Федеральное агентство по образованию

Государственное образовательное учреждение высшего профессионального образования

Нижегородский государственный архитектурно-

строительный университет (ННГАСУ)

*Факультет инженерно-экологических систем и сооружений  
Кафедра информационных систем и технологий*

**КУРСОВАЯ РАБОТА**  
по дисциплине: «Инфокоммуникационные системы и сети»

**На тему: «Создание онлайн-игры»**

Выполнил студент 3 курса гр.ИС-29: Поляков М.А.

Проверил: старший преподаватель: Морозов Н.С.

**Содержание**

[**ЦеЛь РаБоТы** 3](#_Toc124962870)

[**Задачи** 4](#_Toc124962871)

[**Теория** 5](#_Toc124962872)

[**IP-адресация** 5](#_Toc124962873)

[**Протокол связи TCP** 6](#_Toc124962874)

[**Заголовок сегмента TCP** 6](#_Toc124962875)

[**Механизм действия протокола** 7](#_Toc124962876)

[**Топология сети тестирования** 7](#_Toc124962877)

[**Листинг программы** 9](#_Toc124962878)

[**Серверная часть (server.py)** 9](#_Toc124962879)

[**Клиентская часть (client.py)** 11](#_Toc124962880)

[**Заключение** 15](#_Toc124962881)

# **ЦеЛь РаБоТы**

Написать на языке Python игру Камень-Ножницы-Бумага для игры в локальной сети

Клише программы (без сетевой составляющей) представлены по [ссылке](https://github.com/kit8nino/2022-BigData/blob/main/__/main.py) (<https://github.com/kit8nino/2022-BigData/blob/main/__/main.py>)

В игре должен быть чат, который не мешает остальной работе программы

# **Задачи**

1. Изучение маршрутизации, стека TCP/IP
2. Настройка протокола транспортного уровня
3. Настройка сокетов серверной и клиентской части игры

# **Теория**

## **IP-адресация**

**IP-адрес** – это уникальный числовой адрес, однозначно идентифицирующий узел, группу узлов или сеть.

IP-адрес (v4) имеет длину 4 байта и обычно записывается в виде четырех чисел (так называемых «октетов»), разделенных точками – W.X.Y.Z , каждое из которых может принимать значения в диапазоне от 0 до 255, например, 0.0.0.0.

Существует 5 классов IP-адресов – A, B, C, D, E. Принадлежность IP-адреса к тому или иному классу определяется значением первого октета (W).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Класс IP-адреса | A | B | C | D | E |
| Диапазон первого октета | 1-126 | 128-191 | 192-223 | 224-239 | 240-247 |

Протокол IP предполагает наличие адресов, которые трактуются особым образом. К ним относятся следующие:

1. Адреса, значение первого октета которых равно 127. Пакеты, направленные по такому адресу, реально не передаются в сеть, а обрабатываются программным обеспечением узла-отправителя. Таким образом, узел может направить данные самому себе. Этот подход очень удобен для тестирования сетевого программного обеспечения в условиях, когда нет возможности подключиться к сети.
2. Адрес 255.255.255.255. Пакет, в назначении которого стоит адрес 255.255.255.255, должен рассылаться всем узлам сети, в которой находится источник. Такой вид рассылки называется ограниченным широковещанием. В двоичной форме этот адрес имеет вид 11111111 11111111 11111111 11111111.
3. Адрес 0.0.0.0. Он используется в служебных целях и трактуется как адрес того узла, который сгенерировал пакет. Двоичное представление этого адреса 00000000 00000000 00000000 00000000

## **Протокол связи TCP**

**Transmission Control Protocol** (**TCP**, протокол управления передачей) — один из основных протоколов передачи данных интернета. Предназначен для управления передачей данных интернета. Пакеты в TCP называются сегментами.

Механизм TCP предоставляет поток данных с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым (в отличие от UDP) целостность передаваемых данных и уведомление отправителя о результатах передачи.

### **Заголовок сегмента TCP**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Бит** | **0 – 3** | **4 – 6** | **7 – 15** | **16 – 31** |
| **0** | Порт источника, **Source Port** | | | Порт назначения, **Destination Port** |
| **32** | Порядковый номер, **Sequence Number (SN)** | | | |
| **64** | Номер подтверждения, **Acknowledgement Number (ACK SN)** | | | |
| **96** | Длина заголовка, (**Data offset**) | Зарезервировано | Флаги | Размер окна, **Window Size** |
| **128** | Контрольная сумма, **Checksum** | | | Указатель важности, **Urgent Point** |
| **160** | Опции (необязательное, но используется почти всегда) | | | |
| **160/192+** | Данные | | | |

### **Механизм действия протокола**

В отличие от традиционной альтернативы — UDP, который может сразу же начать передачу пакетов, TCP устанавливает соединения, которые должны быть созданы перед передачей данных. TCP-соединение можно разделить на 3 стадии:

* Установка соединения
* Передача данных
* Завершение соединения

## **Топология сети тестирования**

**Сетевая топология** — это конфигурация графа, вершинам которого соответствуют конечные узлы сети (компьютеры и коммуникационное оборудование (маршрутизаторы), а рёбрам — физические или информационные связи между вершинами.

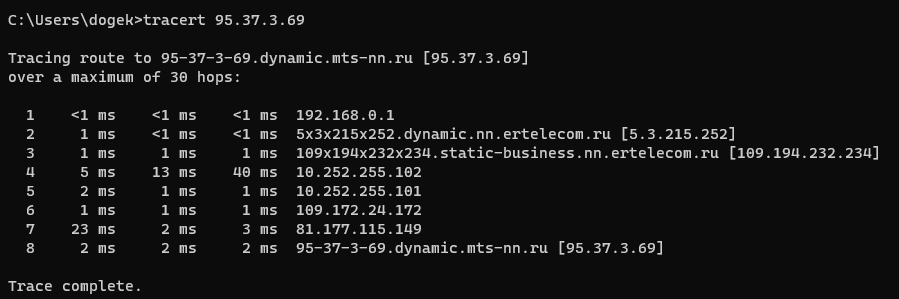


Рис 1. Трассирование устройства вне локальной сети

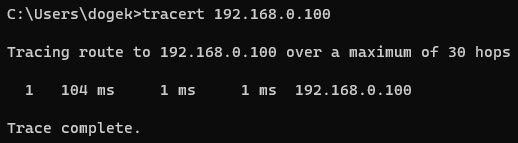


Рис 2. Трассирование устройства в локальной сети

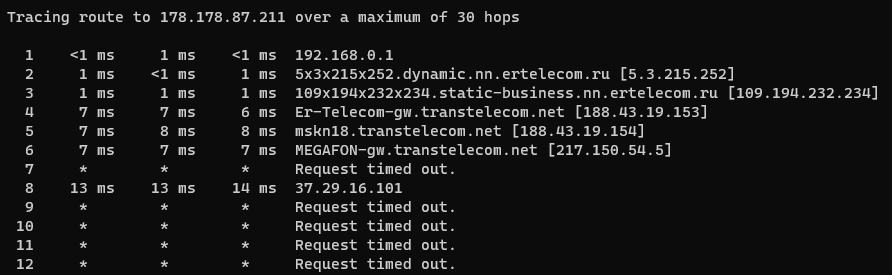


Рис 3. Трассирование устройства в моб. сети (не особо удачно)

# **Листинг программы**

## **Серверная часть (server.py)**

*import* json  
*import* socket  
*import* threading  
*from* typing *import* Optional, Union  
  
  
*class* Result:  
 WIN = "win"  
 LOSE = "lose"  
 DRAW = "draw"  
  
  
*class* RPSServer:  
 *def* \_\_init\_\_(*self*, host, port):  
 *self*.host = host  
 *self*.port = port  
 *self*.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 *self*.sock.bind((*self*.host, *self*.port))  
 *self*.clients: list[socket.socket] = []  
 *self*.actions: dict[socket.socket, dict[str, Union[int, str]]] = {}  
  
 *def* listen(*self*):  
 *self*.sock.listen(2)  
 print("Server started")  
 *while True*:  
 client, address = *self*.sock.accept()  
 threading.Thread(target=*self*.client\_handler, args=(client,)).start()  
 *self*.clients.append(client)  
 print(f"New client connected")  
  
 *def* distribute(*self*, data: str, author: Optional[socket.socket] = *None*):  
 *for* client *in self*.clients:  
 *if* client == author:  
 *continue* client.send(data.encode())  
  
 *def* client\_handler(*self*, client: socket.socket):  
 *while True*:  
 *try*:  
 data = json.loads(client.recv(1024).decode())  
 *if not* data:  
 *continue* command = data["command"]  
 nickname = data["nickname"]  
 message = data["message"]  
 print(f"Nickname: {nickname} Command: {command} Message: {message}")  
 *if* command == "chat":  
 *self*.distribute(json.dumps(data), client)  
 *continue  
 if* command == "action":  
 *self*.actions[client] = {  
 "action": int(message),  
 "nickname": nickname,  
 }  
 *if* len(*self*.actions) == 2:  
 *self*.send\_result()  
 *self*.actions = {}  
 *except* Exception *as* e:  
 print(f"Client error: {e}")  
 *self*.clients.remove(client)  
 client.close()  
 print("Client closed")  
 *return False  
  
 def* send\_result(*self*):  
 user = list(*self*.actions.keys())[0]  
 opponent = list(*self*.actions.keys())[1]  
 user\_choice = *self*.actions[user]["action"]  
 opponent\_choice = *self*.actions[opponent]["action"]  
  
 user\_result = Result.LOSE  
 opponent\_result = Result.WIN  
  
 *if* user\_choice == opponent\_choice:  
 user\_result = Result.DRAW  
 opponent\_result = Result.DRAW  
  
 *if* (user\_choice + 1) % 3 == opponent\_choice:  
 user\_result = Result.WIN  
 opponent\_result = Result.LOSE  
  
 user.send(  
 json.dumps(  
 {  
 "command": "result",  
 "message": user\_result,  
 "nickname": *self*.actions[opponent]["nickname"],  
 }  
 ).encode()  
 )  
 opponent.send(  
 json.dumps(  
 {  
 "command": "result",  
 "message": opponent\_result,  
 "nickname": *self*.actions[user]["nickname"],  
 }  
 ).encode()  
 )  
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 RPSServer("0.0.0.0", 8081).listen()

## **Клиентская часть (client.py)**

*import* json  
*import* random  
*from* enum *import* IntEnum  
*from* tkinter *import* Tk, Frame, Button, Label, END  
*import* tkinter *as* tk  
*import* threading  
*import* socket  
*from* typing *import* Optional  
  
  
*class* Action(IntEnum):  
 Rock = 0  
 Scissors = 1  
 Paper = 2  
  
  
*class* GameCommand:  
 *def* \_\_init\_\_(*self*, game: "Main", choice: Action):  
 *self*.choice = choice  
 *self*.game = game  
  
 *def* process\_button(*self*):  
 *self*.game.client.send("action", str(*self*.choice.value))  
  
 *def* \_\_call\_\_(*self*):  
 *for* btn *in self*.game.game\_buttons:  
 *if* btn["state"] == tk.DISABLED:  
 *return* btn["state"] = tk.DISABLED  
 threading.Thread(target=*self*.process\_button).start()  
  
  
*class* Main(Frame):  
 *def* \_\_init\_\_(*self*, client\_: "SocketClient"):  
 main\_root = Tk()  
 main\_root.geometry("430x360+200+200")  
 main\_root.title("Камень, ножницы, бумага")  
 main\_root.resizable(*False*, *False*)  
 main\_root["bg"] = "#FFF"  
 super(Main, *self*).\_\_init\_\_(main\_root)  
 *self*.client = client\_  
 client\_.game = *self  
 self*.root = main\_root  
 *self*.opponent\_name = ""  
 *self*.game\_buttons = []  
 *self*.game\_start\_label: Optional[Label] = *None  
 self*.game\_data\_label: Optional[Label] = *None  
 self*.opponent\_label: Optional[Label] = *None  
 self*.entry: Optional[tk.Entry] = *None  
 self*.txt: Optional[tk.Text] = *None  
 self*.\_button\_font = ("Times New Roman", 15)  
 *self*.\_mini\_button\_font = ("Times New Roman", 13)  
 *self*.win = *self*.draw = *self*.lose = 0  
  
 *self*.start\_iu()  
  
 *def* game\_data\_text(*self*):  
 *return* f"Побед: {*self*.win}\nПроигрышей:" f" {*self*.lose}\nНичей: {*self*.draw}"  
  
 *def* send\_button(*self*, \*args):  
 input\_text = *self*.entry.get()  
 *if not* input\_text:  
 *return  
 self*.txt.configure(state="normal")  
 *self*.txt.insert(END, f"Я -> {input\_text}\n")  
 *self*.txt.see("end")  
 *self*.txt.configure(state="disabled")  
 *self*.entry.delete(0, END)  
 *self*.client.send("chat", input\_text)  
  
 *def* start\_iu(*self*):  
 *self*.game\_buttons = [  
 Button(  
 *self*.root,  
 text="Камень",  
 font=*self*.\_button\_font,  
 command=GameCommand(*self*, Action.Rock),  
 ),  
 Button(  
 *self*.root,  
 text="Ножницы",  
 font=*self*.\_button\_font,  
 command=GameCommand(*self*, Action.Scissors),  
 ),  
 Button(  
 *self*.root,  
 text="Бумага",  
 font=*self*.\_button\_font,  
 command=GameCommand(*self*, Action.Paper),  
 ),  
 ]  
  
 *self*.game\_buttons[0].place(x=10, y=100, width=120, height=50)  
 *self*.game\_buttons[1].place(x=155, y=100, width=120, height=50)  
 *self*.game\_buttons[2].place(x=300, y=100, width=120, height=50)  
 *self*.master.bind("<Return>", *self*.send\_button)  
  
 *self*.game\_start\_label = Label(  
 *self*.root,  
 text="Начало игры!",  
 bg="#FFF",  
 font=("Times New Roman", 18, "bold"),  
 )  
 *self*.game\_data\_label = Label(  
 *self*.root,  
 justify="left",  
 font=*self*.\_mini\_button\_font,  
 text=*self*.game\_data\_text(),  
 bg="#FFF",  
 )  
 *self*.opponent\_label = Label(  
 *self*.root,  
 justify="right",  
 font=*self*.\_mini\_button\_font,  
 text=f"Оппонент: Нет",  
 bg="#FFF",  
 )  
 *self*.game\_start\_label.place(x=150, y=5)  
 *self*.game\_data\_label.place(x=5, y=5)  
 *self*.opponent\_label.place(x=145, y=55)  
  
 *self*.txt = tk.Text(  
 *self*.root, font=*self*.\_mini\_button\_font, width=47, height=8, bg="#c1b7c9"  
 )  
 *self*.txt.configure(state="disabled")  
 *self*.txt.place(x=10, y=160)  
 scrollbar = tk.Scrollbar(*self*.txt)  
 scrollbar.place(relheight=1, relx=0.958)  
 *self*.entry = tk.Entry(  
 *self*.root, font=*self*.\_mini\_button\_font, width=35, bg="#556066"  
 )  
 *self*.entry.place(x=10, y=335)  
 send = Button(  
 *self*.root,  
 text="Отправить",  
 font=*self*.\_mini\_button\_font,  
 command=*self*.send\_button,  
 width=9,  
 height=1,  
 )  
 send.place(x=335, y=325)  
  
 *def* normalize\_buttons(*self*):  
 *for* button *in self*.game\_buttons:  
 button["state"] = tk.NORMAL  
  
  
*class* SocketClient:  
 *def* \_\_init\_\_(*self*, name: str):  
 *self*.client = *None  
 self*.name = name  
 *self*.game: Main | *None* = *None  
  
 def* result\_handler(*self*, message: str):  
 *if* message == "draw":  
 *self*.game.draw += 1  
 *self*.game.game\_start\_label.configure(text="Ничья")  
 *if* message == "win":  
 *self*.game.win += 1  
 *self*.game.game\_start\_label.configure(text="Победа")  
 *if* message == "lose":  
 *self*.game.lose += 1  
 *self*.game.game\_start\_label.configure(text="Проигрыш")  
 *self*.game.game\_data\_label.configure(text=*self*.game.game\_data\_text())  
 *self*.game.normalize\_buttons()  
  
 *def* socket\_start(*self*, host: str, port: int):  
 *self*.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 *self*.client.connect((host, port))  
 *while True*:  
 data = *self*.client.recv(1024)  
 *if not* data:  
 *continue* data = json.loads(data.decode())  
 command = data["command"]  
 nickname = data["nickname"]  
 message = data["message"]  
  
 *self*.game.opponent\_label.configure(text=f"Оппонент: {nickname}")  
 *if* command == "result":  
 *self*.result\_handler(message)  
 *if* command == "chat":  
 *self*.game.txt.configure(state="normal")  
 *self*.game.txt.insert(END, f"{nickname} -> {message}\n")  
 *self*.game.txt.see("end")  
 *self*.game.txt.configure(state="disabled")  
  
 *def* send(*self*, command: str, message: str):  
 data = json.dumps(  
 {  
 "command": command,  
 "nickname": *self*.name,  
 "message": message  
 }  
 )  
 *self*.client.sendall(data.encode())  
  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 nick = f"user\_{random.randint(0, 100000)}"  
 print(f"{nick=}")  
 client = SocketClient(name=nick)  
 app = Main(client)  
 app.pack()  
  
 game\_thread = threading.Thread(target=app.mainloop)  
 socket\_thread = threading.Thread(  
 target=client.socket\_start, args=("192.168.0.106", 8081)  
 )  
 socket\_thread.start()  
 game\_thread.run()

# **Заключение**

Были изучены основные понятия сетевых компонентов, такие как: IP-Адресация, протокол связи TCP (TCP/IP), основные понятия трассировки сетей.

В результате работы были создана программа с клиент-серверным взаимодействием, настроено (до сброса настроек сети) взаимодействие двух устройств в локальной сети для игры.