Федеральное Государственное бюджетное образовательное учреждение высшего 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:**

**«Разработка онлайн-игры»**

Выполнил студент 3 курса гр.ИС-30: Красильников.Н.Д

Проверил: старший преподаватель: Морозов Н.С.

Содержание

[Содержание 2](#_Toc125486972)

[**Цель работы** 2](#_Toc125486973)

[**Задачи** 3](#_Toc125486974)

[**Теория** 4](#_Toc125486975)

[**IP-адресация** 4](#_Toc125486976)

[**Протокол связи TCP** 6](#_Toc125486977)

[**Заголовок сегмента TCP** 7](#_Toc125486978)

[**Механизм действия протокола** 8](#_Toc125486979)

[**Топология сети тестирования** 8](#_Toc125486980)

[**Настройки сети для игры онлайн.** 9](#_Toc125486981)

[**Листинг программы** 11](#_Toc125486982)

[**Серверная часть (gameserver.py)** 11](#_Toc125486983)

[**Клиентская часть (clientgame.py)** 14](#_Toc125486984)

[**Заключение** 19](#_Toc125486985)

[**Список литературы** 20](#_Toc125486986)

# **Цель работы**

* Изучить систему маршрутизации;
* Изучить систему стека TCP/IP;

# **Задачи**

1. Написать на языке Python игру Камень-Ножницы-Бумага для игры в сети Интернет.
2. Настройка протокола транспортного уровня
3. Настройка сокетов серверной и клиентской части игры

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

# **Теория**

## **IP-адресация**

**IP-адрес** (aй-пи адрес, сокращение от англ. Internet Protocol Address) — уникальный идентификатор (адрес) устройства (обычно компьютера), подключённого к локальной сети или интернету.

IP-адрес представляет собой 32-битовое (по версии IPv4) или 128-битовое (по версии IPv6) двоичное число. Удобной формой записи IP-адреса (IPv4) является запись в виде четырёх десятичных чисел (от 0 до 255), разделённых точками, например, 192.168.0.1. (или 128.10.2.30 — традиционная десятичная форма представления адреса, а 10000000 00001010 00000010 00011110 — двоичная форма представления этого же адреса).

IP-адреса представляют собой основной тип адресов, на основании которых сетевой уровень протокола IP передаёт пакеты между сетями. IP-адрес назначается администратором во время конфигурирования компьютеров и маршрутизаторов.

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 (192.168.0.0/16, 172.16.0.0/12 или 10.0.0.0/8). Если же сеть должна работать как составная часть Интернета, то адрес сети выдаётся провайдером либо pегиональным интернет-регистратором (Regional Internet Registry, RIR). Всего существует пять RIR: ARIN, обслуживающий Северную Америку; APNIC, обслуживающий страны Юго-Восточной Азии; AfriNIC, обслуживающий страны Африки; LACNIC, обслуживающий страны Южной Америки и бассейна Карибского моря; и RIPE NCC, обслуживающий Европу, Центральную Азию, Ближний Восток. Региональные регистраторы получают номера автономных систем и большие блоки адресов у ICANN, а затем выдают номера автономных систем и блоки адресов меньшего размера локальным интернет-регистраторам (Local Internet Registries, LIR), обычно являющимся крупными провайдерами.

Номер узла в протоколе 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 связи TCP**

**TCP** - это протокол транспортного уровня, предоставляющий транспортировку (передачу) потока данных, с необходимостью предварительного установления соединения, благодаря чему гарантирует уверенность в целостности получаемых данных, также выполняет повторный запрос данных в случае потери данных или искажения. Помимо этого протокол TCP отслеживает дублирование пакетов и в случае обнаружения - уничтожает дублирующиеся пакеты. TCP - это аббревиатура от **T**ransmission **C**ontrol **P**rotocol (Протокол Управления Передачей) - является обязательным протоколом стандарт **TCP/IP**, определенный в стандарте RFC 793, "Transmission Control Protocol (TCP)".

В отличие от протокола **UDP** гарантирует целостность передаваемых данных и подтверждения отправителя о результатах передачи. Используется при передаче файлов, где потеря одного пакета может привести к искажению всего файла.

TCP обеспечивает свою надежность благодаря следующему:

* Данные от приложения разбиваются на блоки определенного размера, которые будут отправлены.
* Когда TCP посылает сегмент, он устанавливает таймер, ожидая, что с удаленного конца придет подтверждение на этот сегмент. Если подтверждение не получено по истечении времени, сегмент передается повторно.
* Когда TCP принимает данные от удаленной стороны соединения, он отправляет подтверждение. Это подтверждение не отправляется немедленно, а обычно задерживается на доли секунды
* TCP осуществляет расчет контрольной суммы для своего заголовка и данных. Это контрольная сумма, рассчитываемая на концах соединения, целью которой является выявить любое изменение данных в процессе передачи. Если сегмент прибывает с неверной контрольной суммой, TCP отбрасывает его и подтверждение не генерируется. (Ожидается, что отправитель отработает тайм-аут и осуществит повторную передачу.)
* Так как TCP сегменты передаются в виде IP датаграмм, а IP датаграммы могут прибывать беспорядочно, также беспорядочно могут прибывать и TCP сегменты. После получения данных TCP может по необходимости изменить их последовательность, в результате приложение получает данные в правильном порядке.
* Так как IP датаграмма может быть продублирована, принимающий TCP должен отбрасывать продублированные данные.
* TCP осуществляет контроль потока данных. Каждая сторона TCP соединения имеет определенное пространство буфера. TCP на принимающей стороне позволяет удаленной стороне посылать данные только в том случае, если получатель может поместить их в буфер. Это предотвращает от переполнения буферов медленных хостов быстрыми хостами.

### **Заголовок сегмента TCP**



### **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.

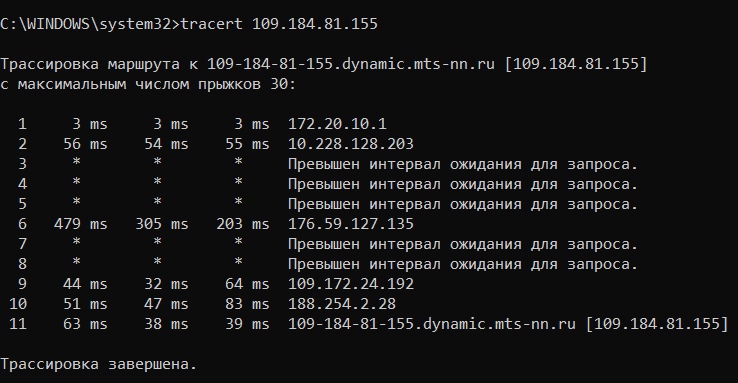


Рис 1. Трассирование устройства вне локальной сети через мобильную точку доступа к серверу

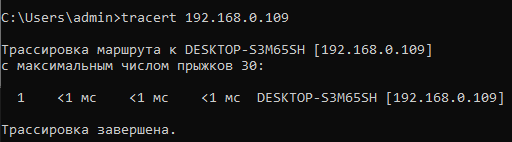


Рис 2. Трассирование устройства в локальной сети

# **Настройки сети для игры онлайн.**

Для настройки игры онлайн, мы должны узнать IPv4 компьютера в локальной сети.

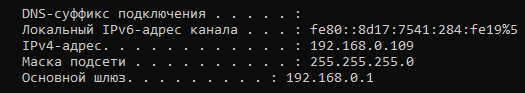


Рис.3 узнаем свой адрес в локальной сети

Этот ip-адрес вводим в код серверной части приложения и добавляем порт на который будет производиться переадресация, задаем адресс для понимания приложения на каком пк мы находимся.

Далее заходим в настройки роутера, и создаем правило переадресации портов, для своего сервера, для соединения по сети Интернет. (у меня это переадресация с порта 61911 на порт 61911)

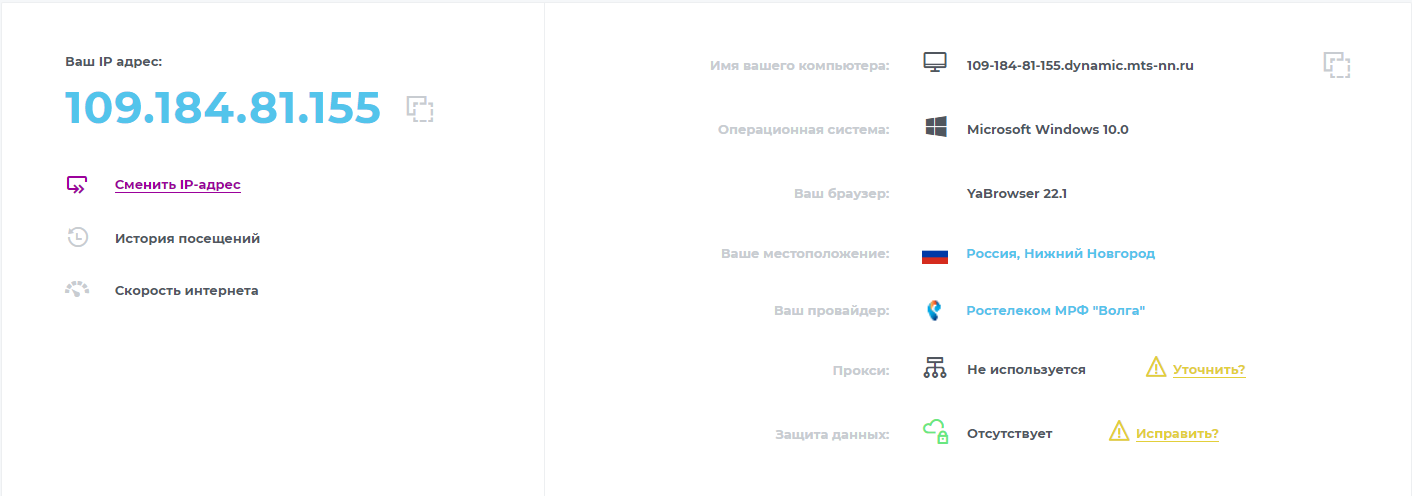


Рис.4 ip-адрес сети в которой находится сервер

В клиенской части приложения вводим ip-адрес сети в которой находится сервер, а так же порт с которого идет переадресация.  
  
Все, теперь мы можем играть по сети онлайн.

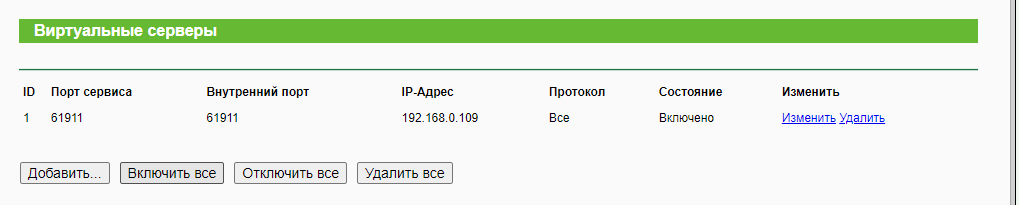


Рис.5 Создания правила переадресации порта

# **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)

while True:

client, address = self.sock.accept()

threading.Thread(target=self.client\_handler, args=(client,)).start()

self.clients.append(client)

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"]

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(e)

self.clients.remove(client)

client.close()

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("192.168.0.109", 61911).listen()

## **Клиентская часть (game.py)**

import json

import threading

import socket

from typing import Optional

from enum import IntEnum

from tkinter import Tk, Frame, Button, Label, END

import tkinter as tk

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"Это ник компа : {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=110)

self.game\_buttons[1].place(x=155, y=100, width=120, height=110)

self.game\_buttons[2].place(x=300, y=100, width=120, height=110)

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="#c3d7df")

self.txt.configure(state="disabled")

self.txt.place(x=430, y=0)

self.entry = tk.Entry(self.root, font=self.mini\_button\_font, width=36, bg="white")

self.entry.place(x=430, y=177)

send = Button(self.root,text="Отправить",font=self.mini\_button\_font,command=self.send\_button,width=9,height=2,)

send.place(x=764, y=160)

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("860x220+200+200")

main\_root.title("Камень, ножницы, бумага")

main\_root.resizable(False, False)

main\_root["bg"] = "#FFF"

nick = "Это получает ноут"

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=("109.184.81.155", 61911))

socket\_thread.start()

game\_thread.run()

# **Заключение**

Разработали онлайн игру «Камень-Ножницы-Бумага» на языке программирования Python, используя библиотеку tkinter для графического интерфейса, а также библиотеку Socket для серверной части. В процессе разработки игры было принято решение прописать новый порт в домашнем роутере для игры по сети Интернет на двоих человек. В результате была написана программа для серверно-клиентного взаимодействия, а также взаимодействие двух различных устройств.

# **Список литературы**

1. Transmission Control Protocol // Wikipedia URL: https://ru.wikipedia.org/wiki/Transmission\_Control\_Protocol (дата обращения: 18.12.2022).
2. Write a Multithreaded Server in Python // TechBeamers URL: https://techbeamers.com/python-tutorial-write-multithreaded-python-server/ (дата обращения: 19.12.2022).
3. Socket — Low-level networking interface / docs.python.org: [сайт]. URL: <https://docs.python.org/3/library/socket.html> (дата обращения: 17.12.2022).
4. Tkinter — Python interface to Tcl/Tk / docs.python.org: [сайт]. URL: <https://docs.python.org/3/library/tkinter.html> (дата обращения: 18.12.2022).
5. Что такое IP-Адрес [Электронный ресурс]. URL: https://2ip.ru/article/ip/ (Дата обращения: 15.12.2022).
6. Статья «Протокол TCP» [Электронный ресурс] URL: https://pc.ru/docs/network/tcp (дата обращения:17.12.2022)
7. Статья «TCP» [Электронный ресурс] URL: https://github.com/teach-telecom/telecom-book-2015/blob/master/transport/TCP.md (дата обращения 17.12.2022)
8. Статья «Теория:Протоколы:TCP» [Электронный ресурс] URL: https://mikrotik.wiki/wiki/Теория:Протоколы:TCP#Механизм\_действия\_протокола (дата обращения 16.12.2022)
9. Иструкция «Как открыть порты у роутера TP-Link?» [Электронный ресурс] URL: https://www.tp-link.com/ru/support/faq/72/ (дата обращения 15.12.2022)