Оглавление

Цель работы:

Смоделировать программу, осуществляющую передачу видеопотока на основе алгоритмов tcp и udp.

Описание протоколов:

UDP

Протокол UDP (User Datagram Protocol, RFC-768) является одним из основных протоколов, расположенных непосредственно над IP. Он предоставляет прикладным процессам транспортные услуги, немногим отличающиеся от услуг протокола IP. Протокол UDP обеспечивает доставку дейтограмм, но не требует подтверждения их получения. То есть, доставка сообщений не гарантирована.

Время определяется только объемом передаваемых данных. Чтобы данные были доставлены за минимальное время, добавляемый заголовок должен быть минимальной длины.

Заголовок состоит из следующих данных:

- Порт получателя
- Порт источника
- Размер данных
- Контрольная сумма (заголовок + данные, либо только заголовок)

TCP

Протокол TCP (Transmission Control Protocol, RFC 793) - один из основных протоколов передачи данных интернета. Предназначен для управления передачей данных интернета. Пакеты в TCP называются сегментами. В стеке протоколов TCP/IP выполняет функции транспортного уровня модели OSI.

Основная идея tcp – при получении сообщения, посылается квитанция.

Если данные доставлены, то ошибок гарантировано нет.

Задержки между пакетами не постоянны, случайны и зависят от числа промежуточных узлов (на маршрут мы никак не можем повлиять) и от загруженности сети.

Заголовок состоит из следующих данных:

- Порт источника
- Порт получателя
- Порядковый номер (позволяет контролировать порядок сообщений)

- Номер подтверждения (Когда сообщение содержит флаг ACK, то значение в номере подтверждения должно соответствовать следующему порядковому номеру (SYN), которое отправитель сообщения с флагом ACK ожидает получить от передающей системы)
- Длина заголовка
- Резерв (зарезервировано для будущего использования)
- Флаги или управляющие биты
- Размер окна приема
- Контрольная сумма
- Указатель срочности

Описание программы:

Программа написана на языке Python с использованием библиотеки socket, и состоит из четырех ключевых модулей:

- Tcp_server
  Представляет собой сервер для отправки и приема tcp сообщений.
- Tcp_client
  Представляет собой клиента tcp, который подключается к серверу и получает от него файл
- Udp_server
  Представляет собой сервер для отправки и приема udp сообщений.
- Udp_client
  Представляет собой клиента udp, который подключается к серверу и получает от него файл

Описание работы реализованной программы:

TCP

1. Сначала запускается сервер, он ожидает подключений клиентов (слушает указанный порт);
2. После запускается клиент (если запустить сначала клиента, не запустив предварительно сервер, то программа не сработает);
3. Клиент подключается к указанному адресу и порту;
4. Сервер принимает подключение клиента;
5. Сервер в цикле отправляет клиенту файл кусками по 4096 бит.
6. Сервер дожидается

UDP

1. Сначала запускается сервер, он ожидает подключений клиентов (слушает указанный порт);
2. После запускается клиент (если запустить сначала клиента, не запустив предварительно сервер, то программа, как и с tcp версией, не сработает);
3. С клиента отправляется «приветственное» сообщение для создания подключения;
4. Сервер принимает на порт «приветственное» сообщение клиента, создавая с ним связь;
5. Сервер отправляет файл клиенту файл в цикле кусочками по 4096 бит.

Список ограничений, введенный при разработке:

В связи с работой на ос windows брандмауэра, не удалось осуществить работу серверной части на данной ос. Решением данной проблемы был запуск серверной части на Unix подобных системах (в частности, ос Manjaro). При этом на клиентскую часть приложения данные ограничения не распространяются.

TCP

Максимальное количество подключенных клиентов = 1

Файл считывается кусочками по 4096 бит

Зарезервированный порт - 9001

UDP

Файл считывается кусочками по 4096 бит

Зарезервированный порт - 9001

Описание выполненного эксперимента:

Локально, то есть сервер и клиент находятся на одном устройстве:

```
"C:\study\7 сем\Инфокомуникационные системы и сети\
DESKTOP-PGH926V
TCP server DESKTOP-PGH926V (ip: 192.168.56.1)
waiting connect
('192.168.56.1', 51468) connected
file kz720p.mp4     size = 18336583 bytes
sent in 0.014967918395996094 seconds

Process finished with exit code 0
```

Рисунок 1. Результат работы tcp сервера на одном устройстве.

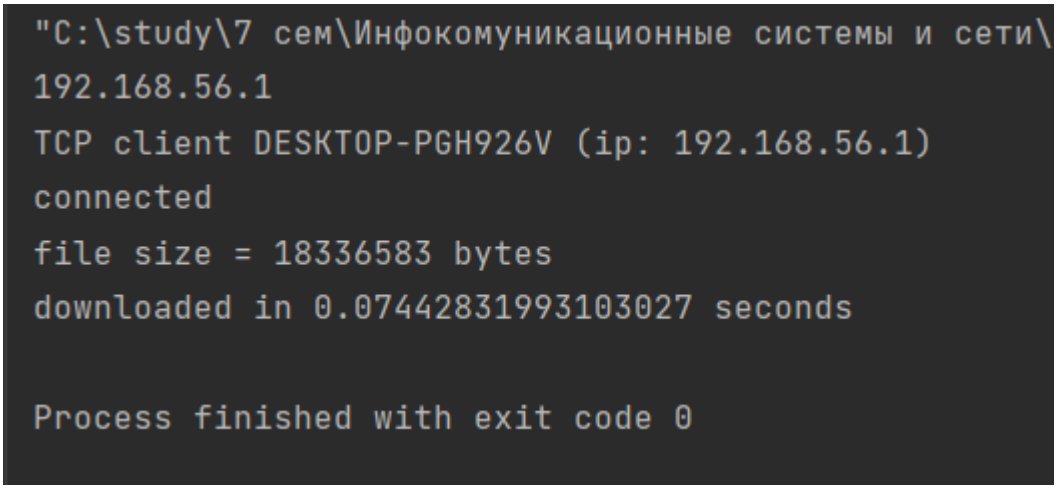

Рисунок 2. Результат работы tcp клиента на одном устройстве.

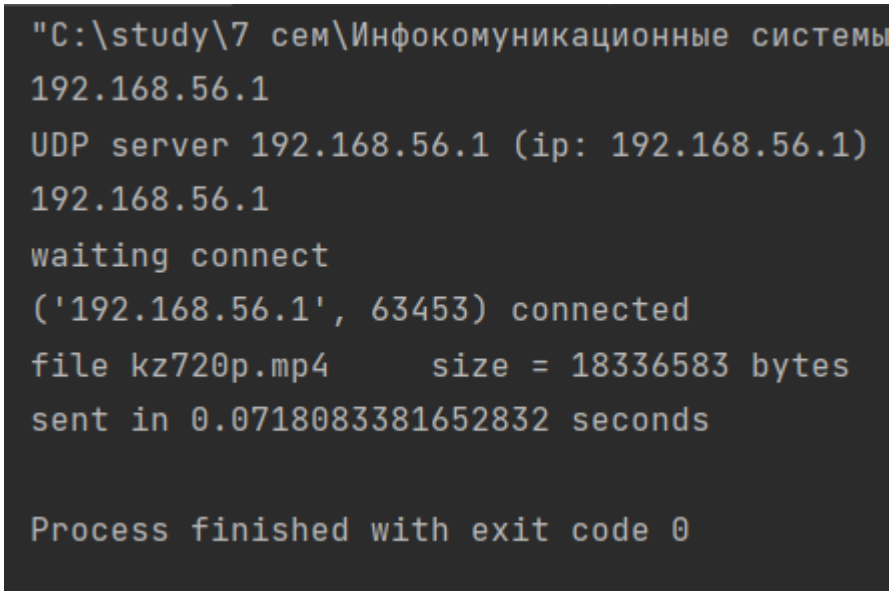

Рисунок 3. Результат работы udp сервера на одном устройстве.

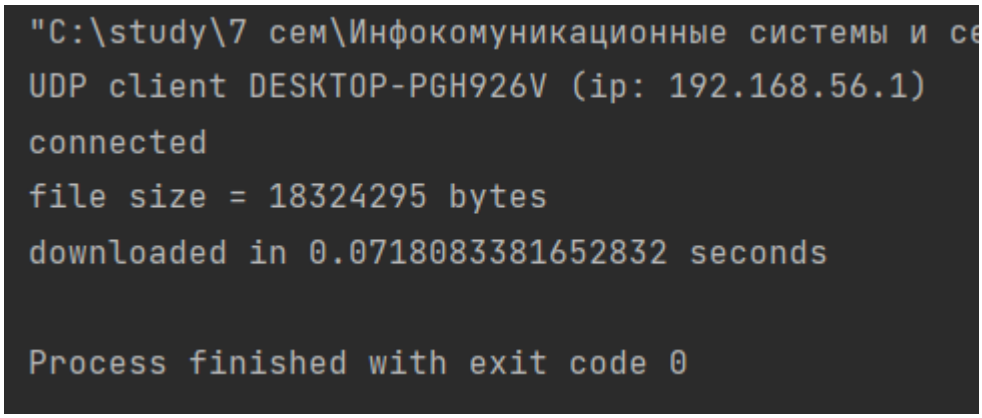

Рисунок 4. Результат работы udp клиента на одном устройстве..

Сервер и клиент находятся на разных устройствах:

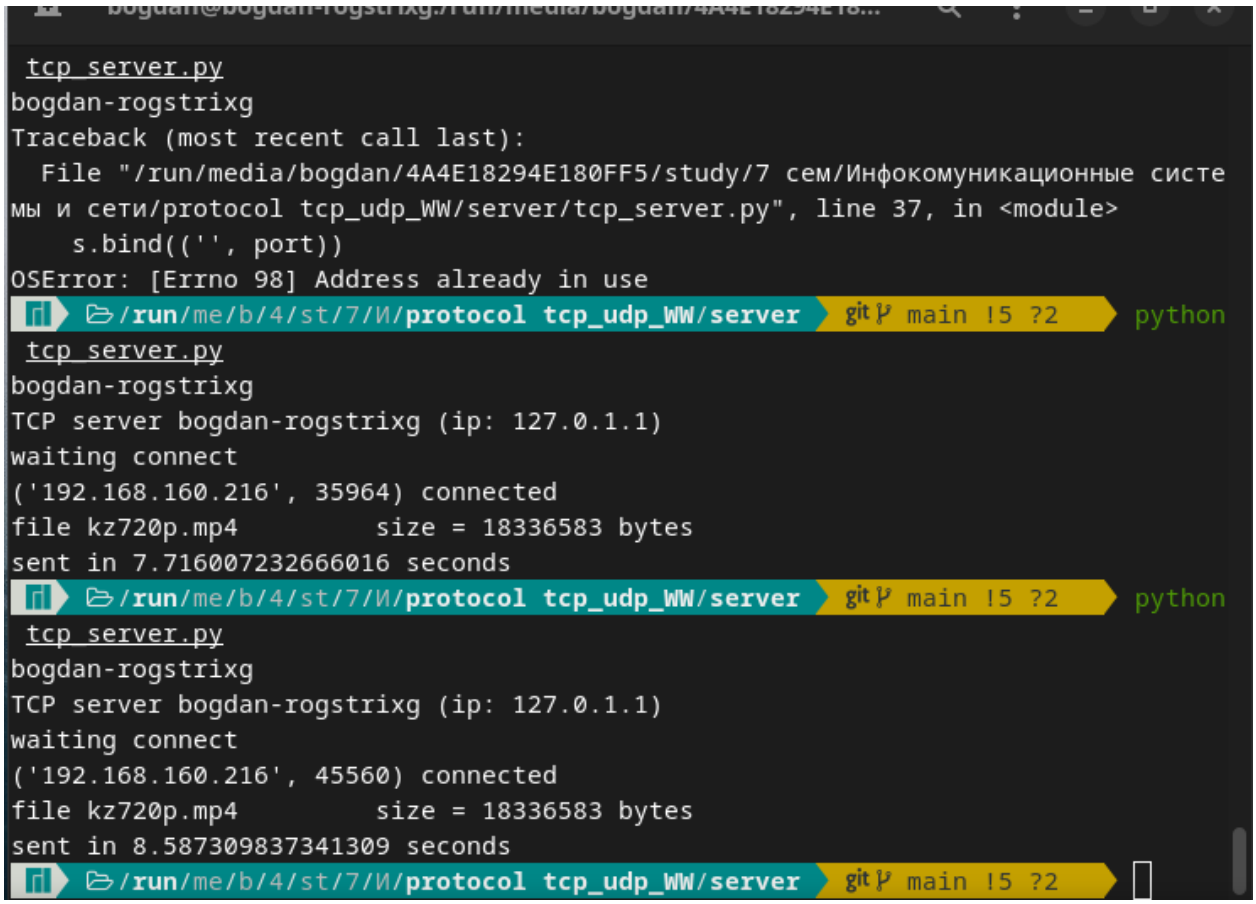

Рисунок 5. Результат работы tcp сервера на разных устройствах.

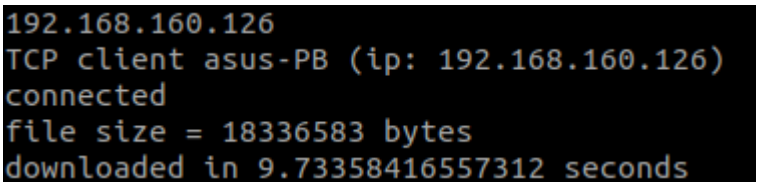

Рисунок 6. Результат работы tcp клиента на разных устройствах.

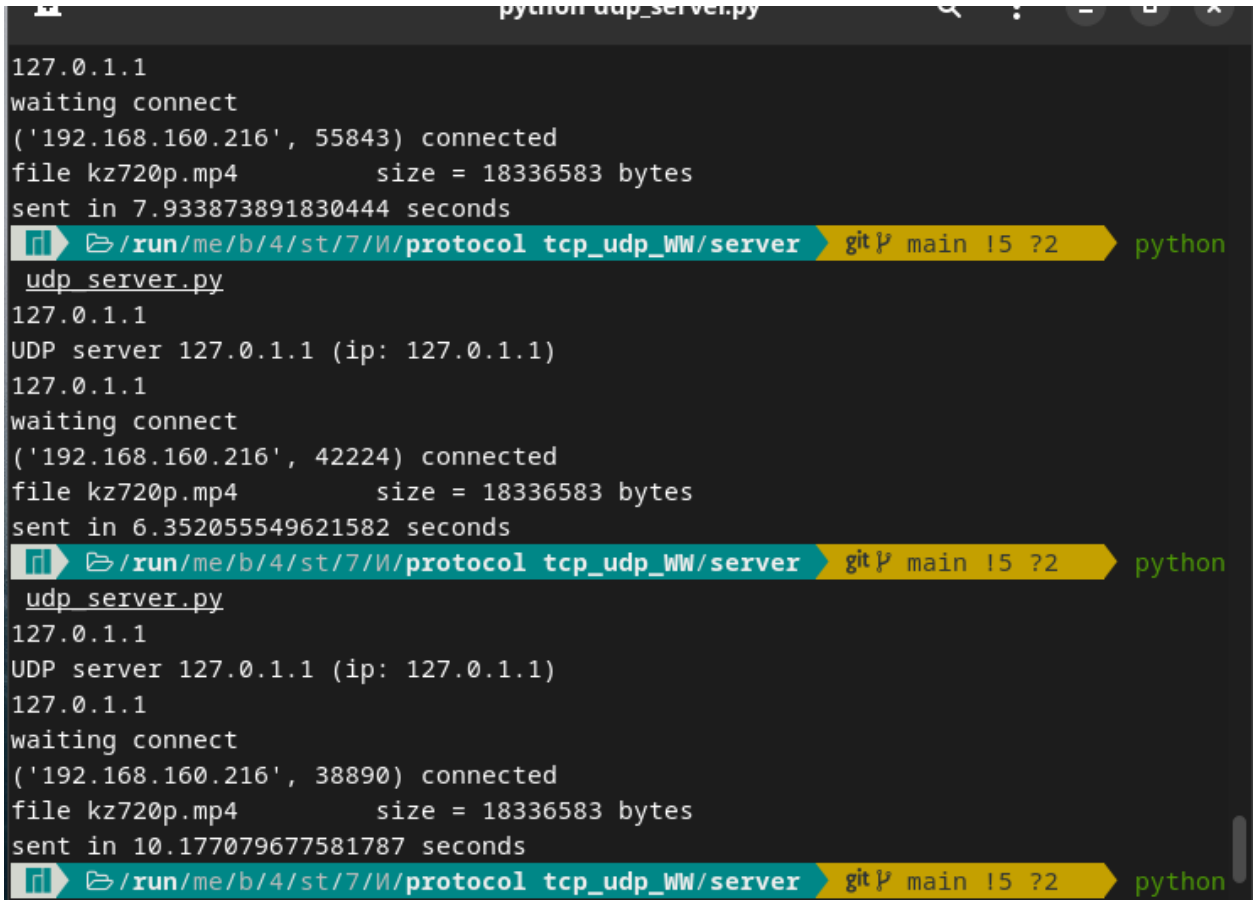

Рисунок 7. Результат работы udp сервера на разных устройствах.

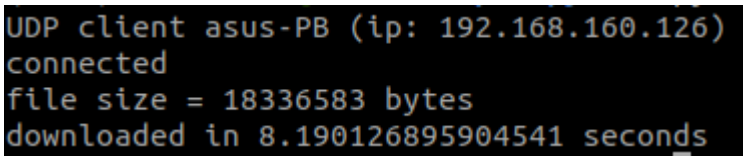

Рисунок 8.Результат работы udp клиента на разных устройствах.

Таблица 1. Сравнение передач по протоколам tcp и udp.

Номер эксперимента	Используемый протокол	Время отправления (секунд)	Время приема (секунд)	Процент принятых пакетов	Размер файла (байт)
1	Tcp	8.58	9.73	100 %	18336583
	Udp	6.35	8.19	99.2 %	
2	Tcp	3.83	4.5	100 %	18336583
	Udp	4.29	5.27	92.32 %	
3	Tcp	7.25	7.53	100 %	18336583
	Udp	3.85	4.73	75.71 %	

Эксперимент поставлен следующим образом: телефон был переведен в режим точки доступа (раздает сигнал wi-fi) и к данной точке доступа были подключены только два устройства: сервер (компьютер с серверным кодом) и сервер (компьютер с клиентским кодом)

Вывод:

Таким образом, в результате выполнения данной курсовой работы были реализованы алгоритмы передачи видео трафика по TCP протоколу и передачи видео трафика по UDP протоколу. В время выполнения данной работы были детально изучены TCP и UDP протоколы, а также проведены измерительные тесты, по результатам которых можно сделать следующие выводы:

- TCP протокол хорошо подходит для передачи записанного видео файла, так как файл доходит в целости и подлежит корректному воспроизведению на стороне получателя.
- UDP протокол не подходит для передачи записанного видео файла, так как файл не доходит в целостности до получателя из-за частичной потери пакетов, и не подлежит корректному воспроизведению на стороне получателя
- Локально программа работает значительно быстрее, нежели на разных устройствах
- UDP не всегда гарантирует более быструю доставку, по крайней мере в результатах данной работы.

Листинг кода

TCP server

```
import os
import socket
import time

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = socket.gethostname()
print(host)
port = 9001
s.bind(('', port))
s.listen(3)
print(f"TCP server {host} (ip: {socket.gethostbyname(host)})")
print("waiting connect")

conn, addr = s.accept()
print(addr, "connected")
file_name = "kz720p.mp4"  # input("enter file name: ")
file = open(file_name, "rb")

start_time = time.time()
while True:
    file_data = file.read()
    conn.send(file_data)
    if not file_data:
        break
end_time = time.time()
conn.close()

print(f"file {file_name} \tsize = {os.stat(file_name).st_size} bytes"
      f"\nsent in {end_time - start_time} seconds")
file.close()
```

TCP client

```
import os
import socket
import time


def get_my_ip():
    print(socket.gethostname())
    print(socket.gethostbyname(socket.gethostname()))


tcp_client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = socket.gethostbyname(socket.gethostname())  # input("enter host address of sender: ") #
enter needed host
# host = "172.20.10.3"
print(host)
port = 9001
```

```
print(f"TCP client {socket.gethostname()} (ip: {host})")
tcp_client_socket.connect((host, port))
print("connected")
file_name = "tcp_received.mp4"  # input("enter inner file name: ")
file = open(file_name, "wb")
start_time = time.time()
while True:
    file_data = tcp_client_socket.recv(4096)
    file.write(file_data)
    if not file_data:
        break
end_time = time.time()
file.close()

print(f"file size = {os.stat(file_name).st_size} bytes"
      f"\ndownloaded in {end_time - start_time} seconds")
```

UDP server

```
import os
import socket
import time

udp_server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
host = socket.gethostbyname(socket.gethostname())
print(host)
port = 9001
udp_server_socket.bind(('', port))
# udp_server_socket.listen(1)

print(f"UDP server {host} (ip: {socket.gethostbyname(host)})")
print(host)
print("waiting connect")
connect_message, addr = udp_server_socket.recvfrom(4096)
print(addr, "connected")
file_name = "kz720p.mp4"  # input("enter filename: ")
file = open(file_name, "rb")
start_time = time.time()
while True:
    file_data = file.read(4096)
    # conn.sendto(file_data)
    udp_server_socket.sendto(file_data, addr)
    if not file_data:
        udp_server_socket.sendto(file_data, addr)
        end_time = time.time()
        break
# conn.close()

udp_server_socket.close()
print(f"file {file_name} \tsize = {os.stat(file_name).st_size} bytes"
      f"\nsent in {end_time - start_time} seconds")
```

UDP client

```
import os
import socket
import time


def get_my_ip():
    print(socket.gethostname())
    print(socket.gethostbyname(socket.gethostname()))


udp_client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
host = socket.gethostbyname(socket.gethostname())  # input("enter host address of sender: ") #
enter needed host
# host = "192.168.56.1"
port = 9001

print(f"UDP client {socket.gethostname()} (ip: {host})")
udp_client_socket.sendto(b"connect", (host, port))
print("connected")

file_name = "udp_received.mp4"  # input("enter incoming filename: ")
file = open(file_name, "wb")
start_time = time.time()
while True:
    file_data, adr = udp_client_socket.recvfrom(4096)
    file.write(file_data)
    if not file_data:
        end_time = time.time()
        break
file.close()
print(f"file size = {os.stat(file_name).st_size} bytes"
      f"\ndownloaded in {end_time - start_time} seconds")
```