Санкт-Петербургский Политехнический Университет Петра Великого

Институт Компьютерных Наук и Технологий

Высшая Школа Интеллектуальных Систем и Суперкомпьютерных Технологий

**Отчёт**

По лабораторным работам

**Дисциплина:** Технологии компьютерных сетей

**Работу выполнил студент**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сергеев И.К.

Группа 3530901/60201

**Преподаватель**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алексюк А.О.

г. Санкт-Петербург

2019

Оглавление

[1) TCP чат с использованием потоков 3](#_30j0zll)

[2) TCP чат с использованием Poll 4](#_1fob9te)

[3) UDP (DNS) 5](#_3znysh7)

[Описание задания 5](#_2et92p0)

[Структура DNS-сообщения 5](#_tyjcwt)

[Описание проекта 8](#_3dy6vkm)

[4) Индивидуальное задание: Система дистанционного тестирования 8](#_1t3h5sf)

[Описание задания 8](#_4d34og8)

[Протокол взаимодействия 9](#_2s8eyo1)

[Описание проекта 10](#_17dp8vu)

[Сетевые утилиты 10](#_3rdcrjn)

[1)](#_26in1rg) Утилита ipconfig 10

[2)](#_lnxbz9) Утилита Wireshark 11

[3)](#_35nkun2) Утилита ping 12

[Выводы 13](#_1ksv4uv)

[Приложение 13](#_44sinio)

# TCP чат с использованием потоков

**Описание Системы**

Система представляет из себя серверное и клиентское приложения, обменивающиеся текстовыми сообщениями посредством TCP протокола. Обмен сообщениями происходит в соответствии со следующими правилами:

1. Каждое сообщение клиента содержит заголовок длиной 4 байта, в котором хранится размер оставшейся части посылки.
2. Формат сообщения: <time>[nickname] text
3. Сообщения форматируются на стороне клиента
4. Посылка после заголовка содержит форматированное сообщения.

**Клиент**

Программа-клиент написана на языке С. Исполняемому файлу в качестве аргументов командной строки подаются: адрес сервера, его порт и никнейм пользователя.

Программа выполняется в нескольких потоках:

* Основной:
  + С помощью системной функции socket открывает сокет, необходимый для подключения к серверу.
  + Устанавливает TCP соединение, используя функцию connect и созданный ранее сокет.
  + Запускает поток записи пользовательских сообщений в сокет.
  + Запускает поток чтения сообщений сервера.
  + Ожидает завершения запущенных потоков.
* Слушающий сокет:
  + Прослушивает сокет
  + С помощью функции read считывает сначала заголовок посылки, содержащий размер остальной части посылки.
  + Затем считывает оставшуюся часть посылки (сообщение от другого клиента) и добавляет её в циклический буфер сообщений.
  + Вызывает функцию вывода буфера сообщений в терминал.
* Пишущий в сокет:
  + При запуске однократно спрашивает имя клиента и отправляет его на сервер
  + Далее в цикле считывает сообщения клиента и отправляет их на сервер

**Сервер**

Программа-сервер написана на языке С.

Программа выполняется в нескольких потоках:

* Основной:
  + Инициализирует все необходимые мьютексы
  + Последовательно вызывает функции socket, bind, listen для настройки сокета, устанавливающего новые соединения.
  + Устанавливает новые соединения, запуская отдельный клиентский поток для каждого из них
  + При запуске единовременно читает имя клиента и записывает его в специальный массив.
* Клиентский:
  + Запускает цикл, в котором: читает длину посылки, затем читает саму посылку, добавляет к принятому от клиента тексту сообщения дату и имя пользователя, после чего записывает полученную строку в буфер сообщений.
  + Рассылает отправленное сообщение всем подключённым клиентам.

Ссылка на исходный код находится в приложении.

# TCP чат с использованием Poll

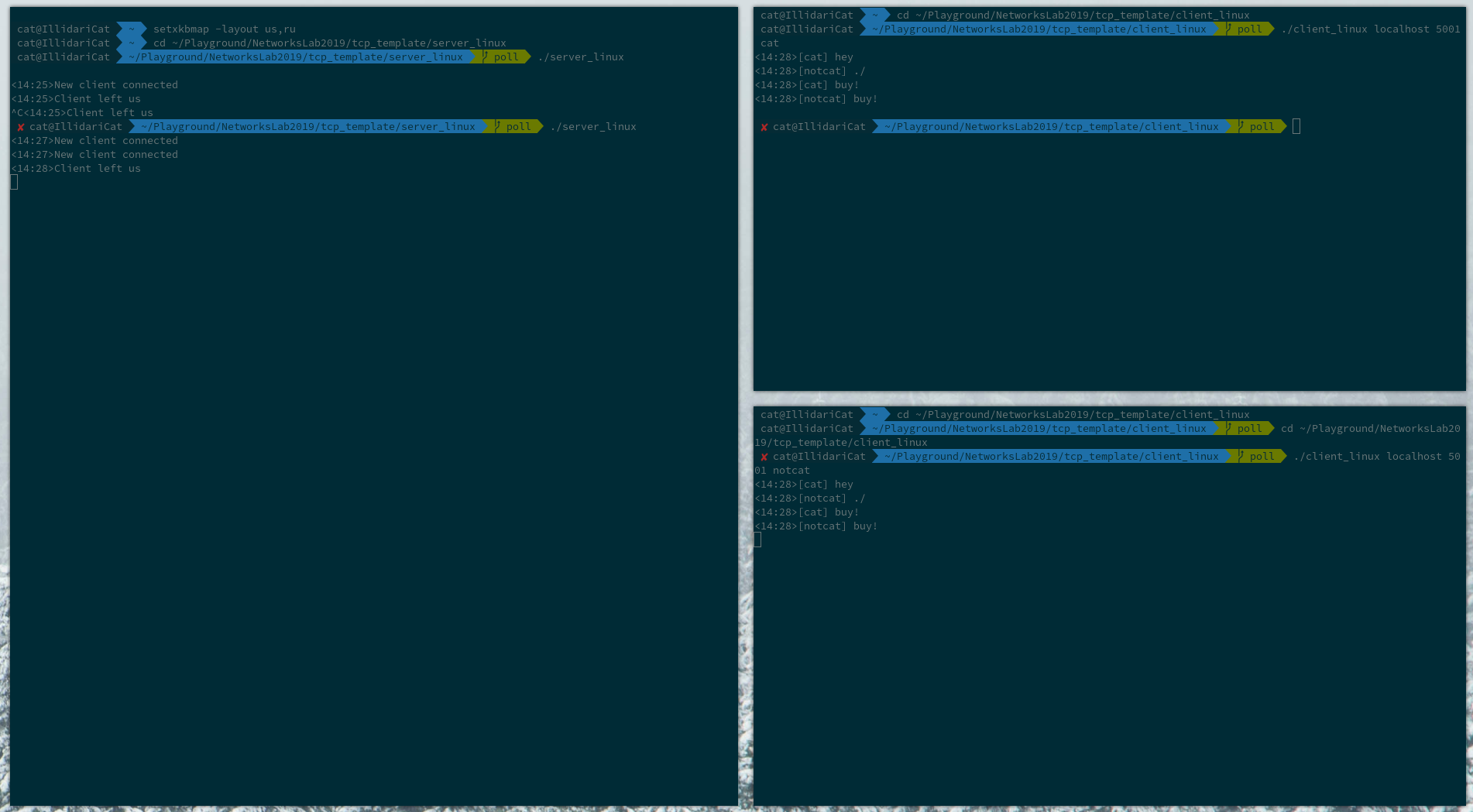
**Описание системы**

Данная система работает по тому же протоколу, что и система из предыдущего раздела. Разница заключается в реализации – сервер данного раздела реализован с помощью неблокирующих сокетов.

**Сервер**

Программа-сервер написана на языке С. Исполняемому файлу в качестве аргумента командной строки передаётся номер порта для прослушивания. Благодаря использованию poll, сервер работает в одном потоке, последовательно обрабатывая все события. В цикле обработки выполняется итерация по всем наблюдаемым дескрипторам с целью определения возникновения события. Если событие связано с дескриптором основного сокета, выполняется процедура создания соединения с новым клиентом. В противном случае выполняется процедура чтения сообщения от уже подключённого клиента.

Ссылка на исходный код находится в приложении.  
Демонстрация работы:



# UDP (TFTP)

## Описание задания

В данном задании был реализован TFTP-клиент и TFTP-сервер на основе реального TFTP протокола через UDP. Программа была реализована на языке javaScript на базе Node.js. Использовались следующие сетевые вызовы:

* Создание UDP-сокета

const server = udp.createSocket('udp4');

* Создание слушающего UDP-сокета

server.bind(3000);

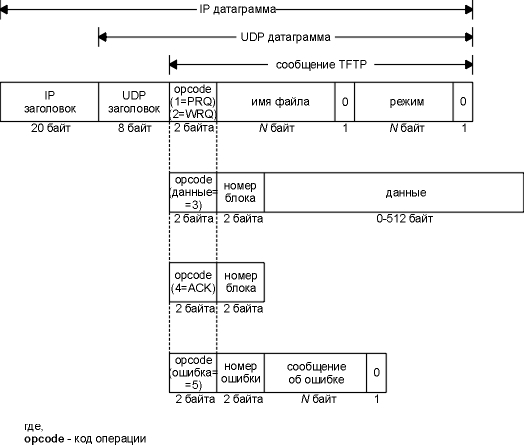
* Отправка сообщения

this.socket.send(new Packet({ type: Packet.types.ack, number }).build(), this.info.port, this.info.address, rs);

* Чтение сообщения

server.on('message', (msg, rinfo) => { /\*...\*/ });

## Структура TFTP-сообщения



## Описание проекта

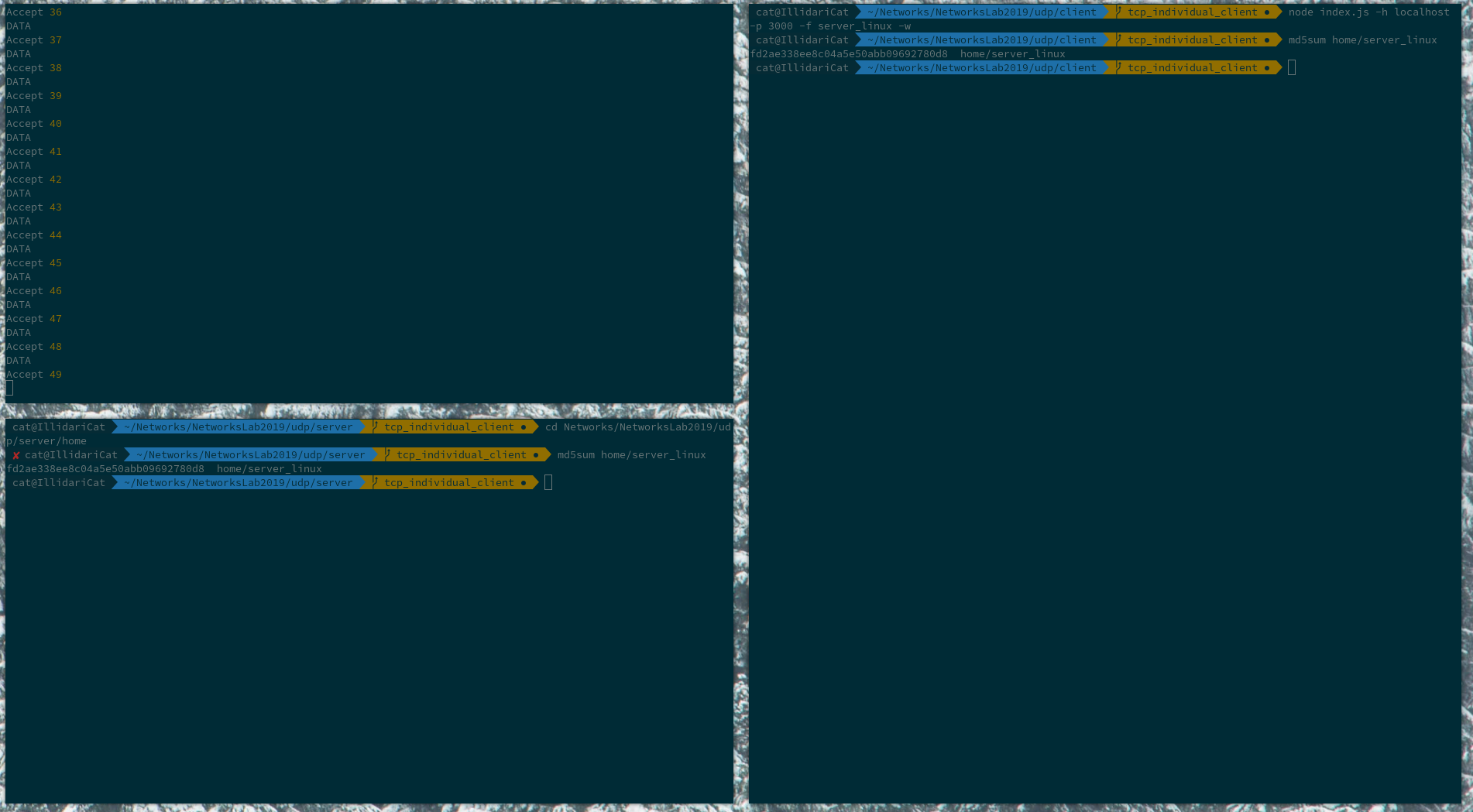
1. Работа сервера

Сервер обрабатывает WQR (запросы на запись файл на сервер) и RRQ (запросы на чтение файлов с сервера).

1. Работа клиента

Клиент выполнен в виде программы с аргументами командной строки. Поддерживает WRQ и RRQ запросы.

Демонстрация работы:



# Индивидуальное задание: Система дистанционного тестирования

## Описание задания

Разработать клиент-серверную систему дистанционного тестирования знаний, состоящую из централизованного сервера тестирования и клиентов тестирования.

***Основные возможности.***

Серверное приложение должно реализовывать следующие функции:

1) Прослушивание определенного порта

2) Обработка запросов на подключение по этому порту от клиентов

3) Поддержка одновременной работы нескольких клиентов через механизм нитей

4) Регистрация клиента, выдача клиенту результата его последнего теста, выдача клиенту списка тестов

5) Получение от клиента номера теста

6) Последовательная выдача клиенту вопросов теста и получение ответов на вопросы

7) После прохождения теста – выдача клиенту его результата

8) Обработка запроса на отключение клиента

9) Принудительное отключение клиента

Клиентское приложение должно реализовывать следующие функции:

1) Установление соединения с сервером

2) Посылка регистрационных данных клиента

3) Выбор теста

4) Последовательная выдача ответов на вопросы сервера

5) Индикация результатов теста

6) Разрыв соединения

7) Обработка ситуации отключения клиента сервером

Настройки приложений.

Разработанное клиентское приложение должно предоставлять пользователю настройку IP-адреса или доменного имени удалённого сервера тестов и номера порта, используемого сервером.

Разработанное серверное приложение должно хранить вопросы и правильные ответы нескольких тестов.

Методика тестирования.

Для тестирования приложений запускается сервер «Удаленного тестирования» и несколько клиентов. В процессе тестирования проверяются основные возможности сервера по параллельному тестированию нескольких пользователей.

## Протокол взаимодействия

За основу протокола взаимодействия между клиентом и сервером был принят JSON-RPC - протокол удалённого вызова процедур, использующий JSON для кодирования сообщений.  
Пример сообщения:

|  |
| --- |
| {  "action": "tests",  "params": {  "tests": [  {  "title": "Physical test",  "id": "a82ad2e1931f1ca4ea9f12e572202cd8"  },  {  "title": "Random test",  "id": "4b7fa75c02babed45914aec65d04caf2"  }  ],  "last": {  "title": "Random test",  "result": 0.3333333333333333  }  }  } |

Всё взаимодействие представляет из себя своего рода конечный автомат из состояний, в которых могут находиться клиент и сервер, обмениваясь сообщениями.  
Протоколом транспортного уровня для этого проекта является TCP.

## Описание проекта

Проект представляет из себя:

* клиентское десктопное приложение, написанное нa Electron-Vue, работающее как с сервером, созданным в рамках данного проекта, так и с сервером, написанным коллегой (Графовым Денисом) на Dart.
* сервер, написанный на языке JS на базе Node.js. Сервер также способен работать с клиентом Графова Дениса, написанным на Flutter.

# Сетевые утилиты

# Утилита ipconfig

**Описание**

Утилита ifconfig предназначена для настройки параметров сетевого интерфейса и назначения сетевого адреса компьютера.

В случае если не передаются никакие параметры, то утилита ifconfig показывает текущую конфигурацию сетевого интерфейса.

**Использование команды без аргументов:**

docker0 Link encap:Ethernet HWaddr 02:42:5c:85:9c:4d

inet addr:172.17.0.1 Bcast:172.17.255.255 Mask:255.255.0.0

inet6 addr: fe80::42:5cff:fe85:9c4d/64 Scope:Link

UP BROADCAST MULTICAST MTU:1500

RX packets:146928 errors:0 dropped:0 overruns:0 frame:0

TX packets:295551 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:0

RX bytes:10838636 (10.3 MiB) TX bytes:400001584 (381.4 MiB)

eno1 Link encap:Ethernet HWaddr ec:f4:bb:09:91:2c

UP BROADCAST MULTICAST MTU:1500

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

Interrupt:20 Memory:f7d00000-f7d20000

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

inet6 addr: ::1/128 Scope:Host

UP LOOPBACK RUNNING MTU:65536

RX packets:387322 errors:0 dropped:0 overruns:0 frame:0

TX packets:387322 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:237253752 (226.2 MiB) TX bytes:237253752 (226.2 MiB)

wg0 Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00

inet addr:203.0.113.2 P-t-P:203.0.113.2 Mask:255.255.255.0

UP POINTOPOINT RUNNING NOARP MTU:1420

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:298 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

wlp3s0 Link encap:Ethernet HWaddr 40:f0:2f:c4:86:31

inet addr:10.215.5.41 Bcast:10.215.7.255 Mask:255.255.252.0

inet6 addr: fe80::6c7a:ea67:245b:b91a/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500

RX packets:12617207 errors:0 dropped:2194 overruns:0 frame:0

TX packets:2821763 errors:0 dropped:0 overruns:0 carrier:0

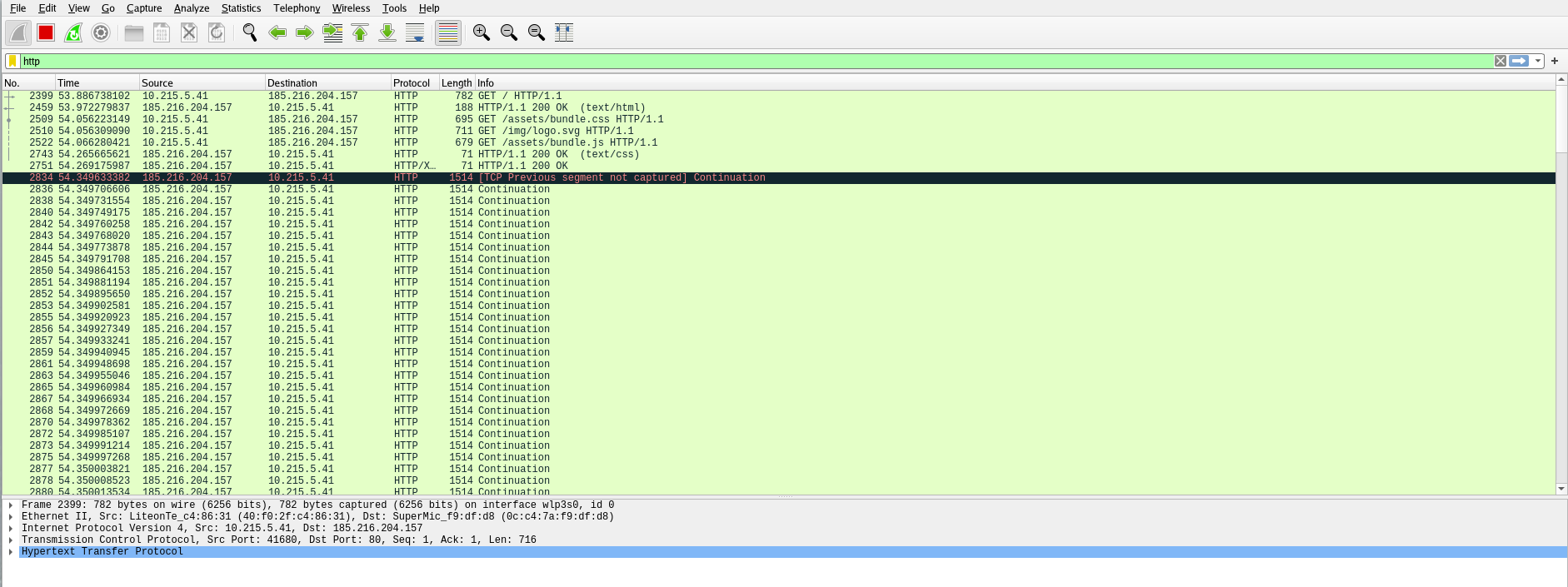
collisions:0 txqueuelen:1000

RX bytes:16910433344 (15.7 GiB) TX bytes:325160434 (310.0 MiB)

# Утилита Wireshark

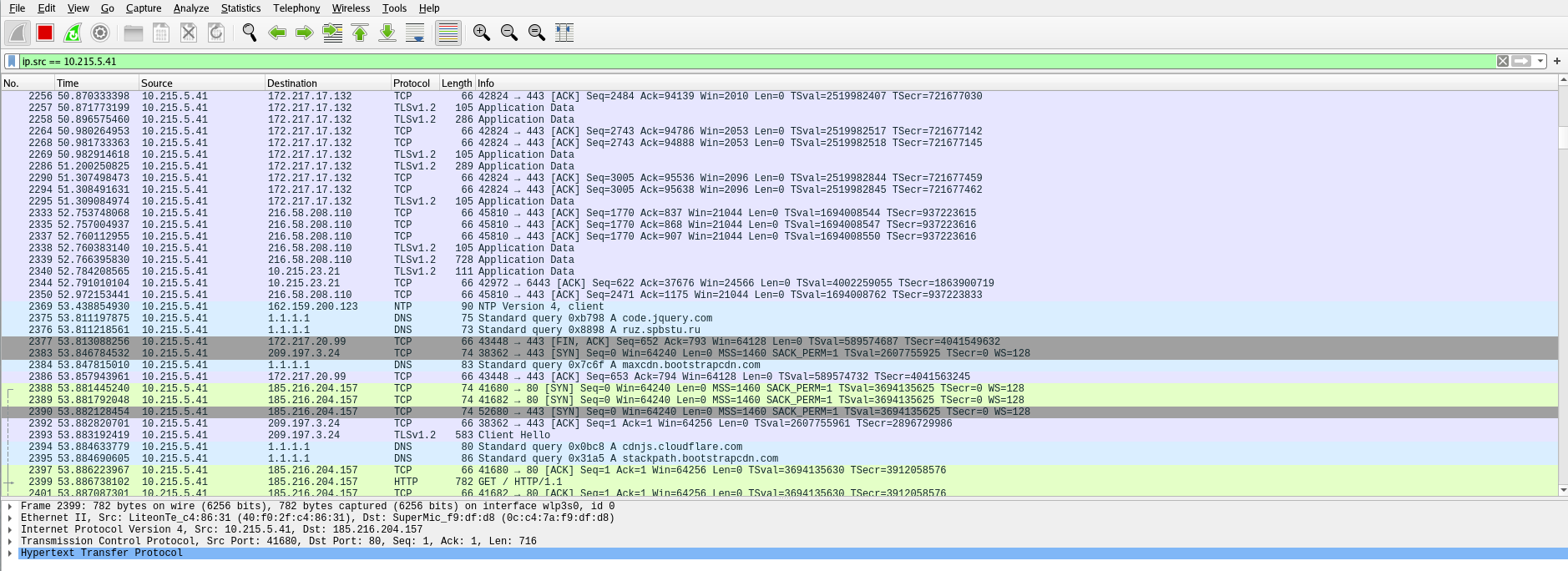
Wireshark — известный инструмент для захвата и анализа сетевого трафика, который используется для исследования сетевых приложений и протоколов, нахождения проблем в работе сети и выяснения причины этих проблем.

**Фильтр по протоколу http:**



Выбрав пакет http, можно заметить, что HTTP инкапсулируется в TCP (транспортный уровень), TCP инкапсулируется в IP (сетевой уровень), а IP в свою очередь инкапсулируется в Ethernet.

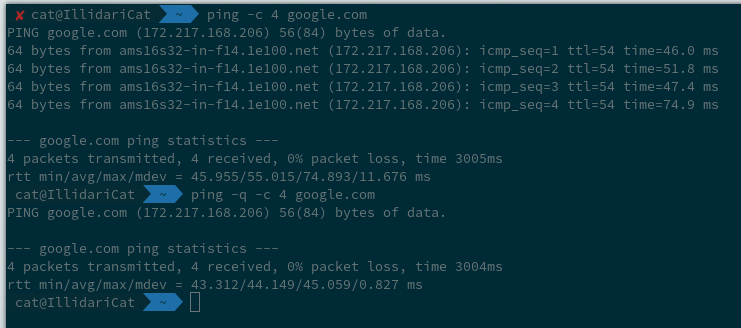
**Фильтр по ip адресу:**



# Утилита ping

Утилита ping используя ICMP протокол посылает сигнал ECHO\_REQUEST и ожидает ответа от удаленного хоста в виде ECHO\_RESPONSE. Когда заданное количество пакетов отправлено и получено или программа завершается с сигналом SIGINT, выводится информация о количестве отправленных, полученных и потерянных пакетов, минимальное, максимальное и среднее время ответа.

**Использование команды с флагом «-с» (ограничение количества отправленных пакетов) и с флагом «-q» (тихий режим):**



# Выводы

При проектировании высокопроизводительных сетевых приложений важно решить, какой метод обработки сетевых событий использовать. Выбор правильного метода может быть критически важной вещью для архитектуры приложения.

Неблокирующие сокеты (и использование poll) – позволяет ограничиться одним потоком обработки клиентов, что экономит память. Минусы данного подхода заключаются в следующем:

* большой объём действий, затрачиваемых на одного клиента может повысить время отклика программы для других клиентов.
* для поддержания производительности программы, работа со сторонними ресурсами (диск, база данных, …) тоже должна быть неблокирующей, либо с использованием потоков (что влечёт за собой минусы потоков, от которых мы хотели избавиться использованием poll).

Потоки (создание отдельного потока для каждого клиента) – позволяют экономить процессорные ресурсы, но при этом каждый поток расходует какое-то количество памяти, что может оказаться критичным при большом количестве клиентов.

# Приложение

**Исходные коды**

* TCP чат с использованием потоков: <https://github.com/tiger31/NetworksLab2019/tree/master/tcp_template>
* TCP чат с использованием poll: <https://github.com/tiger31/NetworksLab2019/tree/poll/tcp_template>
* UDP (TFTP): <https://github.com/tiger31/NetworksLab2019/tree/udp/udp>
* Индивидуальное задание:

<https://github.com/tiger31/NetworksLab2019/tree/tcp_individual_client/tcp>

[https://github.com/tiger31/NetworksLab2019/tree/tcp\_individual\_server/tcp](https://github.com/tiger31/NetworksLab2019/tree/tcp_individual_client/tcp)