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: «Разработка онлайн-игры»

Выполнил студент 3 курса гр. ИС-29: Быляв И.Е.

Проверил старший преподаватель: Морозов Н.С.

2023г.

**Оглавление**

[**Введение** 3](#_Toc125011204)

[**Задание:** 3](#_Toc125011205)

[**Теоретическая часть** 4](#_Toc125011206)

[**IP-адресация** 4](#_Toc125011207)

[**Протокол связи TCP/IP** 6](#_Toc125011208)

[**Маршрутизация TCP/IP** 9](#_Toc125011209)

[**Программный код** 10](#_Toc125011210)

[**Сервер** 10](#_Toc125011211)

[**Клиент** 13](#_Toc125011212)

[**Список литературы** 20](#_Toc125011213)

# **Введение**

## **Задание:**

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

# **Теоретическая часть**

## **IP-адресация**

**IP-адрес — уникальный сетевой адрес узла в компьютерной сети, построенной на основе стека протоколов TCP/IP** (TCP/IP – это набор интернет-протоколов, о котором мы поговорим в дальнейших статьях). 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**.  
  
Далее давайте разберёмся с тем, что называется **классом IP** (именно в этом моменте в лабораторной работе была неточность).  
  
IP-адреса делятся на 5 классов (A, B, C, D, E). A, B и C — это классы коммерческой адресации. D – для многоадресных рассылок, а класс E – для экспериментов.

Класс А: 1.0.0.0 — 126.0.0.0, маска 255.0.0.0  
Класс В: 128.0.0.0 — 191.255.0.0, маска 255.255.0.0  
Класс С: 192.0.0.0 — 223.255.255.0, маска 255.255.255.0  
Класс D: 224.0.0.0 — 239.255.255.255, маска 255.255.255.255  
Класс Е: 240.0.0.0 — 247.255.255.255, маска 255.255.255.255

Теперь о **«цвете»** IP. IP бывают **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.  
  
Публичные адреса назначаются публичным веб-серверам для того, чтобы человек смог попасть на этот сервер, вне зависимости от его местоположения, то есть через Интернет. Например, игровые сервера являются публичными, как и сервера Хабра и многих других веб-ресурсов.  
Большое отличие частных и публичных IP адресов заключается в том, что используя частный IP адрес мы можем назначить компьютеру любой номер (главное, чтобы **не было совпадающих номеров**), а с публичными адресами всё не так просто. Выдача публичных адресов контролируется различными организациями.

## **Протокол связи TCP/IP**

Стек протоколов TCP/IP (Transmission Control Protocol/Internet Protocol, протокол управления передачей/протокол интернета) — сетевая модель, описывающая процесс передачи цифровых данных. Она названа по двум главным протоколам, по этой модели построена глобальная сеть интернет. Сейчас это кажется невероятным, но в 1970-х информация не могла быть передана из одной сети в другую. Чтобы обеспечить такую возможность, был разработан стек интернет-протоколов, известный как TCP/IP.

Разработка сетевой модели осуществлялась при содействии Министерства обороны США, поэтому иногда модель TCP/IP называют DoD (Department of Defence) модель. Если вы знакомы с моделью OSI, то вам будет проще понять построение модели TCP/IP, потому что обе модели имеют деление на уровни, внутри которых действуют определенные протоколы и выполняются собственные функции.

Существуют и другие версии описания модели, в том числе включающие иное количество уровней и их наименований. Мы рассмотрим четыре из них:

1. Канальный уровень (link layer)

Предназначение канального уровня — дать описание тому, как происходит обмен информацией на уровне сетевых устройств, определить, как информация будет передаваться от одного устройства к другому. Информация здесь кодируется, делится на пакеты и отправляется по нужному каналу, т.е. среде передачи.

Этот уровень также вычисляет максимальное расстояние, на которое пакеты возможно передать, частоту сигнала, задержку ответа и т.д. Все это — физические свойства среды передачи информации. На канальном уровне самым распространенным протоколом является Ethernet, который мы рассмотрим в конце статьи.

1. Межсетевой уровень (internet layer)

Глобальная сеть интернет состоит из множества локальных сетей, взаимодействующих между собой. Межсетевой уровень используется, чтобы описать обеспечение такого взаимодействия.

Межсетевое взаимодействие — это основной принцип построения интернета. Локальные сети по всему миру объединены в глобальную, а передачу данных между этими сетями осуществляют магистральные и пограничные маршрутизаторы.

Именно на межсетевом уровне функционирует протокол IP, позволивший объединить разные сети в глобальную. Как и протокол TCP, он дал название модели, рассматриваемой в статье.

1. Транспортный уровень (transport layer)

Постоянные резиденты транспортного уровня — протоколы TCP и UDP, они занимаются доставкой информации.

TCP (протокол управления передачей) — надежный, он обеспечивает передачу информации, проверяя дошла ли она, насколько полным является объем полученной информации и т.д. TCP дает возможность двум конечным устройствам производить обмен пакетами через предварительно установленное соединение. Он предоставляет услугу для приложений, повторно запрашивает потерянную информацию, устраняет дублирующие пакеты, регулируя загруженность сети. TCP гарантирует получение и сборку информации у адресата в правильном порядке.

UDP (протокол пользовательских датаграмм) — ненадежный, он занимается передачей автономных датаграмм. UDP не гарантирует, что всех датаграммы дойдут до получателя. Датаграммы уже содержат всю необходимую информацию, чтобы дойти до получателя, но они все равно могут быть потеряны или доставлены в порядке отличном от порядка при отправлении.

UDP обычно не используется, если требуется надежная передача информации. Использовать UDP имеет смысл там, где потеря части информации не будет критичной для приложения, например, в видеоиграх или потоковой передаче видео. UDP необходим, когда делать повторный запрос сложно или неоправданно по каким-то причинам.

Протоколы транспортного уровня не интерпретируют информацию, полученную с верхнего или нижних уровней, они служат только как канал передачи, но есть исключения. RSVP (Resource Reservation Protocol, протокол резервирования сетевых ресурсов) может использоваться, например, роутерами или сетевыми экранами в целях анализа трафика и принятия решений о его передаче или отклонении в зависимости от содержимого.

1. Прикладной уровень (application layer)

В модели TCP/IP отсутствуют дополнительные промежуточные уровни (представления и сеансовый) в отличие от OSI. Функции форматирования и представления данных делегированы библиотекам и программным интерфейсам приложений (API) — своего рода базам знаний, содержащим сведения о том, как приложения взаимодействуют между собой. Когда службы или приложения обращаются к библиотеке или API, те в ответ предоставляют набор действий, необходимых для выполнения задачи и полную инструкцию, каким образом эти действия нужно выполнять.

Протоколы прикладного уровня действуют для большинства приложений, они предоставляют услуги пользователю или обмениваются данными с «коллегами» с нижних уровней по уже установленным соединениям. Здесь для большинства приложений созданы свои протоколы. Например, браузеры используют HTTP для передачи гипертекста по сети, почтовые клиенты — SMTP для передачи почты, FTP-клиенты — протокол FTP для передачи файлов, службы DHCP — протокол назначения IP-адресов DHCP и так далее.

## **Маршрутизация TCP/IP**

Маршрутом называется путь, по которому пакеты пересылаются от отправителя к получателю. Маршрут определяет не полный путь, а только сегмент пути от хоста до шлюза (или от шлюза до шлюза), который может переслать пакеты целевому хосту. Список маршрутов хранится в таблице маршрутизации ядра. Описание маршрута содержит такую информацию, как список сетей, достижимых локальным хостом, и список шлюзов для отправки пакетов в удаленные сети. При получении дейтаграммы шлюз ищет в таблицах маршрутизации следующий узел ее маршрута до целевого хоста и отправляет дейтаграмму этому узлу.

В таблицу маршрутизации ядра можно добавлять несколько маршрутов к одному и тому же хосту. Процедура выбора маршрута сначала находит все маршруты, соответствующие запросу, а потом выбирает маршрут с минимальной метрикой расстояния. При наличии нескольких маршрутов одинаковой длины выбирается тот маршрут, который задан наиболее точно. Если несколько маршрутов совпадают по обоим критериям, то эти маршруты применяются по очереди.

# **Программный код**

## **Сервер**

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("127.0.0.1", 12345).listen()

## **Клиент**

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\_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}\nНичей: {self.draw}\nПроигрышей:" f" {self.lose}"

def send\_button(self):

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\_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:\nНет",

bg="#FFF",

)

self.game\_data\_label.place(x=5, y=5)

self.opponent\_label.place(x=300, y=0)

self.txt = tk.Text(

self.root, font=self.mini\_button\_font, width=45, height=8, bg="#CCCCCC"

)

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="#6e645d"

)

self.entry.place(x=10, y=335)

send = Button(

self.root,

text="Отправить",

font=self.mini\_button\_font,

command=self.send\_button,

width=8,

height=1,

)

send.place(x=339, 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"]

self.game.opponent\_label.configure(text=f"Оппонент:\n {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"my opponent"

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", 12345)

)

socket\_thread.start()

game\_thread.run()

# **Список литературы**

1. Wikipedia[Электронный ресурс] URL: https://ru.wikipedia.org/wiki/IP-адрес
2. Habr[Электронный ресурс] URL: <https://habr.com/ru/post/350878/>
3. Eternalhost[Электронный ресурс]URL:<https://eternalhost.net/blog/sistemnoe-administrirovanie/chto-takoe-ip-adres#p3>
4. Wikipedia[Электронныйресурс] URL:<https://ru.wikipedia.org/wiki/TCP/IP>
5. Академик[Электронный ресурс]<URL:https://dic.academic.ru/dic.nsf/ruwiki/647>