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 Смирнова А.Н.

Проверил Морозов Н.С.

Нижний Новгород

2022 г.

**Содержание**

[**Цель** **работы** 3](#_Toc125384868)

[**Задачи** 3](#_Toc125384869)

[**Теоретическая часть** 4](#_Toc125384870)

[***Маршрут до хоста*** 4](#_Toc125384871)

[***Маршрут к сети*** 4](#_Toc125384872)

[***Маршрут по умолчанию*** 4](#_Toc125384873)

[***Циклический маршрут*** 4](#_Toc125384874)

[***Маршрут оповещения*** 4](#_Toc125384875)

[***Топология сети тестирования*** 5](#_Toc125384876)

[**Пример работы программы** 13](#_Toc125384877)

[**Заключение** 14](#_Toc125384878)

[**Список литературы** 15](#_Toc125384879)

# **Цель работы**

Создание онлайн-игры с отправкой пакетов через сокеты, используя протокол транспортного уровня

# **Задачи**

1. изучение маршрутизации стека TCP/IP;
2. настройка протокола транспортного уровня;
3. настройка сокетов серверной и клиентской части игры.

# **Теоретическая часть**

Маршрутизация — процесс определения оптимального маршрута данных в сетях связи.

Маршруты могут задаваться административно (статические маршруты), либо вычисляться с помощью алгоритмов маршрутизации, базируясь на информации о топологии и состоянии сети, полученной с помощью протоколов маршрутизации (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; в простых конфигурациях может выполняться и компьютерами общего назначения, соответственно настроенными.

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. Существует пять типов маршрутов:

## ***Маршрут до хоста***

Определяет шлюз, который может переслать пакеты указанному хосту в другой сети.

## ***Маршрут к сети***

Определяет шлюз, который может переслать пакеты другому хосту указанной сети.

## ***Маршрут по умолчанию***

Определяет шлюз, которому будут отправлены пакеты, если не был задан маршрут до целевого хоста или маршрут к сети целевого хоста.

## ***Циклический маршрут***

Маршрут по умолчанию для всех пакетов, отправляемых по адресам локальной сети. IP-адрес циклического маршрута всегда 127.0.0.1.

## ***Маршрут оповещения***

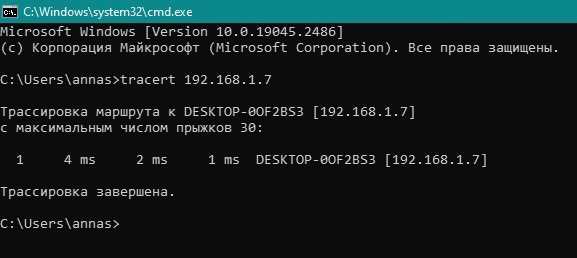
Маршрут по умолчанию для всех пакетов оповещения. Каждой подсети, в которой у сети есть IP-адрес, автоматически присваиваются два маршрута оповещения (один - адресу подсети, а другой - адресу оповещения подсети).

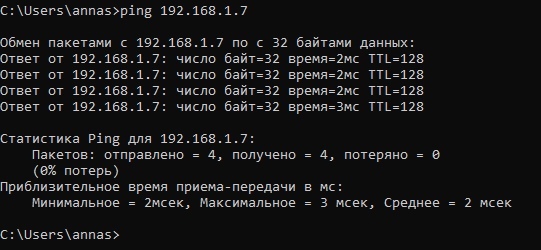
Список маршрутов хранится в таблице маршрутизации ядра. Описание маршрута содержит такую информацию, как список сетей, достижимых локальным хостом, и список шлюзов для отправки пакетов в удаленные сети. При получении дейтаграммы шлюз ищет в таблицах маршрутизации следующий узел ее маршрута до целевого 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.

## ***Топология сети тестирования***

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



**Код программы**

import json, threading, random, socket, tkinter as tk

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

class Main(Frame):

def \_\_init\_\_(self, root, client\_: 'SocketClient'):

super(Main, self).\_\_init\_\_(root)

self.client = client\_

client\_.game = self

self.root = root

self.game\_buttons = []

self.set\_opponent\_name('Some cool guy')

self.set\_my\_name('Nagibator777')

self.startUI()

self.opponent\_choise = IntVar()

def startUI(self):

self.game\_btns = [Button(root, text='Камень', font=('Times New Roman', 15),

command=lambda x=1: self.btn\_click(x)),

Button(root, text='Ножницы', font=('Times New Roman', 15),

command=lambda x=2: self.btn\_click(x)),

Button(root, text='Бумага', font=('Times New Roman', 15),

command=lambda x=3: self.btn\_click(3))]

self.game\_btns[0].place(x=10, y=100, width=120, height=50)

self.game\_btns[1].place(x=155, y=100, width=120, height=50)

self.game\_btns[2].place(x=300, y=100, width=120, height=50)

self.lbl = Label(root, text='Начало игры!', bg='#FFF',

font=('Times New Roman', 18, 'bold'))

self.lbl.place(x=150, y=5)

self.win = self.drow = self.lose = 0

self.lbl2 = Label(root, justify='left', font=('Times New Roman', 13),

text=f'Побед: {self.win}\nПроигрышей:'

f' {self.lose}\nНичей: {self.drow}',

bg='#FFF')

self.lbl3 = Label(root, justify='right', font=('Times New Roman', 13),

text=f'Оппонент: {self.opponent\_name}',

bg='#FFF')

self.lbl2.place(x=5, y=5)

self.lbl3.place(x=145, y=55)

self.txt = tk.Text(root, font=('Times New Roman', 12), width=51, height=8, bg='#f0f8ff')

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(root, font=('Times New Roman', 12), width=40, bg='#f0f8ff')

self.entry.place(x=10, y=330)

send = Button(root, text='Отправить', font=('Times New Roman', 12),

command=self.send\_button, width=8, height=1)

send.place(x=340, y=325)

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 btn\_click(self, choise):

self.choise = choise

for btn in self.game\_btns:

btn['state'] = tk.DISABLED

self.lbl3.configure(text=f'Оппонент: {self.opponent\_name}')

self.client.send('action', str(choise))

root.wait\_variable(self.opponent\_choise)

self.calc\_result(choise, self.get\_opponent\_choise())

for btn in self.game\_btns:

btn['state'] = tk.NORMAL

def calc\_result(self, choise, opp\_choise):

if choise == opp\_choise:

self.drow += 1

self.lbl.configure(text='Ничья')

elif choise == 1 and opp\_choise == 2 \

or choise == 2 and opp\_choise == 3 \

or choise == 3 and opp\_choise == 1:

self.win += 1

self.lbl.configure(text='Победа')

else:

self.lose += 1

self.lbl.configure(text='Проигрыш')

print(f'Ход оппонента: {opp\_choise}')

self.lbl2.configure(text=f'Побед: {self.win}\nПроигрышей:'

f' {self.lose}\nНичьей: {self.drow}')

self.set\_opponent\_choise = IntVar()

def set\_my\_name(self, name):

self.my\_name = name

def get\_my\_name(self):

return self.my\_name

def set\_opponent\_name(self, name):

self.opponent\_name = name

def is\_opponent\_chosen(self):

return self.opponent\_choise != 'None'

def set\_opponent\_choise(self, opp\_choise):

root.after(20, self.opponent\_choise.set, opp\_choise)

def get\_opponent\_choise(self):

return self.opponent\_choise.get()

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.lbl.configure(text='Ничья')

if message == 'win':

self.game.win += 1

self.game.lbl.configure(text='Победа')

if message == 'lose':

self.game.lose += 1

self.game.lbl.configure(text='Проигрыш')

self.game.lbl2.configure(text=f'Побед: {self.game.win}\nПроигрышей: {self.game.lose}\nНичей: {self.game.drow}')

for btn in self.game.game\_btns:

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.lbl3.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\_\_':

print(f'Игра запущена')

root = Tk()

root.geometry('430x360+200+200')

root.title('Камень, ножницы, бумага')

root.resizable(False, False)

root['bg'] = '#FFF'

nick = f'Человек №{random.randint(0, 100000)}'

print(f'Ваше имя: {nick}')

client = SocketClient(name=nick)

app = Main(root, client)

app.pack()

game\_thread = threading.Thread(target=root.mainloop)

socket\_thread = threading.Thread(

target=client.socket\_start, args=('192.168.1.2', 8081))

socket\_thread.start()

game\_thread.run()

import json, socket, threading

from typing import Optional, Union

class Server:

def \_\_init\_\_(self, host, port):

self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.sock.bind((host, 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()

print('Подключено:', address)

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 = 'lose'

opponent\_result = 'win'

if user\_choice == opponent\_choice:

user\_result = 'draw'

opponent\_result = 'draw'

if (user\_choice + 1) % 3 == opponent\_choice:

user\_result = 'win'

opponent\_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()

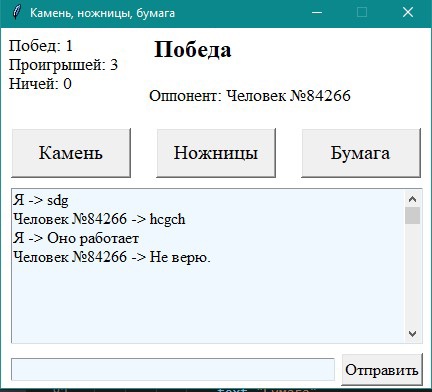
)

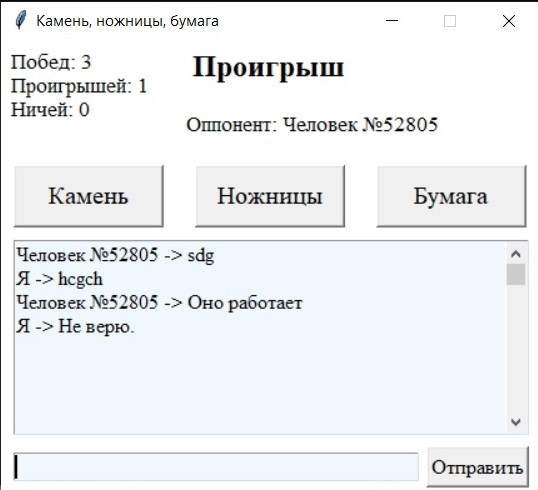
if \_\_name\_\_ == '\_\_main\_\_':

print('Сервер запущен')

Server('192.168.1.2', 8081).listen()

# **Пример работы программы**

****

****

# **Заключение**

В ходе выполнения курсовой работы мы разработали онлайн-игру «Камень, ножницы, бумага». При помощи связывания двух вычислительных машин. В результате было разработано приложение на языке программирования Python, интерфейс был создан с помощью библиотеки tkinter. Подключение происходит через сокеты.

Разбиение, передаваемых данных на пакеты соответствующего размера, отправка этих пакетов в сеть и обработка на другой стороне происходит полностью с помощью протокола TCP. А за исправление ошибок, сохранение последовательности данных и обработку доставки отвечает сокет.

# **Список литературы**

1. Всё об IP-адресах и о том, как с ними работать [Электронный ресурс]. URL: <https://cyberleninka.ru/article/n/ip-adresatsiya-i-informatsionnaya-bezopasnost/viewer> (Дата обращения: 18.12.2022).
2. Руководство по программированию сокетов на Python [Электронный ресурс]. URL: <https://habr.com/ru/company/skillfactory/blog/690186/> (Дата обращения: 18.12.2022).
3. Курс «Компьютерные сети» [Электронный ресурс] URL: <http://math.gsu.by/wp-content/uploads/courses/networks/построение_больших_сетей> (Дата обращения: 18.12.2022).
4. Статья «Протоколы передачи данных» [Электронный ресурс] URL: <https://tproger.ru/explain/protokoly-peredachi-dannyh-chto-jeto-kakie-byvajut-i-v-chjom-razlichija/> (Дата обращения: 18.12.2022).
5. Документация IBM«Маршрутизация TCP/IP»[Электронныйресурс] <URL:https://www.ibm.com/docs/ru/aix/7.1?topic=protocol-tcpip-routing> (Дата обращения 18.12.2022)
6. Очень простой чат(клиент/сервер) на Python.[Электронныйресурс] URL: <https://xn--90aeniddllys.xn--p1ai/ochen-prostoj-chatklient-server-na-python/> (Дата обращения 18.12.2022)
7. Таблица «безопасных» цветов от студии Артемия Лебедева [Электронный ресурс] URL: <https://www.artlebedev.ru/colors/> (Дата обращения 1.12.2022)
8. Конвертер цветов из одной цветовой модели в ряд других [Электронный ресурс] URL: <https://colorscheme.ru/color-converter.html> (Дата обращения 18.12.2022)