ФГБОУ ВО

ННГАСУ

Кафедра информационных систем и технологий

Курсовая работа

Инфокоммуникационные системы и сети

**Разработка онлайн игры**

Выполнила студент 3 курса группы ИС-30

Варламов Д. Д. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Проверил преподаватель

Морозов Н.С. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Нижний Новгород

2022 год

Оглавление

[Цель работы 3](#_Toc125476203)

[Задачи 3](#_Toc125476204)

[Теоретическая часть 4](#_Toc125476205)

[**IP** 4](#_Toc125476206)

[**Маршрутизация** 5](#_Toc125476207)

[**Стек TCP/IP** 6](#_Toc125476208)

[**Преобразование IP-адресов в символьные адреса** 7](#_Toc125476209)

[**Маска подсети** 7](#_Toc125476210)

[Заключение 12](#_Toc125476211)

[Список использованной литературы 13](#_Toc125476212)

[Приложение: листинг кода программы 14](#_Toc125476213)

# Цельработы

Разработка онлайн-игры с отправкой пакетов, используя сокеты, а так же протокол транспортного уровня

# 

# ****Задачи****

1. Изучить систему маршрутизации;
2. Произвести настройку протокола транспортного уровня;
3. Произвести настройку сокетов серверной и клиентской части игры;
4. Изучить систему стека TCP/IP;

# Теоретическая часть

## **IP**

IP-адрес — уникальный сетевой адрес узла в компьютерной сети, построенной на основе стека протоколов TCP/IP. IP-адрес представляет собой серию из 32 двоичных бит (единиц и нулей).

Так как человек невосприимчив к большому однородному ряду чисел, такому как этот 11100010101000100010101110011110 (здесь, к слову, 32 бита информации, так как 32 числа в двоичной системе), было решено разделить ряд на четыре 8-битных байта и получилась следующая последовательность: 11100010.10100010.00101011.10011110.

Это не сильно облегчило жизнь и было решение перевести данную последовательность в, привычную нам, последовательность из четырёх чисел в десятичной системе, то есть 226.162.43.158. 4 разряда также называются октетами. Данный IP адрес определяется протоколом IPv4. По такой схеме адресации можно создать более 4 миллиардов IP-адресов.

Максимальным возможным числом в любом октете будет 255 (так как в двоичной системе это 8 единиц), а минимальным – 0.

IP-адреса делятся на 5 классов (A, B, C, D, E).

* **Класс A**. Старший бит в адресах такого формата всегда равен 0. За идентификацию сети отвечает начальный октет, позволяющий разместить 127 уникальных сетей. Оставшиеся 3 октета используются для обозначения узлов, максимальное количество которых составляет 17 млн. на каждую сеть.
* **Класс B**. Первые биты IP-адреса равны 10. Начальные два октета относятся к идентификатору сети, а последние два – к идентификатору узла. Возможно создание 16384 сетей, каждая из которых поддерживает размещение 65000 узлов.
* **Класс C**. Начальные биты IP-адреса равны 110. За идентификацию сети отвечают первые три октета, позволяющие создать 2 млн. сетей. Последний октет отводится для идентификации узлов, максимальное число которых составляет 254 на каждую сеть.
* **Класс D**. Запись IP-адреса начинается с битов 1110. В сетях подобного формата используется широковещательная рассылка сообщений нескольким узлам.
* **Класс E**. IP-адреса зарезервированы для использования в будущем. Первые биты всегда равны 11110.

Так же IP-адреса делятся на «цвета» - белые и серые (или публичные и частные). Публичным IP адресом называется IP адрес, который используется для выхода в Интернет. Адреса, используемые в локальных сетях, относят к частным. Частные IP не маршрутизируются в Интернете.

## **Маршрутизация**

**Маршрутизация** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Routing*) — процесс определения оптимального маршрута данных в сетях связи.

Маршруты могут задаваться административно ([статические маршруты](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BC%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F)), либо вычисляться с помощью [алгоритмов маршрутизации](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B_%D0%BC%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8), базируясь на информации о [топологии](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_%D1%82%D0%BE%D0%BF%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F) и состоянии сети, полученной с помощью [протоколов маршрутизации](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB_%D0%BC%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8) (динамические маршруты).

Статическими маршрутами могут быть:

маршруты, не изменяющиеся во времени;

маршруты, изменяющиеся по расписанию;

Маршрутизация в компьютерных сетях выполняется специальными программно-аппаратными средствами — [маршрутизаторами](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80); в простых конфигурациях может выполняться и компьютерами общего назначения, соответственно настроенными.

**Маршрутизаторы** представляют собой либо специализированные вычислительные машины, либо компьютеры с несколькими IP-интерфейсами, работа которых управляется специальным программным обеспечением.

Маршрутизация состоит из двух этапов:

1. На первом этапе происходит изучение сети, какие подсети есть в этой составной сети, какие маршрутизаторы и как эти маршрутизаторы объединены между собой.
2. Второй этап маршрутизации выполняется когда сеть уже изучена и на маршрутизатор поступил пакет, для этого пакета нужно определить куда именно его отправить. Иногда для второго этапа маршрутизации используется отдельный термин “продвижение” (forwarding).

В ряде случаев маршрутизатор может преобразовывать заголовок пакета, заменяя адреса отправителя и/или получателя пакета. В частности, это происходит при взаимодействии локальной сети (имеющей свои адреса) с глобальной сетью Интернет. В этом случае локальная сеть может быть видна извне по одному глобальному IP-адресу.

Для того чтобы маршрутизатор мог направлять пакеты с одним глобальным адресом тем или иным получателям в локальной сети, используется таблица NAT, где помимо IP-адресов указываются порты, идентифицирующие приложения, устанавливающие соединение. При этом номера портов указаны не в заголовке IP-пакета, а в заголовке сегмента TCP либо UDP (сегменты инкапсулируются в поле данных IP-пакетов). Это позволяет осуществлять взаимно-однозначную идентификацию получателя и отправителя в тех случаях, когда за одним глобальным адресом находится множество компьютеров локальных сетей.

## **Стек TCP/IP**

Протокол TCP/IP – это целая сетевая модель, описывающая способ передачи данных в цифровом виде. На правилах, включенных в нее, базируется работа интернета и локальных сетей независимо от их назначения и структуры.

Протокол TCP/IP основан на OSI и так же, как предшественник, имеет несколько уровней, которые и составляют его архитектуру. Всего выделяют 4 уровня – канальный (интерфейсный), межсетевой, транспортный и прикладной.

## **Преобразование IP-адресов в символьные адреса**

Технология активно используется для назначения буквенно-цифровых названий веб-ресурсов. При вводе домена в адресной строке браузера сначала происходит обращение к специальному серверу [DNS](https://timeweb.com/ru/community/articles/vvedenie-v-terminologiyu-elementy-i-ponyatiya-dns-1). Он всегда прослушивает порт 53 у всех компьютеров, которые подключены к интернету, и по запросу преобразует введенное название в стандартный IP-адрес.

После определения точного местонахождения файлов сайта включается обычная схема работы – от прикладного уровня с кодированием данных до обращения к физическому оборудованию на уровне сетевых интерфейсов. Процесс называется инкапсуляцией информации. На принимающей стороне происходит обратная процедура – декапсуляция.

## **Маска подсети**

Устройства различают части IP-адреса при помощи **маски подсети** – 32-битной строки, разделённой на 4 октета, как и IP-адрес. При установке соединения каждый октет IP-адреса сопоставляется с октетом маски подсети.

По умолчанию в стандартной домашней сети маска подсети имеет вид: 255.255.255.0.

В примере маска IP-адреса указана в десятичном представлении и содержит числа «255» и «0». Первое отвечает за идентификацию сети, а второе за обозначение конечного узла.

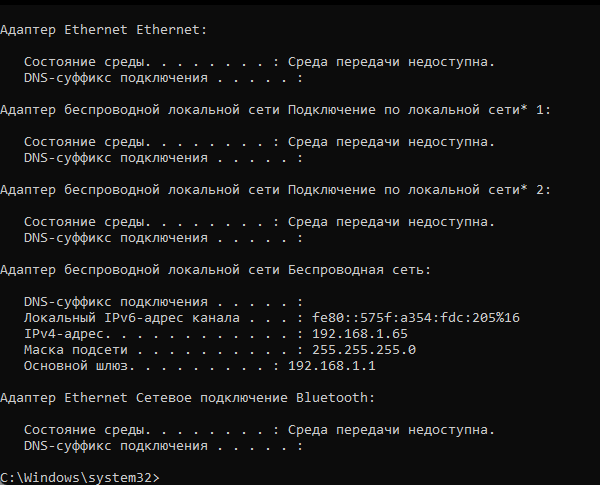


Рисунок 1 - Результат работы команды ipconfig

Чтобы узнать свой глобальный (внешний) IP-адрес (выдается провайдером и помогает устройству подключаться к сети Интернет) достаточно перейти на сайт  [2ip.ru](https://2ip.ru/).

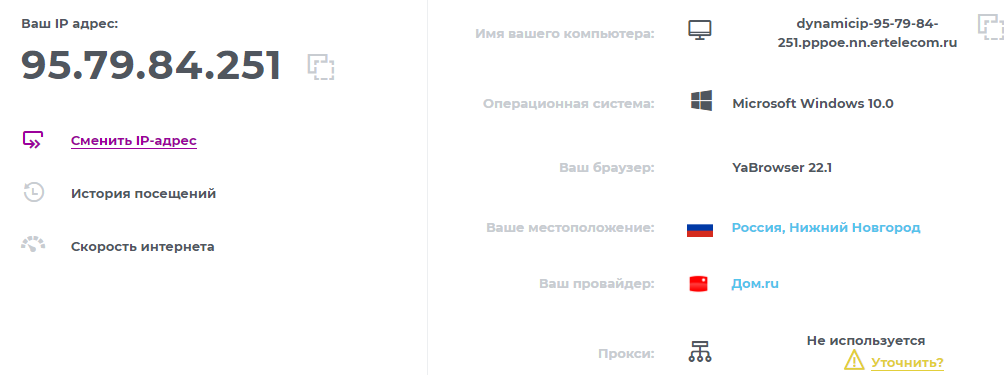


Рисунок 2 – Сервис 2ip.ru

**Топология сети тестирования:**

Разрабатываемая программа будет соединять пару компьютеров посредством связи клиент-сервер. С помощью команды tracert рассмотрим маршрут, через который проходят пакеты данных:

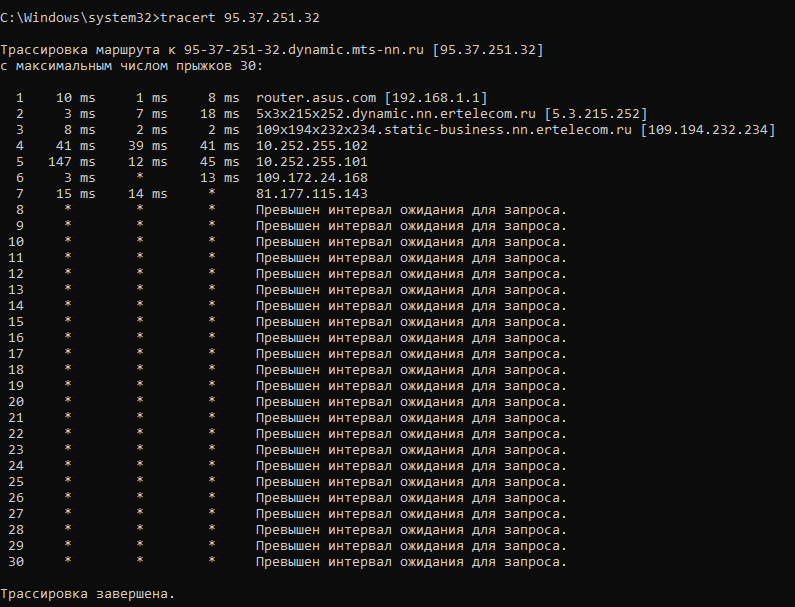
1. при подключении к другому компьютеру, который использует беспроводное соединение к сети Интернет: 

Рисунок 3 – Результат команды tracert для домашней сети

Следовательно, при передаче данных от одного компьютера к другому, пакеты проходят через 7 узлов.

1. при подключении к устройству, используемому мобильную точку доступа:

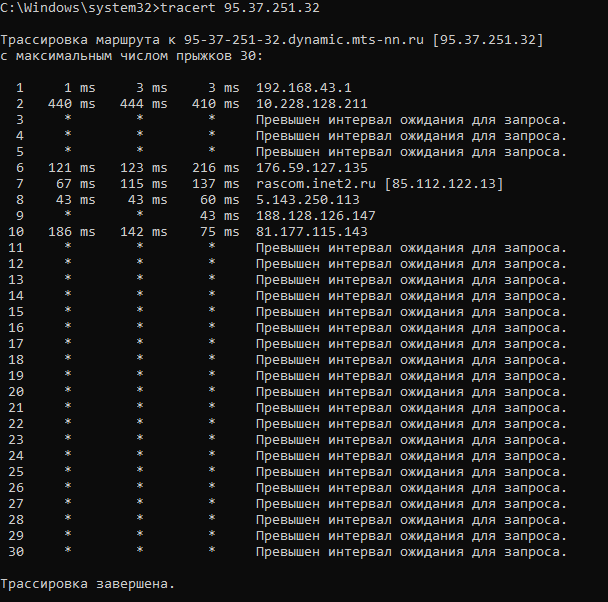


Рисунок 4 – Результаты команды tracert для

мобильной точки доступа

При передаче данных на мобильное устройство пакеты проходят через 10 узлов.

# Заключение

В ходе выполнения курсовой работы была рассмотрена реализация передачи данных через протокол TCP. Разработка приложения потребовала изучения таких аспектов, как: IP-адресация, маршрутизация стека TCP/IP, работа с сокетами клиентской и серверной части.

В результате была разработана игра «Камень, Ножницы, Бумага», предназначенная для сетевой игры, используя Python 3, библиотеку tkinter, а также модуль socket. Также были проверены отличия подключения через домашний интернет и мобильную точку доступа. В результате данной проверки домашний интернет оказался эффективнее, чем мобильная точка доступа, так как на достижение цели понадобилось на 3 шага меньше.

# Список использованной литературы

1. [Socket](https://docs.python.org/3/library/socket.html#module-socket) — Low-level networking interface / docs.python.org: [сайт]. URL: <https://docs.python.org/3/library/socket.html> (дата обращения: 17.12.2022).
2. [Tkinter](https://docs.python.org/3/library/tkinter.html#module-tkinter) — Python interface to Tcl/Tk / docs.python.org: [сайт]. URL: <https://docs.python.org/3/library/tkinter.html> (дата обращения: 16.12.2022).
3. *Джо Брокмайер, Ди-Анн Лебланк, Рональд Маккарти, мл.* Маршрутизация в Linux = Linux Routing. — М.: [«Вильямс»](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2002. — С. 240. — [ISBN 1-57870-267-4](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/1578702674).
4. Линков В. В, «Всё об IP адресах и о том, как с ними работать» [Электронный ресурс]. URL: https://habr.com/ru/post/350878/ (Дата обращения: 15.12.2022).
5. Морковкин Егор Андреевич, Новичихина Алёна Александровна, Замулин Иван Сергеевич IP-адресация и информационная безопасность [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/ip-adresatsiya-i-informatsionnaya-bezopasnost/viewer (Дата обращения: 17.12.2022).
6. Infinet wireless, «Понятие маршрутизации» [Электронный ресурс]. URL: https://wiki.infinetwireless.com/pages/viewpage.action?pageId=76808266 (Дата обращения: 15.12.2022).
7. Статья «Протоколы передачи данных» [Электронный ресурс] URL: https://tproger.ru/explain/protokoly-peredachi-dannyh-chto-jeto-kakie-byvajut-i-v-chjom-razlichija/

# Приложение: листинг кода программы

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}\n Проигрышей:" f" {self.lose}\n Ничей: {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)

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))

        print("Соединение установлено")

        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)

    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("430x160+200+200")

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

    main\_root.resizable(False, False)

    main\_root["bg"] = "#FFF"

    nick = "Danil"

    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=("95.37.251.32", 3383)

    )

    socket\_thread.start()

    game\_thread.run()