
self.c6.grid(row=0, column=20, columnspan=13, sticky="wens", ipady=10)

#self.tblFrame.pack(expand=True, fill="both", pady=10, padx=10)
```

^ this was the most complicated frame to set up. It has 33 columns, one for the bit offset label, and 32 for the bits in each row of the graph. I also initialized the labels here. These labels were put in the grid, but the frame is not yet packed because we don’t want to display it until we add

the data.

```
#Reset btn grid + image save button
self.rFrame = ttk.Frame(root)

self.rFrame.columnconfigure(0, weight = 2)
self.rFrame.columnconfigure(1, weight = 2)

self.rBtn = ttk.Button(self.rFrame, padding=10, text="Reset", command = lambda:self.reset())
self.saveBtn = ttk.Button(self.rFrame, padding=10, text="Save as png", command = lambda:self.capture_widget(self.tblFrame))

self.rBtn.grid(row=0, column=0, sticky = "wens", padx=70, pady=20)
self.saveBtn.grid(row=0, column=1, sticky="wens", padx=70, pady=20)

self.rFrame.pack(side="bottom")
```

^ the final frame contains the reset button and the save button. The reset button calls the reset function, and the save button calls the capture_widget function, sending in the table frame as a parameter.

```
def importPcap(self):
    filename = filedialog.askopenfilename(title="Select a PCAP")
    try:
        sniff(offline=filename, prn=self.display, store=0, count=1)
    except:
        print("cannot read file")
        pass
```

^ the next function in the file is the importPcap function. This opens up a file dialog, allowing the user to choose a file to import. If possible, the program sniffs this file, sending the first packet to the display function (show later). If it can't read the file or id no file is entered, it prints "cannot read file" to the console and silently passes in the tkinter view.

```
def reset(self):
    self.tblFrame.pack_forget()
    self.message.config(text="")
    self.saveBtn.configure(text="Save as png")
```

^ the reset function. This unpacks/undisplays the table frame from the screen. It also changes the message and button textx back to the default.

```
def startSniffing(self, filter):
    self.message.config(text="sniffing ... ")
    self.message.update_idletasks()
    def sniffPackets(self, filter):
        sniff(filter=filter, prn=self.display, count=1)
        self.message.config(text="IP Header:")
    Thread(target=sniffPackets, args=(self, filter), daemon=True).start()
```

^ the startSniffing button takes the filter and uses scapy's sniff() to capture the first packet in that particular filter (TCP, UDP, or ICMP). It sends this packet to the display function. I've also added some threading in order to display the test "sniffing…" in the time between the user pressing the button and the program fetching and displaying the data.

```
def display(self, packet):
    print("displaying graph of captured packet")
    print(packet.show(dump=True))
    #cinfigure the captured data

    #row1

    self.rc11 = ttk.Label(self.tblFrame, text = "0", anchor="center",
                          borderwidth=2,relief = "solid")
    self.rc12 = ttk.Label(self.tblFrame, text = f"Version: {packet.version}", anchor="center",
                          borderwidth=2,relief = "solid")
    self.rc13 = ttk.Label(self.tblFrame, text = f"HDR Length: {packet.ihl}", anchor="center",
                          borderwidth=2,relief = "solid")
    self.rc14 = ttk.Label(self.tblFrame, text = f"Type of Service: {packet.tos}", anchor="center",
                          borderwidth=2,relief = "solid")
    self.rc15 = ttk.Label(self.tblFrame, text=f"Total Length: {packet.len}", anchor="center",
                          borderwidth=2, relief = "solid")

    self.rc11.grid(row=1, column=0, sticky="wens", ipady=10)
    self.rc12.grid(row=1, column=1, columnspan=4, sticky="wens", ipady=10)
    self.rc13.grid(row=1, column=5, columnspan=4, sticky="wens", ipady=10)
    self.rc14.grid(row=1, column=9, columnspan=8, sticky="wens", ipady=10)
    self.rc15.grid(row=1, column=17, columnspan=16, sticky="wens", ipady=10)

    #row2
```

^ the display function is the longest section. This is because we are setting up each row of the graph/table. This is organized by row, and they are all pretty similar just with different values, so here is the first row. We create labels putting the values we got from the scapy sniff into the text

section. During the grid, each label is spaced out to take up the amount of bits that the data takes up. This is done via columnspan.

```
#row4

self.rc41 = ttk.Label(self.tblFrame, text = "96", anchor="center",
                      borderwidth=2,relief = "solid")
self.rc42 = ttk.Label(self.tblFrame, text = f"Source IP Adress: {packet[0][1].src}",
                      anchor="center",
                      borderwidth=2,relief = "solid")


self.rc41.grid(row=4, column=0, sticky="wens", ipady=10)
self.rc42.grid(row=4, column=1, columnspan=32, sticky="wens", ipady=10)




#row5
self.rc51 = ttk.Label(self.tblFrame, text = "128", anchor="center",
                      borderwidth=2,relief = "solid")
self.rc52 = ttk.Label(self.tblFrame, text = f"Destination IP Adress: {packet[0][1].dst}",
                      anchor="center",
                      borderwidth=2,relief = "solid")
```

^ Note: in order to get the proper IP addresses, we had to access a slightly different section of the scapy data – using packet[0][1]. (same for options)

```
def capture_widget(self, widget):
    widget.update()
    widget.focus()

    x0 = widget.winfo_rootx()
    y0 = widget.winfo_rooty()
    x1 = x0 + widget.winfo_width()
    y1 = y0 + widget.winfo_height()

    img = ImageGrab.grab((x0, y0, x1, y1))
    #return img
    img.save(f"packet{self.num_captures}.png")
    self.num_captures = self.num_captures + 1
    self.saveBtn.configure(text="saved!")
```

^ the final function is the one that is called when the save button is pressed. This gathers info about the widget given as a parameter (the table frame). I calculates which part of the tkinter view the widget is taking up and take a screenshot of that portion. It then saves the screenshot into “packet_.png”, based on num_captures, which was initialized at the beginning of the code

```
if __name__ == '__main__':
    #startSniffing("tcp")
    root = tkinter.Tk()
    root.geometry("800x800")
    display = PacketGUI(root=root)
    root.mainloop()
```

^ at the end of this file, we start the program, creating the tkinter root and starting up the GUI and the main loop.

Overall, I had a lot of fun with this project. I enjoyed figuring out how to implement the various features I wanted this program to include as well as learning how to better use tools like tkinter, scapy, and threading. I did have to do a lot more research than I anticipated in order to solve all of the random hurdles I can across, and the sources I referenced (and remembered to keep the link for) are linked below.

GitHub Link

https://github.com/mikaylahubbard/CYB220/tree/4834b3cbc27f599e3416e0e191fec9458dcbbfae/FinalProject

References

https://www.geeksforgeeks.org/python-grid-method-in-tkinter/

https://0xbharath.github.io/art-of-packet-crafting-with-scapy/scapy/inspecting_packets/index.html

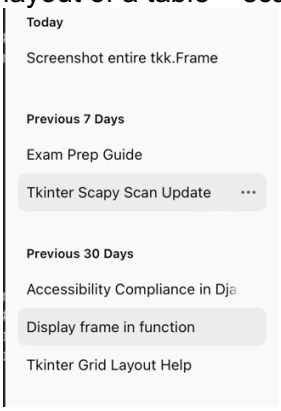
https://stackoverflow.com/questions/7208961/which-widget-do-you-use-for-a-excel-like-table-in-tkinter

https://pypi.org/project/tktable/

https://askubuntu.com/questions/25347/what-command-do-i-need-to-unzip-extract-a-tar-gz-file

https://packaging.python.org/en/latest/guides/installing-using-linux-tools/

ChatGBT - for advice and analyzing my code:
layout of a table – scapy scan – thread - screenshotting



^ for the conversations relevant to this project, I had imported screenshot of my code for the ai to give suggestions, and because I uploaded images, it wouldn't let me download the chats. I can screenshot the chats of me troubleshooting if you would like verification. I'm not sure how exactly to cite chatGBT conversations.

https://stackoverflow.com/questions/39416021/border-for-tkinter-label

https://stackoverflow.com/questions/39259264/is-there-an-option-to-edit-the-padding-inside-of-a-tkinter-entrybox