1. Министерство науки и высшего образования Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Высшая школа кибербезопасности

**ЛАБОРАТОРНАЯ РАБОТА № 7**

1. **«Разработка генератора пакетов»**
2. по дисциплине «Программно-аппаратные средства обеспечения информационной безопасности»
3. Выполнил
4. студент гр. 5151004/10101 Плохотникова М.С.

<*подпись*>

1. Проверил
2. преподаватель Семенов П.О.

<*подпись*>

1. Санкт-Петербург
2. 2024

# Цель работы

В работе необходимо разработать генератор пакетов, который выполняет следующие функции:

* предоставлять возможность пользователю сформировать любой IP, TCP, UDP и ICMP (только Echo request и Echo Reply) пакет в части заполнения всех полей протоколов (в том числе зарезервированных);
* автоматическое определение присутствующих сетевых интерфейсов и возможность выбора пользователем нужного интерфейса;
* автоматическое формирование информации пакетов канального уровня (Ethernet 802.3) – MAC-адреса отправителя и MAC-адреса получателя по указанным ip-адресам;
* предоставлять возможность пользователю заносить в поля протоколов неверных данных (длина, контрольная сумма, фрагментация и т.д.);
* формировать последовательности из сгенерированных сетевых пакетов и осуществлять отправку данных пакетов по выбранному пользователем интерфейсу.

# ход работы

Генератор пакетов реализован на языке программирования Python c использованием библиотек scapy для генерации пакетов, netifaces и winreg для определения сетевых интерфейсов, а также customtkinter для разработки графического интерфейса.

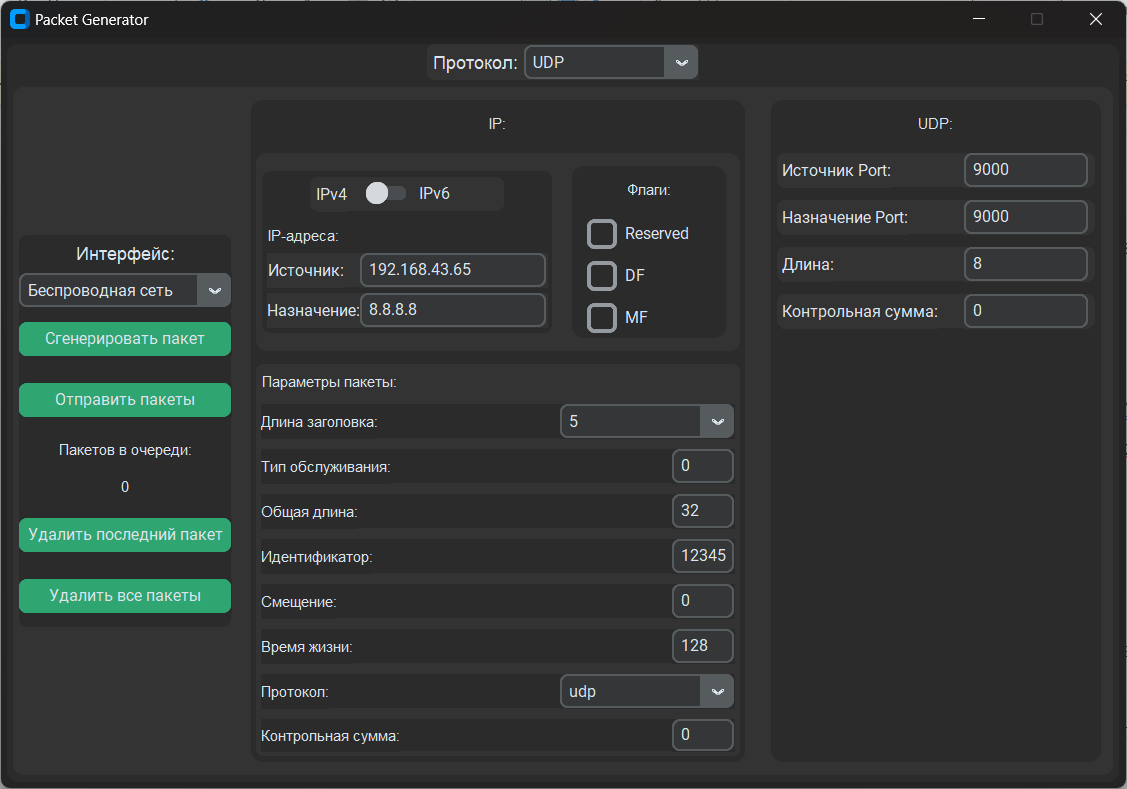


Рисунок 1 - Интерфейс генератора пакетов

Пользователь может ввести параметры протокола, выбрать сетевой интерфейс и сгенерировать пакет, который будет добавлен в очередь. После добавления пакет можно удалить из очереди или очистить всю очередь. При нажатии кнопки «Отправить пакеты» все пакеты из очереди будут отправлены на выбранный сетевой интерфейс.

Автоматическое определение всех сетевых интерфейсов происходит в функции get\_connection\_name\_from\_guid(). Удобочитаемые имена интерфейсов были получены из реестра Windows.

def get\_connection\_name\_from\_guid(self, iface\_guids):

iface\_names = ['(unknown)' for i in range(len(iface\_guids))]

reg = wr.ConnectRegistry(None, wr.HKEY\_LOCAL\_MACHINE)

reg\_key = wr.OpenKey(reg, r'SYSTEM\CurrentControlSet\Control\Network\{4d36e972-e325-11ce-bfc1-08002be10318}')

for i in range(len(iface\_guids)):

try:

reg\_subkey = wr.OpenKey(reg\_key, iface\_guids[i] + r'\Connection')

iface\_names[i] = wr.QueryValueEx(reg\_subkey, 'Name')[0]

except FileNotFoundError:

pass

return iface\_names

Изначально для получения MAC-адреса отправителя использовалась функция getmacbyip(), которая опирается на ARP-пакеты. Однако для определения MAC-адреса интерфейса источника этот подход оказался неподходящим. В итоге MAC-адрес интерфейса источника определяется путем перебора всех доступных сетевых интерфейсов.

Ниже приведена работа разработанного генератора.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2 - Тестирование отправки UDP протокола

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 3 - Тестирование отправки ICMP протокола

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 4 - Тестирование отправки TCP протокола

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 5 - Тестирование отправки IP протокола

# Выводы

В ходе лабораторной работы был разработан генератор пакетов для отправки сетевых пакетов с использованием протоколов IP, TCP, UDP и ICMP.

Разработанная программа имеет возможность настраивать различные параметры пакетов, например: интерфейс, с которого будут отправляться пакеты; IP-адреса отправителя и получателя; типа инкапсулируемого пакета и т.д. Если поле не заполнено, то в данном поле устанавливается значение по умолчанию.

Приложение

Листинг программы «генератор пакетов»:

import customtkinter as ctk

from scapy.all import \*

from scapy.layers.inet import IP, TCP, UDP, ICMP

import netifaces as ni

import winreg as wr

from scapy.layers.l2 import getmacbyip, Ether

ctk.set\_appearance\_mode("Dark")

ctk.set\_default\_color\_theme("green")

class PacketGeneratorApp(ctk.CTk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.num\_packets = 0

self.packet\_queue = []

self.title("Packet Generator")

self.geometry("1000x600")

self.resizable(False, False)

self.protocol\_frame = ctk.CTkFrame(self, corner\_radius=10)

self.protocol\_frame.pack(pady=5, padx=5, fill="both", expand=True)

self.choice\_frame = ctk.CTkFrame(self.protocol\_frame)

self.choice\_frame.pack()

self.protocol\_label = ctk.CTkLabel(self.choice\_frame, text="Протокол:", font=("Arial", 14))

self.protocol\_label.pack(side="left", pady=1, padx=5)

self.protocol\_var = ctk.StringVar(value="TCP")

self.protocol\_combobox = ctk.CTkComboBox(self.choice\_frame, values=["TCP", "UDP", "ICMP", "IP"], command=self.update\_packet\_fields, variable=self.protocol\_var)

self.protocol\_combobox.pack(side="left", pady=1)

self.packet\_frame = ctk.CTkFrame(self.protocol\_frame, corner\_radius=10)

self.packet\_frame.pack(pady=5, padx=5, fill="both", expand=True)

self.buttom\_frame = ctk.CTkFrame(self.packet\_frame)

self.buttom\_frame.pack(side="left", pady=1, padx=5)

self.interfaces\_all = ni.interfaces()

self.x = self.get\_connection\_name\_from\_guid(self.interfaces\_all)

self.int\_label = ctk.CTkLabel(self.buttom\_frame, text="Интерфейс:", font=("Arial", 14))

self.int\_label.pack(pady=1, padx=5)

self.int\_var = ctk.StringVar(value="Беспроводная сеть")

self.int\_combobox = ctk.CTkComboBox(self.buttom\_frame, values=self.x, variable=self.int\_var)

self.int\_combobox.pack(pady=1, fill="x")

self.generate\_button = ctk.CTkButton(self.buttom\_frame, text="Автогенерирование пакета", command=self.auto\_generate\_packet)

self.generate\_button.pack(fill="x", pady=10)

self.generate\_button = ctk.CTkButton(self.buttom\_frame, text="Сгенерировать пакет", command=self.generate\_packet)

self.generate\_button.pack(fill="x", pady=10)

self.generate\_button = ctk.CTkButton(self.buttom\_frame, text="Отправить пакеты", command=self.send\_packet)

self.generate\_button.pack(fill="x", pady=10)

self.info\_packet\_label = ctk.CTkLabel(self.buttom\_frame, text="Пакетов в очереди:", font=("Arial", 12))

self.info\_packet\_label.pack(pady=1)

self.info\_packet = ctk.CTkLabel(self.buttom\_frame, text=str(self.num\_packets), font=("Arial", 12))

self.info\_packet.pack(pady=1)

self.generate\_button = ctk.CTkButton(self.buttom\_frame, text="Удалить последний пакет", command=self.del\_packet)

self.generate\_button.pack(fill="x", pady=10)

self.generate\_button = ctk.CTkButton(self.buttom\_frame, text="Удалить все пакеты", command=self.del\_all\_packet)

self.generate\_button.pack(fill="x", pady=10)

# IP -------------------------------------------------------------------------------

self.ip\_frame = ctk.CTkFrame(self.packet\_frame, corner\_radius=10)

self.ip\_frame.pack(side="left", pady=10, padx=10, fill="both", expand=True)

self.ip\_label = ctk.CTkLabel(self.ip\_frame, text="IP:", font=("Arial", 12))

self.ip\_label.pack(pady=5)

self.ip\_block = ctk.CTkFrame(self.ip\_frame, corner\_radius=10)

self.ip\_block.pack(side="top", fill="x", padx=[5, 5], pady=5)

# IP address----------------

self.ip\_address = ctk.CTkFrame(self.ip\_block)

self.ip\_address.pack(side="left", padx=5, pady=5)

# IP version---------

self.ip\_version\_frame = ctk.CTkFrame(self.ip\_address)

self.ip\_version\_frame.pack(pady=5, padx=5)

self.ip\_version\_label = ctk.CTkLabel(self.ip\_version\_frame, text="IPv4")

self.ip\_version\_label.pack(side="left", padx=5)

self.ip\_version\_var = ctk.StringVar(value="IPv4")

self.ip\_version\_switch = ctk.CTkSwitch(

master=self.ip\_version\_frame,

text="IPv6",

command=self.toggle\_ip\_version,

variable=self.ip\_version\_var,

onvalue="IPv6", offvalue="IPv4"

)

self.ip\_version\_switch.pack(side="left", padx=10)

# IP version---------

self.ip\_address\_label = ctk.CTkLabel(self.ip\_address, text="IP-адресa:", font=("Arial", 12))

self.ip\_address\_label.pack(anchor="nw", padx=5)

self.ip\_src = ctk.CTkFrame(self.ip\_address)

self.ip\_src.pack(fill="x", padx=[5, 5])

self.ip\_source\_label = ctk.CTkLabel(self.ip\_src, text="Источник:")

self.ip\_source\_label.pack(side="left", ipadx=1)

self.ip\_source\_entry = ctk.CTkEntry(self.ip\_src, width=150)

self.ip\_source\_entry.insert(0, "192.168.43.65")

self.ip\_source\_entry.pack(side="right")

self.ip\_dst = ctk.CTkFrame(self.ip\_address)

self.ip\_dst.pack(fill="x", padx=[5, 5], pady=[5,5])

self.ip\_destination\_label = ctk.CTkLabel(self.ip\_dst, text="Назначение:")

self.ip\_destination\_label.pack(side="left")

self.ip\_destination\_entry = ctk.CTkEntry(self.ip\_dst, width=150)

self.ip\_destination\_entry.insert(0, "8.8.8.8")

self.ip\_destination\_entry.pack(side="right")

# IP flag -------------

self.ip\_flag = ctk.CTkFrame(self.ip\_block, corner\_radius=10)

self.ip\_flag.pack(side="top", pady=10, padx=10, fill="both", expand=True)

self.ip\_flag\_label = ctk.CTkLabel(self.ip\_flag, text="Флаги:", font=("Arial", 12))

self.ip\_flag\_label.pack(pady=5)

self.check\_var\_reserved = ctk.StringVar(value="off")

self.checkbox\_reserved = ctk.CTkCheckBox(self.ip\_flag, text="Reserved", command=self.checkbox\_event, variable=self.check\_var\_reserved, onvalue="on", offvalue="off")

self.checkbox\_reserved.pack(pady=5)

self.check\_var\_df = ctk.StringVar(value="off")

self.checkbox\_df = ctk.CTkCheckBox(self.ip\_flag, text="DF", command=self.checkbox\_event, variable=self.check\_var\_df, onvalue="on", offvalue="off")

self.checkbox\_df.pack(pady=5)

self.check\_var\_mf = ctk.StringVar(value="off")

self.checkbox\_mf = ctk.CTkCheckBox(self.ip\_flag, text="MF", command=self.checkbox\_event, variable=self.check\_var\_mf, onvalue="on", offvalue="off")

self.checkbox\_mf.pack(pady=5)

# IP params----------------

self.ip\_params = ctk.CTkFrame(self.ip\_frame)

self.ip\_params.pack(fill="x", padx=[5, 5], side="bottom", pady=5)

self.ip\_params\_label = ctk.CTkLabel(self.ip\_params, text="Параметры пакеты:", font=("Arial", 12))

self.ip\_params\_label.pack(anchor="nw", padx=5)

self.ihl = ctk.CTkFrame(self.ip\_params)

self.ihl.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.ihl\_label = ctk.CTkLabel(self.ihl, text="Длина заголовка:", font=("Arial", 12))

self.ihl\_label.pack(side="left")

self.ihl\_var = ctk.StringVar(value="5")

self.ihl\_combobox = ctk.CTkComboBox(self.ihl, values=["5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15"], variable=self.ihl\_var)

self.ihl\_combobox.pack(side="right", padx=[5, 0])

self.tos = ctk.CTkFrame(self.ip\_params)

self.tos.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tos\_label = ctk.CTkLabel(self.tos, text="Тип обслуживания:", font=("Arial", 12))

self.tos\_label.pack(side="left")

self.tos\_entry = ctk.CTkEntry(self.tos, width=50)

self.tos\_entry.insert(0, 0)

self.tos\_entry.pack(side="right")

self.total\_len = ctk.CTkFrame(self.ip\_params)

self.total\_len.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.total\_len\_label = ctk.CTkLabel(self.total\_len, text="Общая длина:", font=("Arial", 12))

self.total\_len\_label.pack(side="left")

self.total\_len\_entry = ctk.CTkEntry(self.total\_len, width=50)

self.total\_len\_entry.insert(0, 32)

self.total\_len\_entry.pack(side="right")

self.id = ctk.CTkFrame(self.ip\_params)

self.id.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.id\_label = ctk.CTkLabel(self.id, text="Идентификатор:", font=("Arial", 12))

self.id\_label.pack(side="left")

self.id\_entry = ctk.CTkEntry(self.id, width=50)

self.id\_entry.insert(0, 12345)

self.id\_entry.pack(side="right")

self.fragment\_offset = ctk.CTkFrame(self.ip\_params)

self.fragment\_offset.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.fragment\_offset\_label = ctk.CTkLabel(self.fragment\_offset, text="Смещение:", font=("Arial", 12))

self.fragment\_offset\_label.pack(side="left")

self.fragment\_offset\_entry = ctk.CTkEntry(self.fragment\_offset, width=50)

self.fragment\_offset\_entry.insert(0, 0)

self.fragment\_offset\_entry.pack(side="right")

self.TTL = ctk.CTkFrame(self.ip\_params)

self.TTL.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.TTL\_label = ctk.CTkLabel(self.TTL, text="Время жизни:", font=("Arial", 12))

self.TTL\_label.pack(side="left")

self.TTL\_entry = ctk.CTkEntry(self.TTL, width=50)

self.TTL\_entry.insert(0, 128)

self.TTL\_entry.pack(side="right")

self.in\_protocol = ctk.CTkFrame(self.ip\_params)

self.in\_protocol.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.in\_protocol\_label = ctk.CTkLabel(self.in\_protocol, text="Протокол:", font=("Arial", 12))

self.in\_protocol\_label.pack(side="left")

self.in\_protocol\_var = ctk.StringVar(value="ip")

self.in\_protocol\_combobox = ctk.CTkComboBox(self.in\_protocol, values=["icmp", "tcp", "udp", "ip"], variable=self.in\_protocol\_var)

self.in\_protocol\_combobox.pack(side="right", padx=[5, 0])

self.check\_sum = ctk.CTkFrame(self.ip\_params)

self.check\_sum.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.check\_sum\_label = ctk.CTkLabel(self.check\_sum, text="Контрольная сумма:", font=("Arial", 12))

self.check\_sum\_label.pack(side="left")

self.check\_sum\_entry = ctk.CTkEntry(self.check\_sum, width=50)

self.check\_sum\_entry.insert(0, 0)

self.check\_sum\_entry.pack(side="right")

# IP params --------

# TCP -------------------------------------------------------------------------------

self.tcp\_frame = ctk.CTkFrame(self.packet\_frame, corner\_radius=10)

self.tcp\_frame.pack(side="left", pady=10, padx=10, fill="both", expand=True)

self.tcp\_label = ctk.CTkLabel(self.tcp\_frame, text="TCP:", font=("Arial", 12))

self.tcp\_label.pack(pady=5)

self.tcp\_src = ctk.CTkFrame(self.tcp\_frame)

self.tcp\_src.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tcp\_source\_port\_label = ctk.CTkLabel(self.tcp\_src, text="Источник Port:")

self.tcp\_source\_port\_label.pack(side="left", padx=5)

self.tcp\_source\_port\_entry = ctk.CTkEntry(self.tcp\_src, width=120)

self.tcp\_source\_port\_entry.insert(0, 9000)

self.tcp\_source\_port\_entry.pack(side="right", padx=5)

self.tcp\_dst = ctk.CTkFrame(self.tcp\_frame)

self.tcp\_dst.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tcp\_destination\_port\_label = ctk.CTkLabel(self.tcp\_dst, text="Назначение Port:")

self.tcp\_destination\_port\_label.pack(side="left", padx=5)

self.tcp\_destination\_port\_entry = ctk.CTkEntry(self.tcp\_dst, width=120)

self.tcp\_destination\_port\_entry.insert(0, 80)

self.tcp\_destination\_port\_entry.pack(side="right", padx=5)

self.tcp\_seq\_num = ctk.CTkFrame(self.tcp\_frame)

self.tcp\_seq\_num.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tcp\_seq\_num\_label = ctk.CTkLabel(self.tcp\_seq\_num, text="Sequence Number:")

self.tcp\_seq\_num\_label.pack(side="left", padx=5)

self.tcp\_seq\_num\_entry = ctk.CTkEntry(self.tcp\_seq\_num, width=120)

self.tcp\_seq\_num\_entry.insert(0, 1)

self.tcp\_seq\_num\_entry.pack(side="right", padx=5)

self.tcp\_ack\_num = ctk.CTkFrame(self.tcp\_frame)

self.tcp\_ack\_num.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tcp\_ack\_num\_label = ctk.CTkLabel(self.tcp\_ack\_num, text="Acknowledgement Number:")

self.tcp\_ack\_num\_label.pack(side="left", padx=5)

self.tcp\_ack\_num\_entry = ctk.CTkEntry(self.tcp\_ack\_num, width=120)

self.tcp\_ack\_num\_entry.insert(0, 1)

self.tcp\_ack\_num\_entry.pack(side="right", padx=5)

self.tcp\_data\_offset = ctk.CTkFrame(self.tcp\_frame)

self.tcp\_data\_offset.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tcp\_data\_offset\_label = ctk.CTkLabel(self.tcp\_data\_offset, text="Data Offset:")

self.tcp\_data\_offset\_label.pack(side="left", padx=5)

self.tcp\_data\_offset\_entry = ctk.CTkEntry(self.tcp\_data\_offset, width=120)

self.tcp\_data\_offset\_entry.insert(0, 5)

self.tcp\_data\_offset\_entry.pack(side="right", padx=5)

self.tcp\_reserved = ctk.CTkFrame(self.tcp\_frame)

self.tcp\_reserved.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tcp\_reserved\_label = ctk.CTkLabel(self.tcp\_reserved, text="Reserved:")

self.tcp\_reserved\_label.pack(side="left", padx=5)

self.tcp\_reserved\_entry = ctk.CTkEntry(self.tcp\_reserved, width=120)

self.tcp\_reserved\_entry.insert(0, 0)

self.tcp\_reserved\_entry.pack(side="right", padx=5)

self.tcp\_flags = ctk.CTkFrame(self.tcp\_frame)

self.tcp\_flags.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tcp\_flags\_label = ctk.CTkLabel(self.tcp\_flags, text="Flags:")

self.tcp\_flags\_label.pack(side="left", padx=5)

self.check\_var\_urg = ctk.StringVar(value="off")

self.checkbox\_urg = ctk.CTkCheckBox(self.tcp\_flags, text="URG", command=self.checkbox\_event, variable=self.check\_var\_urg, onvalue="on", offvalue="off")

self.checkbox\_urg.pack(pady=5)

self.check\_var\_ack = ctk.StringVar(value="off")

self.checkbox\_ack = ctk.CTkCheckBox(self.tcp\_flags, text="ACK", command=self.checkbox\_event, variable=self.check\_var\_ack, onvalue="on", offvalue="off")

self.checkbox\_ack.pack(pady=5)

self.check\_var\_psh = ctk.StringVar(value="off")

self.checkbox\_psh = ctk.CTkCheckBox(self.tcp\_flags, text="PSH", command=self.checkbox\_event, variable=self.check\_var\_psh, onvalue="on", offvalue="off")

self.checkbox\_psh.pack(pady=5)

self.check\_var\_rst = ctk.StringVar(value="off")

self.checkbox\_rst = ctk.CTkCheckBox(self.tcp\_flags, text="RST", command=self.checkbox\_event, variable=self.check\_var\_rst, onvalue="on", offvalue="off")

self.checkbox\_rst.pack(pady=5)

self.check\_var\_syn = ctk.StringVar(value="off")

self.checkbox\_syn = ctk.CTkCheckBox(self.tcp\_flags, text="SYN", command=self.checkbox\_event, variable=self.check\_var\_syn, onvalue="on", offvalue="off")

self.checkbox\_syn.pack(pady=5)

self.check\_var\_fin = ctk.StringVar(value="off")

self.checkbox\_fin = ctk.CTkCheckBox(self.tcp\_flags, text="FIN", command=self.checkbox\_event, variable=self.check\_var\_fin, onvalue="on", offvalue="off")

self.checkbox\_fin.pack(pady=5)

self.tcp\_window\_size = ctk.CTkFrame(self.tcp\_frame)

self.tcp\_window\_size.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tcp\_window\_size\_label = ctk.CTkLabel(self.tcp\_window\_size, text="Window Size:")

self.tcp\_window\_size\_label.pack(side="left", padx=5)

self.tcp\_window\_size\_entry = ctk.CTkEntry(self.tcp\_window\_size, width=120)

self.tcp\_window\_size\_entry.insert(0, 8192)

self.tcp\_window\_size\_entry.pack(side="right", padx=5)

self.tcp\_checksum = ctk.CTkFrame(self.tcp\_frame)

self.tcp\_checksum.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tcp\_checksum\_label = ctk.CTkLabel(self.tcp\_checksum, text="Checksum:")

self.tcp\_checksum\_label.pack(side="left", padx=5)

self.tcp\_checksum\_entry = ctk.CTkEntry(self.tcp\_checksum, width=120)

self.tcp\_checksum\_entry.insert(0, 0)

self.tcp\_checksum\_entry.pack(side="right", padx=5)

self.tcp\_urgent\_pointer = ctk.CTkFrame(self.tcp\_frame)

self.tcp\_urgent\_pointer.pack(fill="x", padx=[5, 5], pady=[5, 5])

self.tcp\_urgent\_pointer\_label = ctk.CTkLabel(self.tcp\_urgent\_pointer, text="Urgent Pointer:")

self.tcp\_urgent\_pointer\_label.pack(side="left", padx=5)

self.tcp\_urgent\_pointer\_entry = ctk.CTkEntry(self.tcp\_urgent\_pointer, width=120)

self.tcp\_urgent\_pointer\_entry.insert(0, 0)

self.tcp\_urgent\_pointer\_entry.pack(side="right", padx=5)

# UDP------------------------------------------------

self.udp\_frame = ctk.CTkFrame(self.packet\_frame, corner\_radius=10)

self.udp\_frame.pack(side="left", pady=10, padx=10, fill="both", expand=True)

self.udp\_label = ctk.CTkLabel(self.udp\_frame, text="UDP:", font=("Arial", 12))

self.udp\_label.pack(pady=5)

self.udp\_src = ctk.CTkFrame(self.udp\_frame)

self.udp\_src.pack(pady=5, padx=5, fill="x")

self.udp\_source\_port\_label = ctk.CTkLabel(self.udp\_src, text="Источник Port:")

self.udp\_source\_port\_label.pack(side="left", padx=[5, 20])

self.udp\_source\_port\_entry = ctk.CTkEntry(self.udp\_src, width=100)

self.udp\_source\_port\_entry.insert(0, 9000)

self.udp\_source\_port\_entry.pack(side="right", padx=5)

self.udp\_dst = ctk.CTkFrame(self.udp\_frame)

self.udp\_dst.pack(pady=5, padx=5, fill="x")

self.udp\_destination\_port\_label = ctk.CTkLabel(self.udp\_dst, text="Назначение Port:")

self.udp\_destination\_port\_label.pack(side="left", padx=[5, 20])

self.udp\_destination\_port\_entry = ctk.CTkEntry(self.udp\_dst, width=100)

self.udp\_destination\_port\_entry.insert(0, 9000)

self.udp\_destination\_port\_entry.pack(side="right", padx=5)

self.udp\_len = ctk.CTkFrame(self.udp\_frame)

self.udp\_len.pack(pady=5, padx=5, fill="x")

self.udp\_length\_label = ctk.CTkLabel(self.udp\_len, text="Длина:")

self.udp\_length\_label.pack(side="left", padx=[5, 20])

self.udp\_length\_entry = ctk.CTkEntry(self.udp\_len, width=100)

self.udp\_length\_entry.insert(0, 8)

self.udp\_length\_entry.pack(side="right", padx=5)

self.udp\_checksum = ctk.CTkFrame(self.udp\_frame)

self.udp\_checksum.pack(pady=5, padx=5, fill="x")

self.udp\_checksum\_label = ctk.CTkLabel(self.udp\_checksum, text="Контрольная сумма:")

self.udp\_checksum\_label.pack(side="left", padx=[5, 20])

self.udp\_checksum\_entry = ctk.CTkEntry(self.udp\_checksum, width=100)

self.udp\_checksum\_entry.insert(0, 0)

self.udp\_checksum\_entry.pack(side="right", padx=5)

# ICMP ---------------------------------------------

self.icmp\_frame = ctk.CTkFrame(self.packet\_frame, corner\_radius=10)

self.icmp\_frame.pack(pady=10, padx=10, fill="both", expand=True)

self.icmp\_label = ctk.CTkLabel(self.icmp\_frame, text="ICMP:", font=("Arial", 12))

self.icmp\_label.pack(pady=5)

self.icmp\_type = ctk.CTkFrame(self.icmp\_frame)

self.icmp\_type.pack(pady=5, padx=5, fill="x")

self.icmp\_type\_label = ctk.CTkLabel(self.icmp\_type, text="Тип:", font=("Arial", 14))

self.icmp\_type\_label.pack(side="left", pady=5)

self.icmp\_type\_var = ctk.StringVar(value="request")

self.icmp\_type\_combobox = ctk.CTkComboBox(self.icmp\_type, values=["reply", "request"], variable=self.icmp\_type\_var)

self.icmp\_type\_combobox.pack(side="right", pady=1)

self.icmp\_code = ctk.CTkFrame(self.icmp\_frame)

self.icmp\_code.pack(pady=5, padx=5, fill="x")

self.icmp\_code\_label = ctk.CTkLabel(self.icmp\_code, text="Code:")

self.icmp\_code\_label.pack(side="left", padx=5)

self.icmp\_code\_entry = ctk.CTkEntry(self.icmp\_code, width=120)

self.icmp\_code\_entry.insert(0, 0)

self.icmp\_code\_entry.pack(side="right", padx=5)

self.icmp\_checksum = ctk.CTkFrame(self.icmp\_frame)

self.icmp\_checksum.pack(pady=5, padx=5, fill="x")

self.icmp\_checksum\_label = ctk.CTkLabel(self.icmp\_checksum, text="Checksum:")

self.icmp\_checksum\_label.pack(side="left", padx=5)

self.icmp\_checksum\_entry = ctk.CTkEntry(self.icmp\_checksum, width=120)

self.icmp\_checksum\_entry.insert(0, 0)

self.icmp\_checksum\_entry.pack(side="right", padx=5)

self.icmp\_identifier = ctk.CTkFrame(self.icmp\_frame)

self.icmp\_identifier.pack(pady=5, padx=5, fill="x")

self.icmp\_identifier\_label = ctk.CTkLabel(self.icmp\_identifier, text="Identifier:")

self.icmp\_identifier\_label.pack(side="left", padx=5)

self.icmp\_identifier\_entry = ctk.CTkEntry(self.icmp\_identifier, width=120)

self.icmp\_identifier\_entry.insert(0, 1)

self.icmp\_identifier\_entry.pack(side="right", padx=5)

self.icmp\_sequence\_num = ctk.CTkFrame(self.icmp\_frame)

self.icmp\_sequence\_num.pack(pady=5, padx=5, fill="x")

self.icmp\_sequence\_number\_label = ctk.CTkLabel(self.icmp\_sequence\_num, text="Sequence Number:")

self.icmp\_sequence\_number\_label.pack(side="left", padx=5)

self.icmp\_sequence\_number\_entry = ctk.CTkEntry(self.icmp\_sequence\_num, width=120)

self.icmp\_sequence\_number\_entry.insert(0, 137)

self.icmp\_sequence\_number\_entry.pack(side="right", padx=5)

# Data -----------------------------------------------------

self.input\_data = ctk.CTkFrame(self.packet\_frame)

self.input\_data.pack(side="right", pady=1, padx=5, fill="both", expand=True)

self.input\_data\_label = ctk.CTkLabel(self.input\_data, text="Данные:")

self.input\_data\_label.pack(padx=5)

self.window\_input\_data = ctk.CTkTextbox(self.input\_data, width = 150)

self.window\_input\_data.pack(pady=10, fill="y", expand=True)

self.update\_packet\_fields()

def update\_packet\_fields(self, \*args):

selected\_protocol = self.protocol\_var.get()

self.tcp\_frame.pack\_forget()

self.udp\_frame.pack\_forget()

self.icmp\_frame.pack\_forget()

if selected\_protocol == "TCP":

self.tcp\_frame.pack(pady=10, padx=10, fill="both", expand=True)

elif selected\_protocol == "UDP":

self.udp\_frame.pack(pady=10, padx=10, fill="both", expand=True)

elif selected\_protocol == "ICMP":

self.icmp\_frame.pack(pady=10, padx=10, fill="both", expand=True)

def toggle\_ip\_version(self, \*args):

if self.ip\_version\_var.get() == "IPv4":

self.ip\_version\_var.set("IPv4")

print("IPv4")

else:

self.ip\_version\_var.set("IPv6")

print("IPv6")

def checkbox\_event(self, \*args):

print(f"Флаг: {self.check\_var\_reserved.get()}")

def generate\_packet(self):

packet = 0

ip\_source = self.ip\_source\_entry.get()

ip\_destination = self.ip\_destination\_entry.get()

protocol = self.protocol\_var.get()

# Ethernet -------------------

try:

source\_mac =self.get\_mac()

except:

print(f"Не удалось получить MAC-адрес для {ip\_source}")

return

try:

target\_mac = getmacbyip(ip\_destination)

except:

print(f"Не удалось получить MAC-адрес для {ip\_destination}")

return

ethernet\_header = Ether(src=source\_mac, dst=target\_mac)

# IP ------------------------------------------------

if self.ip\_version\_var.get() == "IPv4":

version\_ip = 4

else:

version\_ip = 6

ip\_len\_hd = int(self.ihl\_var.get())

ip\_tos = int(self.tos\_entry.get())

ip\_len = int(self.total\_len\_entry.get())

ip\_id = int(self.id\_entry.get())

ip\_offset = int(self.fragment\_offset\_entry.get())

ip\_ttl = int(self.TTL\_entry.get())

protocol\_str = self.in\_protocol\_var.get()

ip\_checksum = int(self.check\_sum\_entry.get())

var\_df = self.check\_var\_df.get()

var\_mf = self.check\_var\_mf.get()

if var\_df == "on" and var\_mf == "on":

ip\_flags=3

elif var\_df == "on":

ip\_flags=2

elif var\_mf == "on":

ip\_flags=1

else:

ip\_flags=0

ip\_header = IP(dst=ip\_destination, src=ip\_source, version=version\_ip, proto=protocol\_str, flags=ip\_flags, id=ip\_id, ihl=ip\_len\_hd, tos=ip\_tos, len=ip\_len, frag=ip\_offset, ttl=ip\_ttl)

packet = ethernet\_header / ip\_header

# print(packet.show())

if protocol == "TCP":

source\_port = int(self.tcp\_source\_port\_entry.get())

destination\_port = int(self.tcp\_destination\_port\_entry.get())

seq\_num = int(self.tcp\_seq\_num\_entry.get())

ack\_num = int(self.tcp\_ack\_num\_entry.get())

data\_offset = int(self.tcp\_data\_offset\_entry.get())

reserved = int(self.tcp\_reserved\_entry.get())

tcp\_flags = 0

for i, flag in enumerate([self.check\_var\_fin.get(), self.check\_var\_syn.get(), self.check\_var\_rst.get(), self.check\_var\_psh.get(), self.check\_var\_ack.get(), self.check\_var\_urg.get()]):

if flag == "on":

tcp\_flags |= 1 << (i)

if tcp\_flags == 0:

tcp\_flags = None

window\_size = int(self.tcp\_window\_size\_entry.get())

checksum = int(self.tcp\_checksum\_entry.get())

urgent\_pointer = int(self.tcp\_urgent\_pointer\_entry.get())

if data\_offset == 0:

data\_offset = None

if checksum == 0:

checksum = None

tcp\_header = TCP(sport=source\_port, dport=destination\_port, seq=seq\_num, ack=ack\_num, dataofs=data\_offset, reserved=reserved, flags = tcp\_flags, window=window\_size, urgptr=urgent\_pointer)

packet = packet / tcp\_header

# print(packet.show())

elif protocol == "UDP":

source\_port = int(self.udp\_source\_port\_entry.get())

destination\_port = int(self.udp\_destination\_port\_entry.get())

length = int(self.udp\_length\_entry.get())

checksum = int(self.udp\_checksum\_entry.get())

udp\_header = UDP(sport=source\_port, dport=destination\_port, len=length)

packet = packet / udp\_header

# print(packet.show())

elif protocol == "ICMP":

type\_str = self.icmp\_type\_var.get()

code = int(self.icmp\_code\_entry.get())

checksum = int(self.icmp\_checksum\_entry.get())

identifier = int(self.icmp\_identifier\_entry.get())

sequence\_number = int(self.icmp\_sequence\_number\_entry.get())

if type\_str == "reply":

type = 0

else:

type = 8

icmp\_header = ICMP(type=type, code=code, id=identifier, seq=sequence\_number)

packet = packet / icmp\_header

# print(packet.show())

input\_data = self.window\_input\_data.get("1.0", "end-1c")

packet = packet / input\_data

self.packet\_queue.append(packet)

self.num\_packets += 1

self.info\_packet.configure(text=str(self.num\_packets))

def auto\_generate\_packet(self):

packet = 0

ip\_source = self.ip\_source\_entry.get()

ip\_destination = self.ip\_destination\_entry.get()

protocol = self.protocol\_var.get()

# Ethernet -------------------

try:

source\_mac =self.get\_mac()

except:

print(f"Не удалось получить MAC-адрес для {ip\_source}")

return

try:

target\_mac = getmacbyip(ip\_destination)

except:

print(f"Не удалось получить MAC-адрес для {ip\_destination}")

return

ethernet\_header = Ether(src=source\_mac, dst=target\_mac)

# IP ------------------------------------------------

ip\_header = IP(dst=ip\_destination, src=ip\_source)

packet = ethernet\_header / ip\_header

if protocol == "TCP":

source\_port = int(self.tcp\_source\_port\_entry.get())

destination\_port = int(self.tcp\_destination\_port\_entry.get())

tcp\_header = TCP(sport=source\_port, dport=destination\_port)

packet = packet / tcp\_header

elif protocol == "UDP":

source\_port = int(self.udp\_source\_port\_entry.get())

destination\_port = int(self.udp\_destination\_port\_entry.get())

udp\_header = UDP(sport=source\_port, dport=destination\_port)

packet = packet / udp\_header

elif protocol == "ICMP":

type\_str = self.icmp\_type\_var.get()

if type\_str == "reply":

type = 0

else:

type = 8

icmp\_header = ICMP(type=type, id=1, seq=137)

packet = ethernet\_header / ip\_header / icmp\_header

input\_data = self.window\_input\_data.get("1.0", "end-1c")

packet = packet / input\_data

self.packet\_queue.append(packet)

self.num\_packets += 1

self.info\_packet.configure(text=str(self.num\_packets))

def send\_packet(self):

print("click")

for packet in self.packet\_queue:

print(self.int\_var.get())

sendp(packet, iface=self.int\_var.get(), verbose=0)

print(packet.show())

self.del\_all\_packet()

def del\_packet(self):

self.packet\_queue.pop()

self.num\_packets -= 1

self.info\_packet.configure(text=str(self.num\_packets))

def del\_all\_packet(self):

self.packet\_queue[:] = []

self.num\_packets = 0

self.info\_packet.configure(text=str(self.num\_packets))

def get\_connection\_name\_from\_guid(self, iface\_guids):

iface\_names = ['(unknown)' for i in range(len(iface\_guids))]

reg = wr.ConnectRegistry(None, wr.HKEY\_LOCAL\_MACHINE)

reg\_key = wr.OpenKey(reg, r'SYSTEM\CurrentControlSet\Control\Network\{4d36e972-e325-11ce-bfc1-08002be10318}')

for i in range(len(iface\_guids)):

try:

reg\_subkey = wr.OpenKey(reg\_key, iface\_guids[i] + r'\Connection')

iface\_names[i] = wr.QueryValueEx(reg\_subkey, 'Name')[0]

except FileNotFoundError:

pass

return iface\_names

def get\_mac(self):

selected\_value = self.int\_var.get()

for i, value in enumerate(self.x):

if value == selected\_value:

index = i

break

ifaces = self.interfaces\_all

addrs = ni.ifaddresses(ifaces[index])

if ni.AF\_LINK in addrs:

return addrs[ni.AF\_LINK][0]['addr']

if \_\_name\_\_ == "\_\_main\_\_":

app = PacketGeneratorApp()

app.mainloop()