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: Самедов Н.Ю.

Проверил: старший преподаватель: Морозов Н.С.

**Содержание**

[**Цель работы** 3](#_Toc124986256)

[**Задачи** 4](#_Toc124986257)

[**Теория** 5](#_Toc124986258)

[**IP-адресация** 5](#_Toc124986259)

[**Протокол связи TCP** 6](#_Toc124986260)

[**Заголовок сегмента TCP** 6](#_Toc124986261)

[**Механизм действия протокола** 7](#_Toc124986262)

[**Топология сети тестирования** 7](#_Toc124986263)

[**Листинг программы** 9](#_Toc124986264)

[**Серверная часть (server.py)** 9](#_Toc124986265)

[**Клиентская часть (client.py)** 11](#_Toc124986266)

[**Заключение** 15](#_Toc124986267)

# **Цель работы**

* Изучить систему маршрутизации;
* Изучить систему стека TCP/IP;
* Произвести настройку протокола транспортного уровня;
* Произвести настройку сокетов серверной и клиентской части игры;

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).

IP-адрес состоит из двух частей: номера сети и номера узла. В случае изолированной сети её адрес может быть выбран администратором из специально зарезервированных для таких сетей блоков адресов (10.0.0.0/8, 172.16.0.0/12 или 192.168.0.0/16). Для выхода в глобальную сеть необходимо, чтобы был IP из другого блока адресов, либо в локальной сети должен быть сервер, подменяющий внутренний IP-адрес (серый) на внешний IP-адрес (белый), например: [proxy server](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%BA%D1%81%D0%B8-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80" \o "Прокси-сервер), [NAT](https://ru.wikipedia.org/wiki/NAT). Если же сеть должна работать как составная часть [Интернета](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82), то адрес сети выдаётся провайдером либо региональным интернет-регистратором (Regional Internet Registry, [RIR](https://ru.wikipedia.org/wiki/RIR)). Согласно данным на сайте IANA[[1]](https://ru.wikipedia.org/wiki/IP-%D0%B0%D0%B4%D1%80%D0%B5%D1%81#cite_note-1), существует пять RIR: [ARIN](https://ru.wikipedia.org/wiki/ARIN), обслуживающий [Северную Америку](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%B2%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%90%D0%BC%D0%B5%D1%80%D0%B8%D0%BA%D0%B0), а также [Багамы](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3%D0%B0%D0%BC%D1%8B), [Пуэрто-Рико](https://ru.wikipedia.org/wiki/%D0%9F%D1%83%D1%8D%D1%80%D1%82%D0%BE-%D0%A0%D0%B8%D0%BA%D0%BE) и [Ямайку](https://ru.wikipedia.org/wiki/%D0%AF%D0%BC%D0%B0%D0%B9%D0%BA%D0%B0); [APNIC](https://ru.wikipedia.org/wiki/APNIC), обслуживающий страны [Южной](https://ru.wikipedia.org/wiki/%D0%AE%D0%B6%D0%BD%D0%B0%D1%8F_%D0%90%D0%B7%D0%B8%D1%8F), [Восточной](https://ru.wikipedia.org/wiki/%D0%92%D0%BE%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%90%D0%B7%D0%B8%D1%8F) и [Юго-Восточной Азии](https://ru.wikipedia.org/wiki/%D0%AE%D0%B3%D0%BE-%D0%92%D0%BE%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B0%D1%8F_%D0%90%D0%B7%D0%B8%D1%8F), а также [Австралии и Океании](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%81%D1%82%D1%80%D0%B0%D0%BB%D0%B8%D1%8F_%D0%B8_%D0%9E%D0%BA%D0%B5%D0%B0%D0%BD%D0%B8%D1%8F); [AfriNIC](https://ru.wikipedia.org/wiki/AfriNIC" \o "AfriNIC), обслуживающий страны [Африки](https://ru.wikipedia.org/wiki/%D0%90%D1%84%D1%80%D0%B8%D0%BA%D0%B0) и Индийского океана; [LACNIC](https://ru.wikipedia.org/wiki/LACNIC), обслуживающий страны [Южной Америки](https://ru.wikipedia.org/wiki/%D0%AE%D0%B6%D0%BD%D0%B0%D1%8F_%D0%90%D0%BC%D0%B5%D1%80%D0%B8%D0%BA%D0%B0) и бассейна Карибского моря; и [RIPE NCC](https://ru.wikipedia.org/wiki/RIPE_NCC), обслуживающий [Европу](https://ru.wikipedia.org/wiki/%D0%95%D0%B2%D1%80%D0%BE%D0%BF%D0%B0), [Центральную Азию](https://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BD%D1%82%D1%80%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%90%D0%B7%D0%B8%D1%8F), [Ближний Восток](https://ru.wikipedia.org/wiki/%D0%91%D0%BB%D0%B8%D0%B6%D0%BD%D0%B8%D0%B9_%D0%92%D0%BE%D1%81%D1%82%D0%BE%D0%BA). Региональные регистраторы получают номера [автономных систем](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D0%BD%D0%BE%D0%BC%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_(%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82)) и большие блоки адресов у [IANA](https://ru.wikipedia.org/wiki/IANA), а затем выдают номера автономных систем и блоки адресов меньшего размера локальным интернет-регистраторам (Local Internet Registries, [LIR](https://ru.wikipedia.org/wiki/LIR)), обычно являющимся крупными провайдерами.

Номер узла в [протоколе](https://ru.wikipedia.org/wiki/IP) [IP](https://ru.wikipedia.org/wiki/IP) назначается независимо от локального адреса узла. Маршрутизатор по определению входит сразу в несколько сетей. Поэтому каждый порт маршрутизатора имеет собственный IP-адрес. Конечный узел также может входить в несколько IP-сетей. В этом случае компьютер должен иметь несколько IP-адресов, по числу сетевых связей. Таким образом, IP-адрес характеризует не отдельный компьютер или маршрутизатор, а одно сетевое соединение.

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.

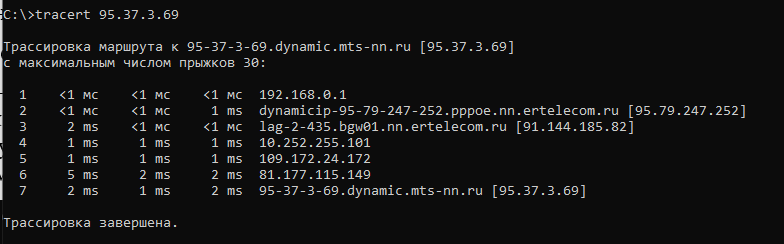


Рис 1. Трассирование устройства вне локальной сети

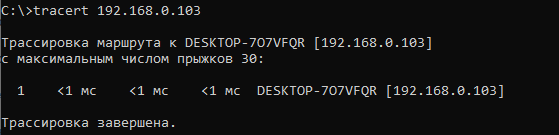


Рис 2. Трассирование устройства в локальной сети

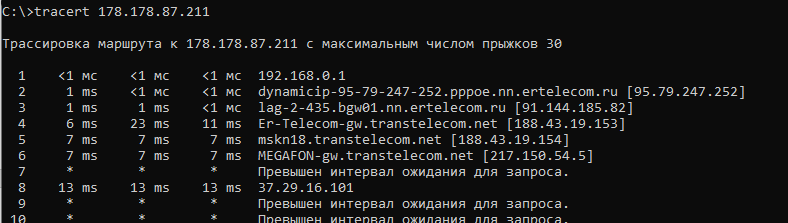


Рис 3. Трассирование устройства в моб. сети

# **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 (game.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, root, client\_: "SocketClient"): |
|  | super(Main, self).\_\_init\_\_(root) |
|  | self.client = client\_ |
|  | client\_.game = self |
|  | self.root = 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="#ebd7ca" |
|  | ) |
|  | 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=45, bg="#6e645d" |
|  | ) |
|  | self.entry.place(x=0, y=335) |
|  | send = Button( |
|  | self.root, |
|  | text="Отправить", |
|  | font=self.mini\_button\_font, |
|  | command=self.send\_button, |
|  | width=9, |
|  | height=1, |
|  | ) |
|  | send.place(x=340, y=325) |
|  |  |
|  |  |
|  | class SocketClient: |
|  | def \_\_init\_\_(self, name: str): |
|  | self.client = None |
|  | self.name = name |
|  | self.game = 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()) |
|  | for btn in self.game.game\_buttons: |
|  | btn["state"] = tk.NORMAL |
|  |  |
|  | 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"] |
|  |  |
|  | # todo: handler |
|  | 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\_\_": |
|  | main\_root = Tk() |
|  | main\_root.geometry("430x360+200+200") |
|  | main\_root.title("Камень, ножницы, бумага") |
|  | main\_root.resizable(False, False) |
|  | main\_root["bg"] = "#FFF" |
|  | nick = f"kesha{random.randint(0, 100000)}" |
|  | print(nick) |
|  | client = SocketClient(name=nick) |
|  | app = Main(main\_root, client) |
|  | app.pack() |
|  |  |
|  | game\_thread = threading.Thread(target=main\_root.mainloop) |
|  | socket\_thread = threading.Thread( |
|  | target=client.socket\_start, args=("127.0.0.1", 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.

# **Список литературы**

1. Transmission Control Protocol // Wikipedia URL: https://ru.wikipedia.org/wiki/Transmission\_Control\_Protocol (дата обращения: 10.01.2023).
2. Write a Multithreaded Server in Python // TechBeamers URL: https://techbeamers.com/python-tutorial-write-multithreaded-python-server/ (дата обращения: 15.01.2023).
3. IP-адрес // Wikipedia URL: https://ru.wikipedia.org/wiki/IP-%D0%B0%D0%B4%D1%80%D0%B5%D1%81 (дата обращения: 10.01.2023).