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: Смирнов А.О.

Проверил старший преподаватель: Морозов Н.С.

2023г.

**Содержание**

[Цель работы 3](#_Toc125065591)

[Задачи 4](#_Toc125065592)

[Теоретическая часть 5](#_Toc125065593)

[I. Протокол связи 5](#_Toc125065594)

[1. IP — Internet Protocol 5](#_Toc125065595)

[2. TCP/IP — Transmission Control Protocol/Internet Protocol 5](#_Toc125065596)

[3. UDP — User Datagram Protocol 5](#_Toc125065597)

[4. FTP — File Transfer Protocol 6](#_Toc125065598)

[5. DNS 6](#_Toc125065599)

[6. HTTP — HyperText Transfer Protocol 6](#_Toc125065600)

[7. NTP – Network Time Protocol 7](#_Toc125065601)

[8. SSH – Secure SHell 7](#_Toc125065602)

[II. IP-адресация 8](#_Toc125065603)

[III. Маршрутизация TCP/IP 9](#_Toc125065604)

[Интерфейс приложения 11](#_Toc125065605)

[Код программы 12](#_Toc125065606)

[I. Серверная часть приложения 12](#_Toc125065607)

[II. Клиентская часть приложения 14](#_Toc125065608)

[Выводы 18](#_Toc125065609)

[Список литературы 19](#_Toc125065610)

# Цель работы

Разработать игру “Цу Е Фа” с чатом на языке программирования Python для игры по локальной сети.

## Задачи

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

## Теоретическая часть

## Протокол связи

### IP — Internet Protocol

Протокол передачи, который первым объединил отдельные компьютеры в единую сеть. Самый примитивный в этом списке. Он является ненадёжным, т. е. не подтверждает доставку пакетов получателю и не контролирует целостность данных. По протоколу IP передача данных осуществляется без установки соединения.

Основная задача этого протокола — маршрутизация датаграмм, т. е. определение пути следования данных по узлам сети.

Популярная версия на текущий момент — IPv4 с 32-битными адресами. Это значит, что в интернете могут хранится 4.29 млрд адресов IPv4. Число большое, но не бесконечное. Поэтому существует версия IPv6, которая поможет решить проблему переполнения адресов, ведь уникальных IPv6 будет 2 ^ 128 адресов (число с 38 знаками).

### TCP/IP — Transmission Control Protocol/Internet Protocol

Это стек протоколов TCP и IP. Первый обеспечивает и контролирует надёжную передачу данных и следит за её целостностью. Второй же отвечает за маршрутизацию для отправки данных. Протокол TCP часто используется более комплексными протоколами.

### UDP — User Datagram Protocol

Протокол, обеспечивающий передачу данных без предварительного создания соединения между ними. Этот протокол является ненадёжным. В нём пакеты могут не только не дойти, но и прийти не по порядку или вовсе продублироваться.

Основное преимущество UDP протокола заключается в скорости доставки данных. Именно поэтому чувствительные к сетевым задержкам приложения часто используют этот тип передачи данных.

### FTP — File Transfer Protocol

Протокол передачи файлов. Его использовали ещё в 1971 году — задолго до появления протокола IP. На текущий момент этим протоколом пользуются при удалённом доступе к хостингам. FTP является надёжным протоколом, поэтому гарантирует передачу данных.

Этот протокол работает по принципу клиент-серверной архитектуры. Пользователь проходит аутентификацию (хотя в отдельных случаях может подключаться анонимно) и получает доступ к файловой системе сервера.

### DNS

Это не только система доменных имён (Domain Name System), но и протокол, без которого эта система не смогла бы работать. Он позволяет клиентским компьютерам запрашивать у DNS-сервера IP-адрес какого-либо сайта, а также помогает обмениваться базами данных между серверами DNS. В работе этого протокола также используются TCP и UDP.

### HTTP — HyperText Transfer Protocol

Изначально протокол передачи HTML-документов. Сейчас же он используется для передачи произвольных данных в интернете. Он является протоколом клиент-серверного взаимодействия без сохранения промежуточного состояния. В роли клиента чаще всего выступает веб-браузер, хотя может быть и, например, поисковый робот. Для обмена информацией протокол HTTP в большинстве случаев использует TCP/IP.

HTTP имеет расширение HTTPS, которое поддерживает шифрование. Данные в нём передаются поверх криптографического протокола TLS.

### NTP – Network Time Protocol

Не все протоколы передачи нужны для обмена классического вида информацией. NTP — протокол для синхронизации локальных часов устройства со временем в сети. Он использует [алгоритм Марзулло](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9C%D0%B0%D1%80%D0%B7%D1%83%D0%BB%D0%BB%D0%BE). Благодаря нему протокол выбирает более точный источник времени. NTP работает поверх UDP — поэтому ему удаётся достигать большой скорости передачи данных. Протокол достаточно устойчив к изменениям задержек в сети.

Последняя версия NTPv4 способна достигать точности 10мс в интернете и до 0,2мс в локальных сетях.

### SSH – Secure SHell

Протокол для удалённого управления операционной системой с использованием TCP. В SSH шифруется весь трафик, причём с возможностью выбора алгоритма шифрования. В основном это нужно для передачи паролей и другой важной информации.

Также SSH позволяет обрабатывать любые другие протоколы передачи. Это значит, что кроме удалённого управления компьютером, через протокол можно пропускать любые файлы или даже аудио/видео поток.

SSH часто применяется при работе с хостингами, когда клиент может удалённо подключиться к серверу и работать уже оттуда.

## IP-адресация

Каждый компьютер в сети TCP/IP имеет адреса трех уровней:

* Локальный адрес узла, определяемый технологией, с помощью которой построена отдельная сеть, в которую входит данный узел. Для узлов, входящих в локальные сети - это МАС-адрес сетевого адаптера или порта маршрутизатора, например, 11-А0-17-3D-BC-01. Эти адреса назначаются производителями оборудования и являются уникальными адресами, так как управляются централизовано. Для всех существующих технологий локальных сетей МАС-адрес имеет формат 6 байтов: старшие 3 байта - идентификатор фирмы производителя, а младшие 3 байта назначаются уникальным образом самим производителем. Для узлов, входящих в глобальные сети, такие как Х.25 или frame relay, локальный адрес назначается администратором глобальной сети.
* IP-адрес, состоящий из 4 байт, например, 109.26.17.100. Этот адрес используется на сетевом уровне. Он назначается администратором во время конфигурирования компьютеров и маршрутизаторов. IP-адрес состоит из двух частей: номера сети и номера узла. Номер сети может быть выбран администратором произвольно, либо назначен по рекомендации специального подразделения Internet (Network Information Center, NIC), если сеть должна работать как составная часть Internet. Обычно провайдеры услуг Internet получают диапазоны адресов у подразделений NIC, а затем распределяют их между своими абонентами.

Номер узла в протоколе IP назначается независимо от локального адреса узла. Деление IP-адреса на поле номера сети и номера узла - гибкое, и граница между этими полями может устанавливаться весьма произвольно. Узел может входить в несколько IP-сетей. В этом случае узел должен иметь несколько IP-адресов, по числу сетевых связей. Таким образом IP-адрес характеризует не отдельный компьютер или маршрутизатор, а одно сетевое соединение.

* Символьный идентификатор-имя, например, SERV1.IBM.COM. Этот адрес назначается администратором и состоит из нескольких частей, например, имени машины, имени организации, имени домена. Такой адрес, называемый также DNS-именем, используется на прикладном уровне, например, в протоколах FTP или telnet.

IP-адрес имеет длину 4 байта и обычно записывается в виде четырех чисел, представляющих значения каждого байта в десятичной форме, и разделенных точками, например:

128.10.2.30 - традиционная десятичная форма представления адреса,

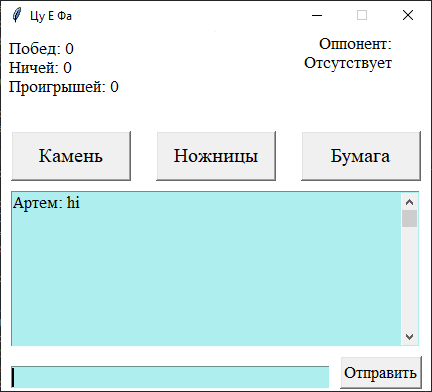
10000000 00001010 00000010 00011110 - двоичная форма представления этого же адреса.

## 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.

# Интерфейс приложения



(Рис1) Интерфейс игры.

# Код программы

## Серверная часть приложения

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", 1337).listen()

## Клиентская часть приложения

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"Артем: {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:\nОтсутствует",

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="#AFEEEE"

)

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="#AFEEEE"

)

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("Цу Е Фа")

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", 1337)

)

socket\_thread.start()

game\_thread.run()

# Выводы

В результате выполнения курсовой работы были изучены основные понятия сетевых компонентов: протокол связи, IP-адресация, маршрутизация TCP/IP. Была произведена настройка протокола транспортного уровня и настройка сокетов клиентской и серверной части приложения.

Также разработана сама игра “Цу Е Фа” с чатом на языке программирования Python для игры по локальной сети.

Изучение данных технологий поможет в дальнейшем привести к разработке функциональных приложений.

# Список литературы

1. Курс «Компьютерные сети» [Электронный ресурс] URL: <http://math.gsu.by/wp-content/uploads/courses/networks/построение_больших_сетей>
2. Статья «Протоколы передачи данных» [Электронный ресурс] URL: <https://tproger.ru/explain/protokoly-peredachi-dannyh-chto-jeto-kakie-byvajut-i-v-chjom-razlichija/>
3. Понимание основ создания адресов TCP/IP и организации подсетей [Электронный ресурс] URL: <https://learn.microsoft.com/ru-ru/troubleshoot/windows-client/networking/tcpip-addressing-and-subnetting>
4. Что такое TCP/IP и как работает этот протокол [Электронный ресурс] URL:

<https://timeweb.com/ru/community/articles/chto-takoe-tcp-ip>

1. Документация IBM «Маршрутизация TCP/IP» [Электронный ресурс] URL: <https://www.ibm.com/docs/ru/aix/7.1?topic=protocol-tcpip-routing>
2. Адресация в IP-сетях [Электронный ресурс] URL:

<https://www.opennet.ru/docs/RUS/ip_network/glava_3.html>